Kratak sadržaj

Sadržaj

Posveta

Ova knjiga je posvećena živoj uspomeni na Dejvida Levina.

Kome ielim da odam priznanje

Drugo izdanje je druga šansa da odamo priznanje i da se zahvalimo onim ljudima bez čije podrške i pomoći ova knjiga ne bi mogla ni da postoji. Ti ljudi su Stacey, Robin i Rachel Liberty.

Takode, moram da se zahvalim svima onima koji su saradivali u pripremi mojih knjiga, kao i Sams-u i WRox-u, zato što su pokazali izuzetnu profesionalnost i kvalitet u radu. Urednici iz Sams-a su obavili sjajan posao. I, stoga, ja osećam potrebu da se zahvalim i odam priznanje Franhattonu Mary Ann Abramson, Greg Guntle i Cris Denny.

U poslednjih nekoliko godina ja sam držao kurs koji se bazira na ovoj knjizi i veliki broj ljudi mi je pomogao u pronalaženju i ispravljanju bagova i grešaka. Ovim ljudima puno dugujem i ovom prilikom bih se izuzetno zahvalio Gregu Newmanu, Corrinne Thompson, a takođe i Catherine Prouty i Jennifer Goldman.

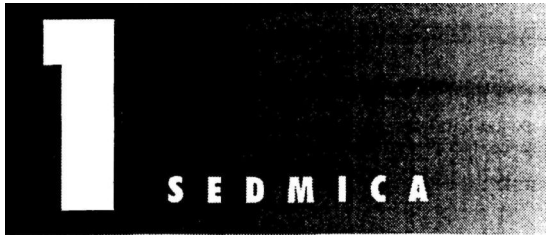
Takođe želim da odam priznanje ljudima koji su me naučili programiranju: Skipu Gilbrechu i Davidu McCuneu, kao i onima koji su me naučili C+ + , Steveu Rogersu i Stephenu Zagieboyloju. Posebno želim da se zahvalim Mikeu Kraleyu, Edu Beloveu, Patricku Johnsonu, Mikeu Rothmanu i Sangam Pand od kojih sam naučio kako se jedan projekat vodi i instalira.

Ostali koji su direktno, ili indirektno doprineli radu na ovoj knjizi su: Scott Boag, David Bogartz, Gene Broadway, Drew i Al Carlson, Frank Childs, Jim Culbert, Thomas Dobbing, James Efstratiou, David Heath, Eric Helliwell, Gizele i Ed Herlihy, Mushtaq Khalique, Matt Kingman, Steve Leland, Michael Smith, Frank Tino, Donovan White, Mark Woodbury, Wayne Wylupski i Alan Zeitchek.

Programiranje je vise posao i kreativno iskustvo, nego što je tehnika. Stoga, ja moram da odam priznanje Tomu Hottensteinu, Jayu Leveu, Davidu Rollertu, Davidu Shnaideru i Robertu Spielvogelu.

Na kraju bih želeo da se zahvalim i gospođi Kalish, koja je naučila mene i moju šestu klasu binarnoj aritmetici 1965. godine, kada ni ona ni mi nismo znali zašto.

U listinzima u ovoj knjizi svaka prava linija programskog koda je numerisana. Ako vidite nenumerisanu liniju u listingu, znaćete da je ona zapravo nastavak prethodne numerisane linije koda (neke linije koda su prevelike da bi cele mogle stati na širinu stranice). U torn slučaju, prilikom ukucavanja koda, ove dve linije unesite kao jednu; nemojte ih deliti.



Na prvi pogled

Dok se pripremate za svoju prvu nedelju učenja programiranja u programskom jeziku C+ + , biće Vam potrebno nekoliko stvari: prevodilac, editor i ova knjiga. Ukoliko nemate C + + prevodilac i editor, još uvek možete koristiti knjigu, ali bez obavljanja vežbi nećete ostvariti maksimalni napredak.

Najbolji način učenja programiranja je pisanje programa! Na kraju svakog dana naći ćete test i vežbe. Odgovorite na sva pitanja i pokušajte da ocenite svoj rad, što objektivnije možete. Kasnija poglavlja se zasnivaju na lekcijama iz ranijih poglavlja, tako da je neophodno da razumete materijal, pre nego što nastavite dalje.

Napomena za C programere

Materijal u prvih pet dana će Vam biti poznat. Budite sigurni da ste prešli materijal i uradili vežbanja, da biste bili u punoj formi pre prelaska na Dan 6, "Osnovne klase".

Kuda ste se uputili?

Prva sedmica pokriva materijal koji Vam je potreban, da biste počeli sa programiranjem uopšte, kao i u C + + posebno. U Danu 1, "Počnimo", i Danu 2, "Struktura C+ + programa", bićete uvedeni u osnovne koncepte, programiranja i strukture programa. U Danu 3, "Promenljive i konstante"

učićete o promenljivim i konstantama, kao i kako da koristite podatke u svojim programima. U Danu 4, "Iskazi i izrazi", učićete kako se program grana, zavisno od obezbedenih podataka i od uslova na koje se nailazi dok se program izvršava. U Danu 5, "Funkcije", učićete šta su funkcije i kako se koriste, dok ćete u Danu 6 učiti o klasama i objektima. Dan 7, "Još o upravljanju programom", naučiće Vas više o upravljanju programom i na kraju prve nedelje već ćete pisati prave objektno-orjentisane programe.

Dan 1

Počnimo

Uvod

Dobrodošli u Naučite za 21 dan C + + . Danas ćete krenuti putem na kome ćete postati C+ + programer. Naučićete

- zašto je C + + standard u razvoju softvera
- korake razvoja C+ + programa
- kako da unesete, prevedete i linkujete svoj prvi C+ + program koji radi.

Kratka istorija jezika C+ +

Računarski jezici su doživeli dramatičan razvoj od prvih elektronskih računara, građenih da pomognu u telemetrijskim izračunavanjima tokom Drugog svetskog rata. U početku, programeri su radili sa računarskim instrukcijama niskog nivoa: mašinskim jezikom. Te instrukcije su bile predstavljene dugim nizovima jedinica i nula. Uskoro, asembleri su izmišljeni, da bi zamenili mašinske instrukcije sa mnemonicima, poput ADD i MOV, koji su bili lakši za rad ljudima.

Vremenom, pojavili su se jezici višeg nivoa, poput BASIC-a i COBOL-a. Oni su omogućili ljudima da rade sa nečim sličnim rečima i rečenicama, poput LET 1 = 100. Te instrukcije su prevedene u mašinski jezik, pomoću *interpretatora* i *prevodilaca*. *Interpretator* prevodi program dok ga čita, direktno izvršavajući naredbe programa. *Prevodilac* prevodi programski kod u prelazni oblik. Ovaj korak se naziva prevodenje i proizvodi objektni fajl. Prevodilac tada poziva *linker,* koji od objektnog fajla proizvodi izvršni program.

Pošto interpretatori direktno učitavaju programski kod, kao što je napisan, i izvršavaju ga na lieu mesta, laki su programerima za rad. Prevodioci, s druge strane, uvode dodatne korake prevodenja i povezivanja koda. Oni proizvode programe, koji su veoma brzi svaki put kada se pokrenu. Medutim, složen zadatak prevodenja izvornog koda u mašinski jezik, koji zahteva puno vremena, je već obavljen.

Druga prednost jezika koji se prevode, poput jezika C+ + , je da možete distribuirati izvršni program i ljudima koji nemaju prevodi. Kod jezika koji se interpretiraju, morate da imate interpretator jezika da biste izvršili program.

Tokom mnogih godina, osnovni cilj programera je bio da pišu kratke programe, koji se brzo izvršavaju. Bilo je potrebno da program bude mali, pošto je memorija bila skupa, a morao je da bude brz, posto je i računarska snaga bila skupa. Pošto računari postaju sve manji, jeftiniji i brži i pošto cena memorije stalno pada, prioriteti su se promenili. Danas je cena programerskog vremena znatno veća od cene većine računara koji se koriste u poslovanju. Dobro napisan i lak za održavanje, kod je danas glavni cilj. Kod lak za održavanje znači da, kada posao to zahteva, program može biti proširen i unapreden bez velikih troškova.

Programi

Reč *program* se koristi u dva značenja: da opiše pojedinačne instrukcije, ili izvorni kod, koji je kreirao programer, i da opiše izvršni softver u celini. Ova razlika može da dovede do velike konfuzije, tako da ćemo pokušati da napravimo razliku izmedu izvornog koda, sa jedne, i izvršnog sa druge strane.

Program može biti skup naredbi napisanih od strane programera ili izvršni softver.

Izvorni kod se može transformisati u izvršni program na dva načina: interpretatori direktno prevode izvorni kod u računarske instrukcije i računar ih izvršava neposredno. Alternativno, prevodioci prevode izvorni kod u program, koji možete izvršavati kasnije. Dok je sa interpretatorima lakše raditi, većina ozbiljnog programiranja se obavlja korišćenjem prevodilaca, pošto se prevedeni program izvršava mnogo brže. C + + je jezik koji se prevodi.

Rešavanje problema

Problemi koje danas programeri rešavaju su se promenili. Pre dvadeset godina, programi su pisani da bi se rukovalo velikim količinama neobradenih podataka. Ljudi koji su pisali programe i ljudi koji su ih koristili su bili računarski profesionalci. Danas, računare koristi mnogo vise ljudi i većina njih zna veoma malo o tome kako računari i programi rade. Računari su alati koje koriste ljudi vise zainteresovani da reše probleme u svom poslovanju, nego da se, "preganjaju" sa računarom.

Ironično zvuči: da bi novim korisnicima postali lakši za korišćenje, programi su postali značajno složeniji. Prošli su dani kada su korisnici kucali kriptične komande na egzotične promptove, samo da bi videli niz podataka. Današnji programi koriste sofisticirane "prijateljske korisničke interfejsove" (eng. *user-frendly interface),* koji uključuju vise prozora, menija, dijaloga i mnogo čega drugog, sa čime smo svi postali "familijarni". Programi pisani da podrže ovaj novi prilaz su daleko složeniji od onih koji su pisani samo pre deset godina.

Istovremeno sa promenom programerski zahteva promenili su se i programski jezici i tehnike korišćene za pisanje programa. Iako je cela priča fascinantna, ova knjiga će se usredsrediti na prelazak sa proceduralnog programiranja na objektno orijentisano programiranje.

Procedurally, strukturno i objektno orijentisano programiranje

Sve do nedavno, o programima se mislilo kao o seriji procedura koje se izvršavaju nad podacima. Procedura, ili funkcija, je skup specifičnih instrukcija - izvršavaju se jedna za drugom. Podaci su bili potpuno odvojeni od procedura i trik u programiranju se sastojao u vođenju evidencije o tome koje funkcije pozivaju koje druge funkcije o promenama podataka. Da bi se "izašlo na kraj" sa ovom potencijalno konfuznom situacijom izmišljeno je strukturno programiranje.

Osnovni princip strukturnog programiranja je jednostavan poput maksime "zavadi, pa vladaj". Na računarski program je potrebno gledati kao da se sastoji od skupa poslova. Svaki posao koji je suviše složen, da bi se mogao jednostavno opisati, trebalo bi razbijati u skupove manjih poslova, sve dok ne budu dovoljno mali da se lako mogu razumeti.

Na primer, izračunavanje prosečne plate svakog zaposlenog u kompaniji je složen posao. Možete ga, međutim, razbiti na potposlove:

1. Pronademo koliko svaki zaposleni zaraduje.
2. Prebrojimo koliko imamo zaposlenih.
3. Saberemo sve zarade.
4. Podelimo ukupnu sumu zarada sa brojem zaposlenih.

Potposao sabiranja zarada opet možemo podeliti na sledeće potposlove:

1. Dohvatimo slogove sa podacima o svakom zaposlenom.
2. Pročitamo podatak o zaradi iz sloga.
3. Dodamo zaradu za zaposlenog ukupnoj sumi zarada.
4. Dohvatimo slog sa podacima o sledećem zaposlenom.

Zatim dohvatanje sloga sa podacima o svakom zaposlenom takode može biti podeljeno na:

1. otvaranje datoteke sa podacima o zaposlenima
2. pozicioniranje na ispravan slog
3. čitanje podataka sa diska.

Strukturno programiranje još uvek predstavlja veoma uspešan prilaz rešavanju kompleksnih problema. Ali već krajem prošle decenije postali su jasni nedostaci strukturnog programiranja.

Prvo, prirodno je misliti o podacima (podacima o zaposlenom, na primer) i o tome šta se može uraditi sa tim podacima (sortiranje, menjanje, itd.), kao o povezanim stvarima.

Drugo, programed primećuju da stalno nalaze nova rešenja za stare probleme. Ovo se često naziva "ponovno izmišljanje točka" i potpuno je suprotno od ponovnog korišćenja postojećih rešenja. Ideja koja se javlja posle ponovnog korišćenja je da gradimo komponente koje imaju poznate osobine i zatim ih koristimo u svojim programima kada su nam potrebni. Ovaj model je modeliran prema svetu hardvera - kada je inženjeru potreban novi tranzistor, on ga ne izmišlja ponovo, već ga bira iz kataloga i nalazi jedan koji radi na način koji mu je potreban, ili ga, eventualno, modifikuje. Softverski inženjer nema takvih mogućnosti.

€ S^^^te Način na koji sada koristimo računare - sa menijima, dugmadima i prozorima - utiče na interaktivan, događajima vođen pristup računarskom programiranju. *Voden dogadajima* (eng. *event driven)* znači da kada se dogadaj desi, korisnik pritisne dugme, ili bira iz menija - program mora da odgovori. Programi postaju sve vise interaktivni, što je važno za dizajn te vrste funkcionalnosti.

Programi starog dizajna teraju korisnika da prolazi korak, po korak kroz seriju ekrana. Moderni programi pokretani dogadajima prikazuju sve opcije odjednom i odgovaraju na akcije korisnika.

Objektno orijentisano programiranje pokušava da odgovori na ove potrebe, pružajući tehnike za savladavanje složenih problema, omogućavajući ponovno korišćenje softverskih komponenti i objedinjavajući podatke sa poslovima koji manipulišu tim podacima.

Suština objektno orijentisanog programiranja je da tretira podatke i procedure, koje manipulišu tim podacima, kao jedan "objekt" - samostalan entitet sa identitetom i odredenim sopstvenim karakteristikama.

C++ i objektno orijentisano programiranje

C++ potpuno podržava objektno orijentisano programiranje, uključujući četiri osnove objektno orijentisanog razvoja: enkapsulacija, skrivanje podataka, nasledivanje i polimorfizam.

Enkapsulacija i skrivanje podataka

Kada je inženjeru potrebno da doda novi otpornik uredaju koji projektuje, on ga, obično, ne pravi ispočetka, već prošeta do kutije sa otpornicima, pogleda obojene trake za označavanje osobina otpornika i izabere onaj koji mu je potreban. Otpornik je "crna kutija", što se inženjera tiče - njemu nije bitno da zna kako otpornik radi, sve dok odgovara specifikacijama; nije potrebno da gleda unutar kutije, da bi je koristio u svom projektu.

Osobina da je jedinica samostalna se naziva *enkapsulacija* (eng. *encapsulation),* kojom možemo postići *skrivanje podataka* (eng. *data hiding),* Skrivanje podataka je visoko cenjena osobina - objekat se može koristiti, a da korisnik ne mora znati ili mariti kako on unutra funkcioniše. Isto tako kao što možete koristiti frižider bez znanja o tome kako kompresor radi, možete koristiti dobro dizajniran objekat bez znanja o njegovim unutrašnjim podacima.

Slično tome, inženjeru nije potrebno da zna unutrašnje stanje otpornika. Sve osobine otpornika su enkapsulairane u njegovom objektu. Nije neophodno znati kako otpornik radi, da bi se efikasno koristio. Ti podaci su skriveni u kućištu otpornika.

Nasledivanje i ponovno korišćenje

Kada inženjeri u Acme Motorima žele da izgrade nova kola, oni imaju dva izbora: mogu da počnu ispočetka, ili mogu da modifikuju postojeći model. Možda je njihov model "Zvezda" skoro savršen, ali možda žele da dodaju turbopunjač i menjač sa šest brzina. Glavni inženjer tada ne bi želeo da počne iz početka, već bi radije rekao: "Hajde da napravimo još jednu 'Zvezdu', ali dodajmo nove mogućnosti. Kola ćemo nazvati 'Kvazar'." "Kvazar" je vrsta "Zvezde", ali sa dodatnim osobinama.

C++ podržava ideju o ponovnom korišćenju preko nasledivanja. Novi tip, koji je proširenje postojećeg, može biti deklarisan. Ova nova potklasa se *izvodi* (eng. *derive)* iz postojećeg tipa i ponekad se naziva izveden tip (eng. derived type). "Kvazar" je izveden iz "Zvezde" i, time nasleduje sve njene kvalitete, ali može dodati novi koje su potrebni. Nasledivanje i njegova primena u programskom jeziku C++ se razmatra u Danu 12, "Nasledivanje" i Danu 15, "Napredno nasledivanje".

Polimorfizam

Novi "Kvazar" može reagovati drugačije od "Zvezde" kada pritisnete nogom pedalu za ubrzavanje. "Kvazar" može da koristi puno ubrizgavanje i turbopunjač, dok "Zvezda", jednostavno, pušta benzin u karburator. Korisnik, naravno, ne mora da zna ništa o tim razlikama. On će samo pritisnuti pedalu i prava stvar će se desiti, zavisno od toga koja kola vozi.

C++ podržava ideju da različiti objekti "rade prave stvari", preko polimorfizma funkcija i polimorfizma klasa. Poly znači "mnogo", dok *morph* znači "forma". Polimorfizam znači da se isto ime može javiti u različitim oblicima i razmatra se Danu 10, "Napredne funkcije" i Danu 13, "Polimorfizam".

Kako se C+ + razvijao

Kada su objektno orijentisana analiza, projektovanje i programiranje počeli da se razvijaju, Bjarne Stroustrup je uzeo najpopularniji programski jezik za razvoj komercijalnog softvera, C, i proširio ga osobinama potrebnim za objektno orijentisano programiranje. Kreirani je C + + je, za manje od decenije od, jezika koji su koristili malobrojni programed u AT&T, postao programski jezik koji stalno koristi približno milion programera širom sveta. Očekuje se da će do kraja ove decenije C + + postati dominantan programski jezik u komercijalnom razvoju softvera.

Tačno je da je C + + nadskup jezika C i da je svaki ispravan C program istovremeno i ispravan C+ + program, ali razlike izmedu programskih jezika C i C + + su veoma značajne. Jeziku C + + je koristio odnos sa C-om tokom mnogih godina, kao što su i C programeri mogli da olakšaju svoj prelazak na C + + . Ipak, da bi mogli da se koriste svim prednostima programskog jezika C+ + , mnogi programeri moraju da se oduče od mnogo Čega što su znali i da nauče potpuno novi put konceptualizacije i rešavanja programskih problema.

ANSI standard

Akreditovani komitet za standard, koji radi pod procedurama Američkog instituta za standarde (eng. American National Standards Institute - ANSI), priprema meduna rodni standard za C + + , čiji nacrt je publikovan i dostupan na Internetu, na adresi www.1ibertyassociates.com.

ANSI standard je pokušaj da se osigura da je C+ + portabilan - da će se programski kod, koji pišete za Microsoftov prevodilac, prevesti bez grešaka i kada koristite prevodilac nekog drugog proizvodača. Pošto programski kod u ovoj knjizi odgovara ANSI standardu, trebalo bi da se prevede bez greške na Macintosh, Windows, ili Alpha računarima.

Za većinu ljudi koji uče C + + ANSI standard će biti neprimetan. Standard je stabilan već neko vreme i svi glavni proizvođači ga podržavaju. Učinili smo maksimalne napore da budemo sigurni da sav programski kod u ovom izdanju knjige odgovara ANSI standardu.

Da li je potrebno da prvo naučim C?

Sledeće pitanje se neminovno postavlja: "Pošto je C+ + nadskup jezika C, da li je potrebno da prvo naučim C?". Stroustrup i većina drugih C + + programera se slažu da nije. Ne samo da je nepotrebno, već Vam nepoznavanje jezika C može biti i prednost. Ova knjiga pokušava da odgovori potrebama ljudi poput vas, koji prilaze programskom jeziku C+ + bez prethodnog iskustva sa C - om. U stvari, u ovoj knjizi se pretpostavlja da nemate nikakvo iskustvo u programiranju.

Pre programiranja

C + + , možda vise nego drugi jezici, zahteva da programer projektuje program pre nego što ga napiše. Trivijalni problemi, poput onih koji se razmatraju u prvih nekoliko poglavlja ove knjige, ne zahtevaju mnogo projektovanja. Složeni problemi, sa kojima se profesionalni programeri suočavaju svaki dan, zahtevaju projektovanje i što je program pažljivije projektovan, to je verovatnije da će rešiti problem, i to na vreme i u okvirima predviđenog budžeta. Dobar projekat takode omogućava da program nema mnogo grešaka i da je lak za održavanje. Izračunato je da 90 odsto cene softvera predstavljaju troškovi debagovanja i održavanja. Dobar projekat može da smanji te troškove i ima značajan uticaj na finalnu cenu.

Prvo pitanje koje treba da postavite kada se pripremate da projektujete neki program je: "Koji problem ja pokušavam da rešim?". Svaki program treba da ima jasan, dobro usaglašen cilj. Ustanovićete da i najjednostavniji program u ovoj knjizi to ima.

Drugo pitanje koje postavlja svaki dobar programer je: "Da li se problem može rešiti bez pisanja posebnog softvera?". Ponovno korišćenje starog programa, korišćenje pera i papira, ili kupovina gotovog softvera je često bolje rešenje problema, nego pisanje novog softvera. Programer koji može da ponudi ove alternative nikada neće imati manjak posla.

Uz pretpostavku da razumete problem i da on zahteva pisanje novog programa, spremni ste da počnete sa projektovanjem svog programa.

Vaša razvojna okolina

U ovoj knjizi je pretpostavljeno da Vaš računar ima mod u kome se može pisati direktno na ekran, bez potrebe da se brine o grafičkom okruženju, kao u slučaju Windowsa, ili Macintosha.

Prevodilac treba da ima sopstveni ugradeni tekst editor, ili možete koristiti komercijalni tekst editor ili program za obradu teksta koji može da snimi tekst datoteke. Najvažnije je, da, bez obzira u čemu pišete svoj program, morate ga snimiti u običnoj tekst datoteci, bez ikakvih naredbi za obradu teksta, sačuvanih u tekstu. Primeri editora koje možete koristiti su Windows Notepad, DOS Edit, Brief, Epsilon, EMACS i vi. Mnogi komercijalni programi za obradu teksta, poput WordPerfecta, Worda i drugih, takođe nude mogućnost snimanja običnih tekst datoteka.

Datoteke kreirane u editoru nazivaju se datoteke sa izvornim kodom i u C+ + se .obično, imenuju sa ekstenzijom .CPP, .CP, ili X. U ovoj knjizi koristićemo ekstenziju .CPP za datoteke sa izvornim kodom, ali proverite koju ekstenziju podrazumeva Vaš prevodilac.

^MAPOMCHA^ Većini C+ + prevodioca nije bitno koju ekstenziju koristite za svoje izvorne kodove, ali, ako drugačije ne odredite, većina će koristiti CPP kao podrazumevanu vrednost.

Koristite jednostavni editor teksta, da biste kreirali svoj izvorni kod, ili koristite ugradeni editor koji dolazi uz Vaš C+ + prevodilac

Nemojte koristiti programe za obradu teksta koji snimaju u datoteku specijaine karaktere za formatiranje teksta. Ako koristite program za obradu teksta, sačuvajte datoteku kao ASCII tekst.

Snimite svoje datoteke sa izvornim kodom sa .C, Д ili .CPP ekstenzijama.

Proverite u dokumentaciji specifičnosti C+ + prevodioca i povezivača (Linker), da biste bili sigurni da znate kako da prevedete i povežete svoj program.

Prevođenje izvornog koda

Bez obzira što C + + izvorni kod može delovati kriptično i što će svako ko ne poznaje C + + imati problema da razume za šta tačno služi, on je još uvek u obliku koji ljudi mogu da čitaju. Vaša datoteka sa izvornim kodom nije program i ne može da se izvršava, ili pokreće kao što može program.

Da biste od svog izvornog koda dobili izvršni program, morate koristiti prevodilac. Kako ćete pokrenuti svoj prevodilac i kako ćete mu saopštiti gde da pronade svoj izvorni kod, menja se od prevodioca do prevodioca; proverite u svojoj dokumentaciji. Kod Borlandovog Turbo C + + prevodioca izabraćete RUN komandu iz menija, ili ćete otkucati

`tc <naziv-datoteke>`

u komandnoj liniji, gde je *<naziv-datoteke>* naziv datoteke sa Vašim izvornim kodom (na primer, test.cpp). Kod drugih prevodilaca to može biti drugačije.

>^ИАРОШЦ^ Ako prevodite izvorni kod sa komandne linije, treba da otkucate sledeće:

Za Borland C+ + prevodilac:	**bcc <naziv-datoteke>**
Za Borland C+ + prevodilac za Windows:	**bcc <naziv-datoteke>**
Za Borland Turbo C+ + prevodilac:	**tc <naziv-datoteke>**
Za Microsoft prevodioce.-	**cl < naziv-datoteke >**

Pošto se izvorni kod prevede, dobija se objektna datoteka. Ona, najčešće, ima ekstenziju .OBJ. Ovo ipak još uvek nije izvršni program. Da biste ga dobili, morate pokrenuti povezivač.

Kreiranje izvršnog programa sa povezivačem

C+ + programi se, obično, kreiraju povezivanjem jednog, ili vise OBJ datoteka sa jednom, ili vise biblioteka. Biblioteka je skup datoteka koje se mogu povezivati. Dobija se sa prevodiocem, može se kupiti posebno, ili napraviti i prevesti. Svi C+ + prevodioci dolaze sa bibliotekom korisnih funkcija (ili procedura) i klasa, koje

možete da uključite u svoje programe. Funkcija je blok koda za rešavanje nekog problema; na primer, sabira dva broja, Ш štampa na ekran. Klasa je kolekcija podataka i funkcija koje se odnose na njih; pričaćemo dosta o klasama, počevši od Dana **5,** "Funkcije".

Koraci za kreiranje izvršnog programa su:

1. Kreirajte datoteku sa izvršnim kodom, sa **.CPP** ekstenzijom.
2. Prevedite izvršni kod u datoteku sa **.OBJ** ekstenzijom.
3. Povežite OBK datoteku sa potrebnim bibliotekama da biste dobili izvršni program.

Razvojni ciklus

Ako bi svaki program radio prvi put kada ga probate, to bi bio potpuni razvojni ciklus: napišete program, prevedete izvorni kod, povežete program i izvršite ga. Na žalost, skoro svaki program, bez obzira kako trivijalan bio, i imaće greške, ili bagove. Neki bagovi dovode do prestanka prevodenja programa, neki dovode do prekida povezivanja, a neki se pokazuju samo tokom rada programa.

Bez obzira koji tip baga nadete, morate ga popraviti i u to su uključene promene izvornog koda, ponovno prevodenje, povezivanje i pokretanje programa. Ciklus je predstavljen na slici 1.1, gde su prikazani koraci u razvojnom ciklusu.

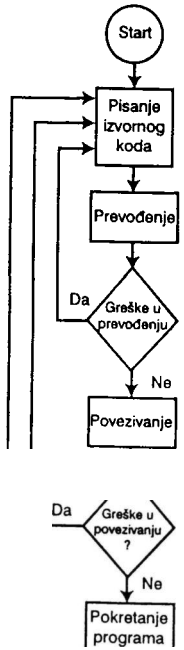
ZDRAVO.CPP! - Vaš prvi C + + program

Tradicionalne knjige o programiranju počinju pisanjem reči "Zdravo svete!" na ekran, ili varijacijama te izjave. Ta tradicija je nastavljena i ovde.

Ukucajte svoj prvi program direktno u editor, tačno kao što je prikazano. Kada budete sigurni da ste ga tačno uneli, snimite program, prevedite ga, povežite i pokrenite. On će odštampati reči "Zdravo svete!" na Vašem ekranu. Ne brinite puno o tome kako program radi - ovo je samo način da upoznate razvojni ciklus. Svaki deo ovog programa objasnićemo tokom nekoliko sledećih dana.

viiPOZORENjry **Listing koji sledi sadrži brojeve linija na levoj strani. Ti brojevi služe za snalaženje u knjizi. Oni ne treba da se unose u editor. Na primer, linuju 1 listinga 1.1, treba da unesete kao-.**

```
#include <iostream.h>
```



Slika 1.1.
Koraci u razvoju C+ + programa

Listing 1.1. ZPRAVO.CPP! program Zdravo svete

```
1: #include <iostream.h>
2:
3: int main()
4: {
5:    cout « "Zdravo svete!\n";
6:       return 0;
7: }
```

Proverite da li ste sve uneli kao u listingu. Obratite pažnju na znake interpunkcije. Znak < < u petoj liniji je simbol redirekcije, koji se na većini tastatura dobija tako što se drži taster Shift i pritisne taster zarez dva puta. Linija 5 se završava znakom tačka-zarez; nemojte ga izostaviti!

Takode budite sigurni da pravilno sledite uputstva svog prevodioca. Većina prevodilaca će proces povezivanja obaviti automatski, ali proverite u dokumentaciji. Ukoliko dobijete izveštaj o greškama, pogledjte pažljivo svoj kod i utvrdite u čemu se razlikuje od koda iz gornjeg listinga. Ukoliko dobijete grešku u liniji **1,** poput cannot f i nd file i ostream.h, proverite dokumentaciju za svoj prevodilac, kako da postavite include putanju, ili promenljive okoline. Ako dobijete grešku da nema prototipa za main, dodajte liniju int main(); tačno pre linije **3.** Biće potrebno da dodate ovu liniju pre početka svake main funkcije u svakom programu u ovoj knjizi. Većina prevodilaca ovo ne zahteva, ali neki to traže.

Vaš program na kraju treba da izgleda ovako:

```
linclude  <iostream.h>


int  main();
{
cout «"Zdravo svete!\n";
   return 0;
```

Pokušajte da pokrenete ZDRAVO. EXE. Trebalo bi da bude ispisano

```
Zdravo svete!
```

direktno na ekran. Ukoliko je tako, čestitamo! Upravo ste uneli, preveli i pokrenuli svoj prvi C + + program. Možda ne izgleda nešto posebno, ali skoro svaki profesionalni C+ + programer je počeo sa tačno ovim programom.

Greške prevodenja

Greške u toku prevodenja se mogu pojaviti zbog velikog broja razloga. Obično su rezultat grešaka u kucanju, ili drugih slučajnih minornih grešaka. Dobri prevodioci će Vam ne samo reći šta ste pogrešili, već će Vam tačno pokazati mesto u kodu gde ste napravili grešku. Odlični prevodioci će Vam čak sugerisati ispravku.

Ovo možete videti tako što ćete namerno postaviti grešku u svoj program. Ukoliko se ZDRAV0.EXE! izvršava bez greške, promenite kod, tako što ćete ukloniti veliku zagradu u liniji 6. Vaš program će sad izgledati poput onog u listingu 1.2.

Listing 1.2.: Demonstracija greške tokom prevodenja

```
1: linclude  <iostream.h>
2:
3: int  main()
4: {
5:    cout « "Zdravo svete!\n";
6:       return 0;
```

Prevedite ponovo svoj program i videćete izveštaj o grešci:

mm*mm^» `Zdravo.cpp, line 5: Compound statement missing terminating } in function main().`

Ovaj izveštaj o grešci Vam saopštava datoteku i broj linije u kojoj je problem i kakav je problem u pitanju (mada priznajem da je, donekle, kriptičan). Izveštaj o grešci Vas upućuje na liniju **5.** Prevodilac nije siguran da li ste nameravali da stavite veliku zagradu, pre, ili posle cout naredbe u liniji **5.** Ponekad Vam greške samo genera-lno opisuju problem. Kada bi prevodilac mogao savršeno da identifikuje svaki pro-blem, mogao bi i sam da popravi kod.

Rezime

Posle čitanja ovog poglavlja, trebalo bi da znate kako se C + + razvio i za rešavanje kojih problema je namenjen. Trebalo bi da ste sigurni da je učenje programskog jezika C+ + pravi izbor za svakoga ko je zainteresovan za programiranje u sledećoj deceniji. C+ + daje alate objektno orijentisanog programiranja i performanse jezika sistemskog nivoa, što sve čini ovaj jezik glavnim izborom programera.

Danas ste naučili kako da unesete, prevedete, povežete i pokrenete svoj prvi C + + program i šta je to normalni razvojni ciklus. Takode ste delimično naučili čemu uopšte služi objektno orijentisano programiranje. Vratićete se ovim temama tokom sledeće tri nedelje.

Pitanja i odgovori

P Koja je razlika izmedu editora teksta i programa za obradu teksta?

O Editor teksta proizvodi datoteke sa običnim tekstom u sebi. Nema komandi za formatiranje teksta, ili drugih specijalnih znakova, koji **se** zahtevaju od pojedinih programa za obradu teksta. Tekstualne datoteke nemaju automatsko prelamanje reči, štampu u italiku i drugo.

P Ako moj prevodilac ima ugradeni editor, da li moram da ga koristim?

O Skoro svi prevodioci će prevesti kod unet u ma kom editoru teksta. Prednost korišćenja ugradenog editora može biti brže kretanje između koraka unosa koda i prevodenja u razvojnom ciklusu. Napredniji prevodioci imaju potpuno integrisanu razvojnu okolinu, omogućujući programerima da pristupe datotekama sa pomoću, edituju i prevode program, sve na istom mestu i da reše greške prevodenja i povezivanja i sve to bez napuštanja te okoline.

P **Da** li **mogu da zanemarim** upozoravajuće **poruke** prevodioca?

0 U mnogim knjigama je nedoumica, ali ja ću zauzeti stav: Ne! Naviknite sebe, od prvog dana, da tretirate upozoravajuće poruke kao greške. C+ + koristi prevodilac, da Vas upozori da radite nešto što možda ne želite. Pratite ova upozorenja i učinite šta je potrebno da nestanu.

P Šta je to vreme prevodenja?

O *Vreme prevodenja* (compile **time)** je vreme kada pokrećete svoj prevodilac, za razliku od vremena povezivanja (kada pokrećete povezivač - link time), ili vremena izvršavanja (kada izvršavate program - run-time). Ovo su samo programerske skraćenice za identifikovanje tri vremena tokom kojih se greške, obično, pokažu.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeduju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Test

1. Koja je razlika izmedu interpretatora i prevodioca?
2. Kako prevodite izvorni kod sa svojim prevodiocem?
3. Sta radi povezivač?
4. Koji su koraci u normalnom razvojnom ciklusu?

Vežbe

1. Pogledajte sledeći program i pokušajte da pogodite šta on radi, ne pokrećući ga.

```
1: linclude <iostream.h>
2: int main()
3: {
4: int x = 5;
5: int y = 7;
6: cout "\n";
7:  cout « x + y « " " « x * y;
8: cout "\n";
9: return 0;
10: }
```

2. Unesite program iz vežbe 1, zatim ga prevedite i povežite. Sta on radi? Da li radi ono što ste pretpostavili?
3. Unesite sledeći program i prevedite ga. Kakve greške dobijate?

```
1: include <iostream.h>
2: int main()
3: {
```

```
4:  cout « "Zdravo svete\n";
5:  return 0;
6: }
```

4. Ispravite grešku u programu iz vežbe **3,** prevedite ga, povežite i pokrenite. Šta on radi?

Dan 2

Delovi C++ programa

C++ program se sastoji od objekata, funkcija, promenljivih i drugih delova. Veći deo ove knjige je posvećen objašnjavanju ovih delova u dubinu, ali da biste stekli osećaj kako program čini celinu morate videti ceo ispravan program. Danas ćete naučiti

- delove C++ programa
- kako delovi rade zajedno
- šta je to funkcija i šta ona radi.

Jednostavni program

Сак i jednostavan program ZDRAVO.CPP, iz Dana 1, "Počnimo", ima dosta zanimljivih delova. U ovom poglavlju ćemo analizirati ovaj program detaljnije. Listing 2.1 ponavlja originalnu verziju ZDRAVO.CPP.

Listing 2.1.: ZDRAVO.CPP, demonstira delove C++ programa

```
linclude <iostream.h>

 int  main()
{
  cout   «  "Zdravo  svete!\n";
      return  0;
```

```
J Ш Ш З ^ Zdravo  svete!
```

ШШШЬ U liniji **l,** datoteka iostream.h je uključena u program. Prvi karakter u liniji je # znak, što predstavlja signal pretprocesoru. Svaki put kada pokrenete svoj prevodilac, pretprocesor se izvršava. Pretprocesor analizira izvorni kod, tražeći linije koje počinju sa # i deluje na te linije pre nego što se prevodilac pokrene.

i ncl ude je instrukcija pretprocesoru koja kaže, "Ono što sledi je naziv datoteke. Nadi tu datoteku i postavi je na ovo mesto." Znači manje i veće oko naziva datoteke ukazuju pretprocesoru da potraži tu datoteku na svim uobičajenim mestima. Ukoliko je vaš prevodilac pravilno konfigurisan, znaci < i > će ukazati pretprocesoru da potraži datoteku iostream.h **u** direktorijumu koji sadrži sve **H** datoteke za vaš prevodilac. Datoteka iostream.h (Input-Output-Stream) se koristi od strane naredbe cout, koja pomaže pri ispisivanju na ekran. Rezultat linije 1 je da uključi datoteku iostream.h u program, kao da ste je sami ukucali.

@ЖШШЬ *Pretprocesor* (eng. *preprocessor)* se izvršava pre prevodioca, svaki put kada se prevodilac pozove. Pretprocesor prevodi svaku liniju koja pocinje znakom # u specijalnu komandu, pripremajući Vašu izvornu datoteku za prevodilac.

Linija 3 označava stvarni početak programa fukcijom main(). Svaki C++ program ima main() funkciju. Generalno, funkcija je blok koda, koja obavlja jednu, ili vise akcija. Obično funkcije pozivaju druge funkcije, ali funkcija `main()` je posebna. Kada se Vaš program pokrene, `main()` se poziva automatski.

Poput ostalih funkcija, `main(),` mora da iskaže koju vrstu vrednosti će vratiti. Tip povratne vrednosti za main() u `ZDRAVO.CPP` je void, što znači da ova funkcija neće vratiti nikakvu vrednost. Vraćanje vrednosti iz funkcija je detaljno obrađeno u Danu 4, "Iskazi i izrazi".

Sve funkcije počinju otvorenom velikom zagradom ({) i završavaju se zatvorenom velikom zagradom (}). Zagrade za `main()` funkciju su u linijama 4 i 7. Sve između otvorene i zatvorene velike zagrade se smatra kao deo funkcije.

Glavni deo programa je u liniji 5. Objekat cout se koristi za štampanje poruke na ekran. Objekte ćemo generalno "pokriti" u Danu 6, "Osnovne klase", a cout i srodni objekat cin detaljno u Danu 17, "Pretprocesor". Ova dva objekta, cout i cin, se koriste u C++ za štampanje stringova i vrednosti na ekran i preuzimanje sa tastature. *String* je samo niz karaktera.

Evo kako se cout koristi: otkucajte reč cout, iza koje treba da se nade operator izlazne redirekcije («). Sve što se nalazi iza operatora izlazne redirekcije biće odštampano na ekranu. Ukoliko želite da odštampate niz karaktera, postavite ga između dvostrukih navodnika ("), kao što je prikazano u liniji 5.

С р Ш И Ш *Tekstualni string* (eng. *text string)* je serija karaktera koji se mogu odstampati.

Zadnja dva karaktera, **`\n`,** upućuju cout da pređe u novi red posle reči Zdravo svete! Ovaj specijalni kod je detaljno objašnjen u Danu 17, kada se razmatra cout.

Svi programi koji se pridržavaju ANSI standarda deklarišu mainQ da vraća celobrojnu vrednost (int). Ova vrednost se "vraća" operativnom sistemu kada program završi sa radom. Neki programeri ukazuju na postojanje greške, vraćajući 1. U ovoj knjizi **`ma i`** n () uvek vraća 0.

Funkcija `main()` se završava u liniji 7 zatvorenom velikom zagradom.

Kratak pogled na cout

U Danu 16, 'Tokovi", videćete kako da koristite cout za štampanje podataka na ekran. Za sada, možete koristiti cout i iako ne razumete u potpunosti kako radi. Da biste odštampali nešto na ekranu, koristite reč cout, praćenu operatorom izlazne redirekcije («), do koga dolazite tako §to dva puta, zaredom, otkucate znak manje (<). Iako je reč o dva karaktera, C++ ih tretira kao jedan.

Iza operatora izlazne redirekcije navedite svoje podatke. Listing 2.2 ilustruje kako se to koristi. Unesite primer tačno kao što je napisan, samo zamenite svoje ime tamo gde vidite Jesse Liberty (ako je Vaše ime Jesse Liberty, ostavite sve tačno kako jeste; to je sjajno - ali ja još uvek ne delim svoje prihode!).

Listing 2.2.: Korišćenje cout

```
// Listing 2,2 korišćenjem cout

#include <iostream.h>
int main()
{
   cout « "Zdravo.\n";
   cout « "Ovo je 5: • « 5 « "\n";
   cout « "Manipulator endl nas prevodi u novi red." « end!;
   cout « "Evo veoma velikog broja: \t" « 70000 « end!;
   cout « "Ovo je zbir 8 i 5: \t\t" « 8+5 « endl;
   cout « "Evo razlomka: \t\t\t" « (float) 5/8 « endl;
   cout « "I veoma velikog broja: \t\t" « (double) 7000*7000 « endl;
   cout « "Nemojte zaboraviti da zamenite Jesse Liberty svojim imenom... \n";
   cout « " Jesse Liberty je C++ programer! \n";
    return 0;
```

```
Zdravo.
Ovo je 5: 5
Manipulator endl nas prevodi u novi red.
Evo veoma velikog broja:            70000
Ovo je zbir 8 i 5:                  13
Evo razlomka:                       0.625
I veoma velikog broja:              4.9e+07
Nemojte zaboraviti da zamenite Jesse Liberty svojim imenom...
Jesse Liberty je C++ programer!
```

U liniji 3 naredba linclude <iostream.h> dovodi do uključivanja datoteke iostream.h **u** izvorni kod. Ovo je potrebno ako koristite cout i srodne funkcije.

U liniji 6 je jednostavan primer upotrebe cout, za štampu niza karaktera. Znak \n je specijalni znak za formatiranje. On ukazuje cout da štampa znak za novi red na ekran.

Tri vrednosti su prosledene za cout **u** liniji 7 i svaka vrednost je odvojena operatorom izlazne redirekcije («). Prva vrednost je string: "0vo je 5: ". Obratite pažnju na prazno mesto posle dve tačke. Prazno mesto je deo stringa. Sledeća vrednost je 5, koja se prosleduje operatoru izlazne redirekcije, kao i znak za *novi red* (eng. new line character) \n (uvek pod dvostrukim, ili jednostrukim navodnicima). Sve to dovodi do štampanja linije na ekran

```
Ovo je 5: 5
```

Pošto nema znaka za novi red posle prvog stringa, sledeća vrednost se odmah zatim štampa. Ovo se naziva *konkatenacija* (eng. *concatenating)* dve vrednosti.

U liniji 8 štampa se poruka sa informacijom na ekran i zatim se posle nje koristi manipulator endl. Namena endl je štampanje znaka za prelazak u novi red na ekran (drugi načini upotrebe manipulatora endl su objašnjeni u Danu 16).

U liniji 9 uvodimo \t, novi znak za formatiranje. On umeće tab karakter i koristi se u linijama 8 do 12, da bi se poravnao prikaz na ekranu. Linija 9 pokazuje da mogu biti štampane ne samo celobrojne vrednosti, već i *dugačke celobrojne vrednosti* (eng. long integer). Linija 10 je primer jednostavnog sabiranja unutar cout. Vrednost od 8+5 je prosledena cout, ali je 13 odštampano.

U liniji 11 za cout je prosledeno 5/8. Termin (float) je pokazao cout da želite da se ovo izračuna kao decimalni broj, pa je zato odštampan razlomak. U liniji 12 za cout je prosledeno 7000*7000 i termin (double) se koristi da ukaže da želite prikaz rezultata u eksponencijalnom zapisu. Sve ovo je detaljno objašnjeno u Danu 3, "Promenljive i konstante", pri objašnjavanju tipova podataka.

U liniji 14 zamenili ste svoje ime i izlaz je potvrdio da ste zaista C++ programer. To mora da je tačno, kada računar kaže!

Komentari

Kada pišete program, sve je uvek jasno i, samo po sebi, razumljivo je šta želite da uradite. Zanimljivo je, medutim, da kada se mesec dana kasnije vratite programu, isti kod može biti krajnje zbunjujući i nejasan. Nisam siguran kako se to uvlači u program, ali uvek tako biva.

Tipovi komentara

C++ komentari se mogu pojavljivati u dva oblika: kao dupla kosa crta (//) i kao kosa crta - zvezdica (/*). Komentar u obliku duple kose crte, koji ćemo pominjati kao komentar u C++ stilu, ukazuje prevodiocu da ignoriše sve što sledi iza komentara, sve do kraja linije.

Komentar kosa crta - zvezdica ukazuje prevodiocu da ignoriše sve što ga sledi, sve dok ne naide na oznaku kraja komentara zvezdica - kosa crta (*/). Ovaj komentar ćemo pominjati kao komentar u C stilu. Svaki /* mora imati zatvarajući */.

Kako ste i sami možda pretpostavili, komentari u C stilu se koriste i u programskom jeziku C, dok komentari u C++ stilu nisu deo zvanične definicije C-a.

Mnogi C++ programeri koriste stalno komentare u C++ stilu u svojim programima, dok komentare u C stilu koriste samo za izbacivanje velikih blokova koda u programu. Možete uključiti komentare u C++ stilu unutar bloka koda, stavljenog u komentar u C stilu; sve, uključujući i komentare u C++ stilu, ignoriše se unutar oznaka komentara u C stilu.

Korišćenje komentara

Kao generalno pravilo, ceo program treba da ima komentare na početku, koji Vam govore šta program radi. Svaka funkcija, takode, treba da ima komentar za objašnjenje šta funkcija radi i koje vrednosti vraća. Na kraju, svaki deo Vašeg programa koji je složen treba komentarisati.

Listing 2.3 demonstrira korišćenje komentara, pokazujući da oni ne utiču na izvršavanje programa, niti na njegov izlaz.

Listing 2.3.: P0M0C.CPP demonstira upotrebu komentara

```
1: #include <iostream.h>
2:
3: int main()
4: {
5:   /* Ovo je komentar
6:   i nastavlja se sve do zatvarajuće
7:   oznake komentara zvezdica-kosa crta */
8:     cout « "Zdravo svete! \n";
9:     // Ovaj komentar se završava krajem reda
10:    cout « "Ovaj komentar je završen! \n";
11:
12:  // Dupla kosa crta, kao komentar, može biti sama u liniji
13: /* kao i kosa crta-zvezdica */
14:     return 0;
15:}
```

```
Zdravo svete!
Ovaj komentar je završen!
```

IggTOhj* Komentare od linije 5 do 7 potpuno ignoriše prevodilac, kao i komentare u linijama 9,12 i 13. Komentar u liniji 9 se završava krajem reda, dok komentari u linijama 5 i 13 zahtevaju znak za kraj komentara.

Komentari na početku svake datoteke

Dobra je ideja pisati blok komentara na početku svake datoteke sa kodom, koju napišete. Stvarni stil tog bloka komentara je stvar ličnog ukusa, ali svako takvo zaglavlje treba da uključi bar sledeće informacije:

- naziv funkcije ili programa
- naziv datoteke
- šta ta funkcija ili program rade
- opis kako program radi
- ime autora
- istorijat revizija (beleška o svakoj promeni)
- koji prevodioci, povezivači i drugi alati su korišćeni pri izradi progama
- dodatne beleške po potrebi.

Na primer, sledeći blok komentara se može pojaviti na početku programa Zdravo svete!

```
y*****************************************************************
Program:              Zdravo svete!
Datoteka:             ZDRAVO.CPP!
Funkcija:             Main (kompletan program se nalazi u ovoj datoteci)
Opis:                 štampa reči "Zdravo svete" na ekranu
Autor:                Jess Liberty (jl)
Razvojno okruženje:   Turbo C++ verzija 4, 486/66 32 MB RAM,
                      Windows 3.1 DOS 6.0. EasyWin modul.
Beleške:              Uvodni, jednostavan program.
Revizije:             1.00 1.10.1994. (jl) Prva verzija
                      1.01 2.10.1994. (jl) Malo slovo za "svet"
*****************************************************************y
```

Veoma je važno obezbediti ažurnost beleški i opise ažurnim. Standardni problem sa zaglavljima je što se zanemare posle početnog kreiranja i tokom vremena postanu vise smetnja, nego pomoć. Ali, ako se propisno ažuriraju, ona mogu biti od neprocenjive pomoći u razumevanju celog programa.

Listinzi u ostalom delu knjige nemaju zaglavlja zbog štednje prostora. To ne smanjuje njihov značaj, tako da će se ipak pojavljivati u programima na kraju svake nedelje.

Poslednje upozorenje o komentarima

Komentari koji objašnjavaju očigledno su beskorisni. U stvari, mogu biti i kontraproduktivni, pošto se kod može promeniti, a programer može zaboraviti da ažurira komentar. Osim toga, ono što je očigledno jednom čoveku, može biti nejasno drugom, tako da je razmišljanje neophodno prilikom unošenja komentara.

Komentari ne treba da govore šta se dešava, već zašto se nešto dešava.

un* **Koristite komentare u svojim programima.

Cuvajte komentare ažurnim.

Koristite komentare da biste objasnili šta koji deo koda radi.

Nemojte koristiti komentare za kod koji je razumljiv sam po sebi.

Funkcije

Funkcija main() nije tipična. Tipične funkcije se pozivaju tokom izvršavanja programa. Program se izvršava liniju, po liniju, po redu kojim se pojavljuju u Vašem izvornom kodu, sve dok se ne stigne do funkcije. Tada se bezuslovno skače unutar programa na kod funkcije da bi se ona izvršila. Posle završetka funkcije, ona vraća kontrolu na liniju koda, neposredno posle poziva funkcije.

Dobra analogija za ovo je oštrenje olovke. Ako crtate sliku i Vaša olovka se polomi, prestaćete da crtate, otići ćete da zaoštrite olovku, a zatim ćete nastaviti tamo gde ste stali. Kada je programu potrebna odredena usluga, on poziva funkciju da obavi tu uslugu, a zatim pokupi ono što ostane kada funkcija završi sa izvršavanjem. Ova ideja je prikazana u listingu 2.4.

Listing 2.4.: Demonstiranje pozivanja funkcije

```
#include <iostream.h>

// Funkcija: Demonstracija funkcije
// štampa korisnu poruku
void DemonstrationFunctionO
{
    cout « "Mi smo unutar Demonstracione funkcije\n";
}

// Funkcija main - štampa poruku, zatim
// poziva DemonstrationFunction, a, zatim, štampa
// drugu poruku.
int main()
{
    cout « "Unutar main\n";
    DemonstrationFunctionO;
    cout << "Nazad u main\n";
```

nastavljase

Listing 2.4.: Demonstiranje pozivanja funkcije *nastavak*

```
18:       return 0;
19: }
```

```
Unutar main
Mi smo unutar Demonstracione funkcije
Nazad u main
```

U liniji 13 zapravo počinje program. U liniji 15 mai n () štampa na ekranu poruku, koja saopštava da smo u main(). Posle štampanja poruke, linija 16 poziva funkciju DemonstrationFunctionO. U ovom slučaju, cela funkcija se sastoji od linije 7, koja štampa drugu poruku. Kada se DemonstrationFunctionO završi (linija 8), vraća se tamo odakle je pozvana. U ovom slučaju program se vraća na liniju 17, u kojoj mai n() štampa poslednju poruku.

Korišćenje funkcija

Funkcije vraćaju ili vrednost, ili void, što znači da ne vraćaju ništa. Funkcija koja sabira dva cela broja može da vrati zbir i zato bi se definisala da vraća celobrojnu (eng. integer) vrednost. Funkcija koja samo štampa poruku nema šta da vrati i zato se deklariše da vraća void.

Funkcija se sastoji od zaglavlja i tela. Zaglavlje se sastoji, po redosledu pojavljivanja, od tipa vrednosti koja se vraća, naziva funkcije i parametara funkcije. Parametri funkcije omogućavaju da se vrednosti proslede funkciji. Zato, ako funkcija treba da sabere dva broja, brojevi bi trebalo da budu parametri funkcije. Tipično zaglavlje funkcije izgleda ovako:

```
int Sum(int a, int b)
```

Parametar funkcije (eng. *parameter)* je deklaracija tipa vrednosti, koja će biti prosledena funkciji; stvarna vrednost prosledena pozivanjem funkcije naziva se argument funkcije. Mnogi programeri koriste ova dva pojma, parametre i argumente, kao sinonime. Drugi, s druge strane, vode računa o tehničkoj razlici. U ovoj knjizi termini će biti korišćeni kao sinonimi.

Telo funkcije se sastoji od otvorene velike zagrade, bez ijedne ili sa vise naredbi i zatvorene velike zagrade. Naredbe obavljaju posao za koji je funkcija namenjena. Funkcija može da vrati vrednost, koristeći naredbu return. To će, takođe, dovesti do kraja rada funkcije. Ukoliko ne stavite naredbu return u svoju funkciju, ona će, automatski, vratiti void na kraju funkcije. Vrednost koja se vraća mora biti tipa koji je deklarisan u zaglavlju funkcije.

^ШПРОШНк^ Funkcije su detaljno obradene u Danu 5, "Funkcije". Tipovi podataka koji mogu biti vraćeni iz funkcije su detaljno prikazani u Danu 3. Danas Vam je dat samo uvid u korišćenje funkcija, s obzirom da će se one koristiti u skoro svim Vašim C++ programima.

U listingu 2.5 prikazana je funkcija sa dva celobrojna parametra, koja vraća celobrojnu vrednost, nemojte brinuti o sintaksi ili specifičnosti rada sa celobrojnim vrednostima (na primer, i nt x). Za sada; to je u potpunosti "pokriveno" u Danu 3. *

Listing 2.5.: FUNKCIJA.CPP prikazuje jednostavnu funkciju

```
1   linclude <iostream.h>
2   int Add (int x, int y)
3   {
4
5     cout « "U Add(), preuzimam « y "\n-                    mi
6     return (x+y);
7
8
9   int main()
10
11          cout « "Ja sam u main()!\n";
12          int a, b, c;
13          cout « "Unesite dva broja: ";
14          cin » a;
15          cin » b;
16          cout « "\nPozivam Add()\n";
17          c=Add(a,b);
18          cout « "\nNazad u main().\n";
19          cout « "c ima vrednost " « c;
20          cout « "\nIzlazim... \n\n";
21          return 0;
22
```

```
Ja sam u main()!
Unesite dva broja: 3 5

Pozivam Add()
U Add(), preuzimam 3 i 5

Nazad u main().
c ima vrednost 8
Izlazim...
```

pr,.. Funkcija Add() je definisana u liniji 2. Preuzima dve celobrojne vrednosti i vraća jednu celobrojnu vrednost. Sam program počinje u linijama 9 i 11, gde štampa poruku. Program traži od korisnika da unese dva broja (linije 13 do 15). Korisnik unosi svaki broj, odvojen praznim mestom, i, zatim, pritiska taster Enter. main() prosleduje dva broja, koje je uneo korisnik kao argumente funkciji Add() u liniji 17.

Funkcija Add() počinje od linije 2. Parametri a i b su odštampani i zatim sabrani. Rezultat se vraća u liniji 6 i funkcija se završava.

u linijama 14 **i** 15 objekat cin **se** koristi da **bi se** dobili brojevi **za** promenljive a **i** b, a **cout se** koristi da **bi se** odštampale vrednosti **na** ekran. Promenljive i drugi aspekti **ovog** programa biće detaljnije istraženi **u** sledećih nekoliko dana.

Rezime

Teškoće **u** učenju složenih procesa, poput programiranja, leže u tome što mnogo stošta zavisi od onoga što tek treba da **se** nauči. Ovo poglavlje vas je uvelo u osnovne delove C + + programa. Ono **Vam je,** takođe, objasnUo razvojni ciklus **i** odreden broj značajnih novih pojmova.

Pitanja i odgovori

P Šta radi linclude?

O To je direktiva pretprocesoru, koji se izvršava kada pokrenete svoj C + + prevodilac. Ova specifična direktiva omogućava da se datoteka, koja se nalazi iza reči include, unese u program, umesto include direktive, baš kao da ste je sami lično uneli kucanjem.

P Koja je razlika između komentara // i /*?

O Komentar dupla kosa crta (//) "prestaje da važi" krajem linije. Kosa crta-zvezdica (/*) komentariše, sve dok se ne naide na zatvarajući deo (*/) •

P Šta razlikuje dobar od lošeg komentara?

0 Dobar komentar govori čitaocu zašto odredeni kod nešto radi, ili objašnjava čemu služi blok koda. Loš komentar drugačije izlaže ono što radi određena linija koda. Kod treba da bude pisan tako da bude jasan, sam po sebi. Čitanje koda treba da objasni šta kod radi, bez potrebe za komentarima.

Radionica

Nudimo Vam test, da Vam pomogne da utvrdite svoje razumevanje obradene teme, 1 vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i proverite da li razumete odgovore, pre nego što pređete na sledeće poglavlje.

Test

1. Koja je razlika između prevodioca i pretprocesora?
2. Zašto je funkcija mai n () posebna?
3. Koja su dva tipa komentara i po čemu se razlikuju?

4. Da li komentari mogu biti ugnježdeni?
5. Da li komentari mogu biti duži od jedne linije?

Vežbe

1. Napišite program koji štampa J a vol im C++ na ekran.
2. Napišite najmanji program koji se može prevesti, povezati i izvršiti.
3. **LOVCI NA GREŠKE:** Unesite sledeći program i prevedite ga. Zašto prevodenje ne uspeva? Kako to možete popraviti?

```
1: linclude <iostream.h>
2: void main()
3: {
4:       cout « Da li ovaj program ima grešku?";
5: }
```

4. Popravite grešku iz treće vežbe i ponovo prevedite program, pa ga povežite i izvršite.

lk*

Dan3

Promenljive i konstante

Programima su potrebni načini za čuvanje podataka koje koriste. Promenljive i konstante nude različite načine za predstavljanje i manipulaciju tim podacima.

Danas ćete naučiti

- kako da deklarišete i definišete promenljive i konstante
- kako da dodelite vrednosti promenljivima i kako da se koristite tim vrednostima
- kako da štampate vrednost promenljive na ekran.

v

Sta je to promenljiva?

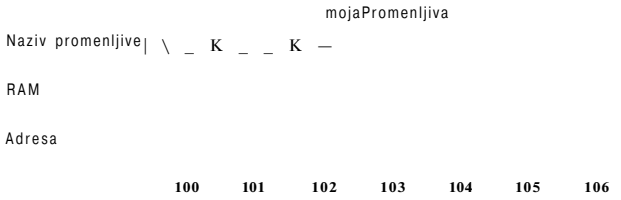
U programskom jeziku C + + *promenljiva* (eng. *variable)* je mesto za čuvanje informacija. Promenljiva je lokacija u memoriji Vašeg računara, u kojoj možete da sačuvate vrednost i iz koje kasnije možete da preuzmete tu vrednost.

Memorija Vašeg računara može se predstaviti kao niz odeljaka. Svaki odeljak je jedan od mnogih istih takvih, nanizanih jedan iza drugog. Svaki odeljak, ili memorijska lokacija, numerisan je sekvencijalno. Ti brojevi su poznati kao memorijske adrese. Promenljiva rezerviše jedan, ili vise odeljaka, u koje možete da smestite skladište neku vrednost.

Ime Vase promenljive (na primer, `mojaPromenljiva`) je oznaka na jednom od odeljaka, tako da ga možete Iako naći, bez znanja stvarne memorijske adrese promenljive. Na slici 3.1 možete videti shematski prikaz memorije. Kako možete videti sa slike,

L

mojaPromenljiva počinje na memorijskoj adresi 103. Zavisno od veličine, mojaPromenljiva može zauzeti jednu, ili vise memorijskih adresa.



Slika 3.1. Shematska prezentacija memorije

yJUPOMINA^. RAM je skraćenica od random access memory i predstavlja memoriju, kojoj se može pristupiti proizvoljnim redom. Kada pokrenete svoj program, on se učitava u RAM iz datoteke sa diska. Sve promenljive se, takode, kreiraju u RAM-u. Kada programeri pričaju o memoriji, obično je o RAM-u reč.

Rezervisanje memorije

Kada definišete promenljivu u programskom jeziku C++, morate da prevodiocu date do znanja o kojem tipu promenljive je reč: celobrojnoj (integer), znakovnoj (character) ili nekoj drugoj. Ove informacije saopštavaju prevodiocu koliko prostora da rezerviše i koju vrstu vrednosti želite da skladištite u svojoj promenljivoj.

Svaki odeljak ima veličinu od jednog *bajta* (eng. *byte)*. Ukoliko tip promenljive koju kreirate ima veličinu od dva bajta, tada on zahteva dva bajta memorije, ili dva odeljka. Tip promenljive (na primer, celobrojna) saopštava prevodiocu koliko memorije (koliko odeljaka) da rezerviše za promenljivu.

Pošto računari koriste bitove i bajtove da predstave vrednosti i pošto se memorija meri u bajtovima, važno je da razumete ove koncepte. Za puni uvid u ovu temu, pročitajte Dodatak B, "Službene red programskog jezika C++".

Veličina celobrojnih promenljivih

Na svakom pojedinom računaru, pojedini tipovi promenljivih zauzimaju jednu istu, nepromenljivu količinu prostora. To znači da bi celobrojna promenljiva (integer) na jednom računaru mogla da zauzima dva bajta, a četiri na drugom, ali da na istom tipu računara uvek zauzima istu kolidnu prostora.

Promenljiva tipa char (koristi se za smeštanje znakova) je, najčešće, dugačka jedan bajt. Celobrojna promenljiva tipa `short integer` na većini računara zauzima dva bajta, `long integer`, obično, zauzima četiri bajta, dok `integer` (bez službenih reči `short`, ili `long`) može zauzimati dva, ili četiri bajta. Program iz listinga 3.1 Vam može pomoći da odredite tačnu veličinu pojedinih tipova podataka na svom računaru.

. *Znak* je jedno slovo, broj, ili simbol, koji zauzima jedan bajt memorije.

Listing 3.1.: Određivanje veličine pojedinih tipova podataka na Vašem računaru

```
linclude <iostream.h>

int main()
{
  cout « "Veličina tipa int je: \t\t" « sizeof(int) « M bajta. \n";
  cout « "Veličina tipa short int je: \t" « sizeof(short) « " bajt. \n";
  cout « "Veličina tipa long int je: \t" « sizeof(long) « " bajta. \nM;
  cout « "Veličina tipa char je: \t\t" « sizeof(char) « M bajt. \n";
  cout « "Veličina tipa float je: \t\t" « sizeof(float) « " bajta. \nH;
10:   cout « "Veličina tipa double je: \t" « sizeof(double) « " bajtova. \n";
11:
12:         return 0;
13: }
```

```
IEIE^ Veličina tipa int je:              2 bajta.
      Veličina tipa short int je:        1 bajt.
      Veličina tipa long int je:         4 bajta.
      Veličina tipa char je:             1 bajt.
      Veličina tipa float je:            4 bajta.
      Veličina tipa double je:           8 bajtova.
```

^NAPOMENJp. Na Vašem računaru broj bajtova može biti različit.

TΠJJTJ^ Veći deo koda iz listinga 3.1 trebalo bi da Vam bude poznat. Novo je samo korišćenje funkcije `sizeofQ` u linijama 5 do 10. `sizeof()` obezbeduje od strane prevodioca i daje veličinu objekta koji joj se prosledi kao parametar funkcije. Na primer, u liniji 5 `int` je prosledena kao parametar u `sizeofQ`. Korišćenjem `sizeofQ`, saznao sam da je na mom računaru `int` iste veličine kao i `short int`, što iznosi 2 bajta.

signed i unsigned

Uz to, svi celobrojni tipovi se mogu javiti u dva oblika: `signed` (označeni) i `unsigned` (neoznačeni). Osnovna ideja je u tome da su Vam nekad potrebni negativni brojevi, a nekada ne. Celobrojni tipovi (`short` i `long`) bez reči unsigned su signed, po definiciji. `Signed` celobrojne pomenljive imaju ili negativnu, ili pozitivnu vrednost. `Unsigned` celobrojne pomenljive uvek imaju pozitivnu vrednost.

Pošto imamo isti broj bajtova, koje zauzimaju i `signed` i `unsigned` celobrojne promenljive, najveci broj koji se može skladištiti u `unsigned integer` je dva puta veći od najvećeg pozitivnog broja koji se može skladištiti u `signed integer`; `unsigned short integer` može da prima brojeve od 0 do 65.535. Pola brojeva koji mogu da se prestave sa `signed short` su negativni, tako da `signed short` može da predstavi samo brojeve od -32.768 do 32.767. Ako Vam ovo deluje konfuzno, obavezno pročitajte Dodatak A, "Prioritet operatora".

Osnovni tipovi promenljivih

Pored celobrojnih, u programskom jeziku C + + ugradeno je još nekoliko osnovnih tipova promenljivih. Oni se mogu zgodno podeliti u celobrojne tipove (o kojima smo do sada govorili), tipove za predstavljanje racionalnih brojeva u pokretnom zarezu i znakovne promenljive.

Promenljive tipova za predstavljanje racionalnih brojeva u pokretnom zarezu imaju vrednosti koje se mogu predstaviti u obliku razlomka. Znakovne promenljive sadrže jedan bajt i koriste se za čuvanje jednog od 256 znakova i simbola ASCII i proširenog ASCII skupa znakova.

ASCII skup znakova je standardizovan za korišćenje na računarima. ASCII je skraćenica od *American Standard Code for Information Interchange*. Skoro svaki postojeći računarski operativni sistem podržava ASCII, mada mnogi podržavaju i druge internacionalne skupove znakova.

Tipovi promenljivih, koji se koriste u C+ + programima opisani su u Tabeli 3.1. Ova tabela pokazuje tipove promenljivih, njihove veličine, i koja vrsta podatka može biti smeštena u tim promenljivim. Vrednosti koje mogu biti smeštene su odredene na osnovu veličine promenljive tog tipa.

Tabela 3.1.: Tipovi promenljivih

Tip	Yelicina	Vrednost
unsigned short int	2 **bajta**	0 do 65.535
short int	2 bajta	-32.768 do 32.767
unsigned long int	4 bajta	0 do 4.294.967.295
long int	4 bajta	-2.147.483.648 do 2.147.483.647
int **(16** bita)	2 bajta	-32.768 do 32.767
int (32 bita)	4 bajta	-2.147.483.648 do 2.147.483.647
unsigned int **(16** bita)	2 bajta	0 do 65.535
unsigned int (32 bita)	4 bajta	Odo 4.294.967.295
char	1 bajt	256 znakova
float	4 bajta	$1.2e^{-38}$ do $3.4e^{38}$
double	8 bajtova	$2.2e^{-308}$ dol.$8e^{308}$

^**NAPOMENA**^ Veličine promenljivih mogu biti razlicite od onih prikazanih u tabeli 3.1, zavisno od toga koji računar i prevodilac upotrebljavate. Ukoliko Vaš računar daje isti izlaz kao program u listingu 3.1, tabela 3.1 bi trebalo da važi i za Vaš prevodilac. Ukoliko je izlaz programa različit, proverite u priručniku za prevodilac koje vrednosti mogu da sadrie tipovi promenljivih kod Vas.

Definisanje promenljive

Promenljivu možete da kreirate, ili definišete, navodeći njen tip, unošenjem sa jednog ili vise razmaka, i navodeći naziv promenljive i tačka - zarez. Naziv promenljive može da bude, praktično, bilo koja kombinacija slova, ali ne može da sadrži prazna mesta. Ispravni nazivi promenljivih su x, `J23qrsnf` i `mojUzrast`. Dobri nazivi promenljivih Vam govore za šta se promenljiva koristi; korišćenje dobrih naziva čini lakšim razumevanje toka programa. Sledeća naredba definiše celobrojnu promenljivu po nazivu `mojUzrast`:

```
int mojUzrast;
```

Kao opštu programersku praksu, nemojte koristiti takve rogobatne nazive, poput `J23qrsnf`, i zadržite korišćenje naziva promenljivih dužine jednog znaka (poput x, ili i) za promenljive koje se koriste samo veoma kratko. Trudite se da koristite opisne nazive, poput `mojUzrast`, ili `kolikoJos`. Takvi nazivi se lakše razumeju tri nedelje kasnije, dok "razbijate" glavu nad tim šta ste mislili kada ste pisali tu liniju koda.

Izvedite sledeći eksperiment: pokušajte da pogodite šta sledeći programi rade, na osnovu prvih nekoliko linija koda.

Primer 1

```
main()
{
    unsigned short x;
    unsigned short y;
    ULONG z;
    z = x * y;
}
```

Primer 2

```
mai n()
{
    unsigned short Sirina;
    unsigned short Duzina;
    unsigned short Povrsina;
    Povrsina = Duzina * Sirina;
}
```

Jasno, drugi program je lakši za razumevanje i neprijatnost unošenja dugačkih naziva promenljivih je vise nego nadoknadena mnogo lakšim održavanjem drugog programa.

Razlikovanje velikih i malih slova

C+ + razlikuje mala i velika slova. Promenljiva `uzrast` je razhcita od promenljive Uzrast, koja je opet razliČita od UZRAST.

^tfOMBl^ Neki prevodioci dozvoljavaju da im se isključi razlikovanje velikih i malih slova. Nemojte to pokusavati; Vaši programi neće raditi sa drugim prevodiocima i drugi C+ + programeri će biti veoma zbunjeni Vašim kodom.

Postoje različite konvencije o tome kako treba imenovati promenljive i mada nije mnogo značajno koju ćete usvojiti, važno je da budete dosledni u njenoj promeni u svojim programima.

Mnogi programeri koriste sva mala slova u nazivima svojih promenljivih. Ukoliko naziv zahteva dve reči (na primei; mojajcola), postoje dve popularne konvencije za imenovanje: moja_kola ili mojaKola. Poslednja konvencija se u žargonu naziva "kamilja konvencija" (eng. camel-notation), zato što veliko slovo izgleda poput grbe kamile.

Neki ljudi smatraju da im korišćenje *donje crte* (eng. *underscore character)* omogućava da lakše čitaju kod, dok drugi zaobilaze korišćenje donje crte, zato što im je teže da je kucaju. Ova knjiga koristi "kamilju konvenciju", u kojoj druga i svaka sledeća reč počinje velikim slovom: mojaKola, brzaBraonLisica i tako dalje.

^JIAPOMEM^ Mnogi iskusni programeri koriste notaciju često opisanu kao Madarska notacija. Ideja je da svakoj promenljivoj prethodi skup karaktera koji opisuju njen tip. Celobrojne promenljive tako mogu da počnu malim slovom i, promenljive tipa long sa l. Druge notacije ukazuju na konstante, globalne promenljive, pokazivače i drugo. Ovo je mnogo vise važno u C programiranju, pošto C+ -f podržava kreiranje korisnički definisanih tipova (pogledajte Dan 6, "Osnovne klase") i zato što je C+ + strogo tipiziran.

Službene reči

Pojedine reči je rezervisano programski jezika C + + i ne mogu da se koriste kao nazivi promenljivih. Njih koristi prevodilac za kontrolu programa. Službene red uključuju if, while, for i main. Uputstvo za korišćenje Vašeg prevodioca treba da sadrži kompletnu listu službenih reči, ali, generalno, ma koji naziv za promenljivu, skoro sigurno nije službena reč.

1 РЛПП:;J^ Promenljivu definišete, navođenjem njenog tipa, pa, zatim, naziva.

Koristite nazive promenljivih sa značenjem.

Zapamtite da C + + razlikuje velika i mala slova.

Nemojte koristiti službene reči C + + za nazive promenljivih.

Obratite pažnju na broj bajtova, koje svaki tip promenljive zahteva u memoriji, i koje vrednosti mogu biti smeštene u promenljivu tog tipa.

Nemojte koristiti unsigned promenljive za negativne brojeve.

Kreiranje vise od jedne promenljive istovremeno

Možete kreirati vise od jedne promenljive istog tipa u istoj naredbi, tako što navedete tip promenljive i, zatim, navedete nazive promenljivih, odvojene zarezima. Na primer:

```
unsigned int mojeGodiste, mojaTezina      // dve unsigned int promenljive
long povrsina, sirina, duzina             // tri long promenljive
```

Kao što možete videti, mojeGodiste i mojaTezina su deklarisane kao unsigned integer promeljive. Druga linija deklariše tri pojedinačne long promenljive sa nazivima povrsina, sirina i duzina. Tip (long) je dodeljen svim promenljivim, tako da ne možete mešati tipove u jednoj naredbi za definiciju.

Dodeljivanje vrednosti promenljivim

Promenljivoj dodeljujete vrednost, koristeći operator dodeljivanja (=). Tako ćete promenljivoj Width dodeliti vrednost 5 sa

```
unsigned short Width;
Width = 5;
```

Možete kombinovati ova dva koraka i inicijalizovati promenljivu Width prilikom njenog deklarisanja sa

```
unsigned short Width = 5;
```

Inicijalizacija veoma lid na dodeljivanje vrednosti i kada je reč o celobrojnim promenljivim, razlika je beznačajna. Kasnije, kada upoznate konstante, videćete da se neke vrednosti moraju inicijalizovati, pošto ih ne možemo dodeliti. Osnovna razlika je da inicijalizacija nastupa u trenutku kreiranja promenljive.

Kao što možete istovremeno definisati vise od jedne promenljive, tako možete i inicijalizovati vise od jedne promenljive prilikom njihovog kreiranja. Na primer:

```
// kreiranje dve long promenljive i njihova inicijalizacija long width =5, length = 7;
```

U ovom primeru inicijalizuje se long celobrojna promenljiva width sa 5 i long celobrojna promenljiva length sa 7. Možete mešati deklaracije i inicijalizacije

```
int myAge = 39, yourAge, hisAge = 40;
```

Ovaj primer kreira tri promenljive tipa int i inicijalizuje prvu i treću.

U listingu 3.2 prikazan je ceo program, spreman za prevodenje, koji izračunava površinu pravougaonika i štampa ga na ekran.

Listing 3.2.: Demonstracija upotrebe promenljivih

```
1:  // Demonstracija upotrebe promenljivih
2:  #include <iostream.h>
3:
4:  int main()
```

nastavlja se

нмшпрпт **Listing 3.2.: Demonstracija upotrebe promenljivih** ***nastavak***

```
5
6       unsigned short int Width = 5, Length;
7       Length = 10;
8
9          // kreiranje unsigned short celobrojne promenljive i njena
10            // inicijalizacija sa rezultatom množenja Width sa Length
11         unsigned short int Area  = Width * Length;
12
13         cout « "širina:" « Width « "\n";
14         cout « "Duzina: " « Length « endl;
15         cout « "Povrsina: " « Area « endl;
16            return 0;
17
```

```
pH»:V|lfr  Širina:5
           Dužina: 10
           Povrsina: 50
```

Linija 2 sadrži include naredbu za iostream biblioteku, koju traži cout.

U liniji 4 počinje program. U liniji 6 Width je definisan kao unsigned short celobrojna promenljiva, i njena vrednost je inicijalizovana na 5. Druga unsi gned short celobrojna promenljiva je takode definisana, ali nije inicijalizovana. U liniji 7 vrednost 10 je dodeljena promenljivoj Length.

U liniji 11, unsigned short celobrojna promenljiva Area je definisana i inicijalizovana vrednošću proizvoda Width sa Length. U linijama 13 do 15 vrednosti promenljivih su odštampane na ekranu. Primetićete da specijalna reč endl kreira novu liniju.

t y p e d e f

Stalno pisanje unsigned short int je krajnje monotono, dosadno i, što je najvažnije, veoma je podložno greškama. C+ + omogućava kreiranje pseudonima za ovu frazu, korišćenjem službene red typedef, što je skraćenica za type definition - definisanje tipa.

U stvarnosti, sa typedef se kreira sinonim i važno je razlikovati ga od kreiranja novog tipa (što ćete i mod posle Dana 6); typedef se koristi pisanjem službene red typedef, iza koje sledi postojed tip i novo ime. Na primer,

```
typedef unsigned short int USHORT
```

kreira novo ime USHORT, koje možete koristiti svuda gde biste, inače, pisali unsigned short int. Listing 3.3 je isti kao listing 3.2, samo se koristi definicija tipa USHORT, umesto unsigned short int.

Listing 3.3. Demonstracija upotrebe t y p e d e f

```
// *****************
// Demonstracija upotrebe typedef
linclude <iostream.h>

typedef unsigned short int USHORT;  //definisanje tipa

void main()
{
   USHORT  Width = 5;
   USHORT Length;
   Length = 10;
   USHORT Area  = Width * Length;
   cout « "Sirina:" « Width « "\n";
   cout « "Duzina: " « Length « endl;
   cout « "Povrsina: " « Area « endl;
}
```

```
Sirina:5
Dužina: 10
Povrsina: 50
```

U liniji 5, USHORT je definisan tip kao sinonim za unsigned short int. Program je veoma sličan programu iz listinga 3.2, a izlaz programa je isti.

Kada se koriste s h o r t , a kada l o n g celobrojne promenljive

Jedan izvor konfuzije za nove C + + programere je dilema kada treba koristiti promenljivu tipa long, a kada tipa short. Pravilo je veoma jednostavno: ako postoji ikakva šansa da će vrednost koju želite da stavite u promenljivu biti suviše velika za njen tip, koristite ved tip.

Kao što se videlo u tabeli 3.1, unsigned short celobrojne promenljive, pretpostavljajud da ima dva bajta, mogu da primaju vrednosti do 65.535; signed short celobrojne promenljive mogu da primaju samo polovinu od toga. Mada unsi gned 1 ong celobrojne promenljive mogu da sadrže veoma velike brojeve (4.294.967.295) to je i dalje veoma ograničeno. Ako su Vam potrebni ved brojevi, moraćete da koristite, f 1 oat ili double, i tada ćete izgubiti određenu tačnost. Float i double mogu da sadrže veoma velike brojeve, ali samo prvih sedam ili 19 cifara je značajno na vedni računara. To znači da se brojevi zaokružuju posle toliko cifara.

Prekoracenje kod unsigned celobrojne promenljive

Činjenica da unsigned long celobrojne promenljive mogu sadržati vrednosti do odredene granice je veoma retko problem, ali šta se dešava kada se prekorači najveća moguća vrednost, koja može biti sadržana u njima?

Kada unsigned long celobrojna promenljiva dostigne svoju maksimalnu vrednost, njena vrednost se postavi na nulu i počne ispočetka, baš poput merača predenog puta u kolima. Programom iz listinga 3.4 pokazuje se šta se dogada kada pokušate da dodelite preveliku vrednost short celobrojnoj promenljivoj.

Listing 3.4.: Demonstracija unešenja suviše velike wrednosti u unsigned celobrojnu vrednost

```
1:   #include <iostream.h>
2:    int main()
3:   {
4:      unsigned short int smallNumber;
5:     smallNumber = 65535;
6:       cout « "mali broj:" « smallNumber « endl;
7:     smal1Number++;
8:       cout « "mali broj:" « smallNumber « endl;
9:     smallNumber++;
10:      cout « "mali broj:" « smallNumber « endl;
11:        return 0;
12: }
```

```
Ш М Г У ^ К  mali broj:65535
             mali broj:0
             mli broj:l
```

J Щ J \ < ^ * U 4> smallNumber je definisan kao `unsigned short int`, što je na mom računaru promenljiva od dva bajta, sposobna da sadrži vrednost između 0 i 65.535. U liniji 5 maksimalna vrednost je dodeljena promenljivoj `smal l Number` i štampa se u liniji 6.

U liniji 7 `smallNumber` inkrementiramo, odnosno povećavamo za jedan. Znak za inkrementiranje je ++ (kao u imenu C++ - inkrement od C). Sada `bi` vrednost `u smallNumber` trebalo da bude `65.536`. Pošto `unsigned short` celobrojne promenljive ne mogu da sadrže brojeve veće od `65.535`, vrednost je ponovo `0`, koja se štampa u liniji 8.

U liniji 9 `smal l Number` se opet inkrementira i njegova nova vrednost `1` se štampa.

Prekoracenje kod signed celobrojne promenljive

Celobrojna promenljiva `signed` se razlikuje od `unsigned` celobrojne promenljive po tome što je pola vrednosti koje može da predstavi negativna. Umesto korišćenja tradicionalnog merača pređenog puta u kolima, pokušajte da predstavite jedan koji rotira gore za pozitivne brojeve i dole za negativne brojeve. Jedna milja od 0 je `ili` 1, ili -1. Kada izadete iz opsega pozitivnih brojeva, naići ćete na najmanji negativan broj

i zatim će se brojati nazad ka 0. Listing 3.5 pokazuje šta se događa kada dodate 1 maksimalnom pozitivnom broju u unsigned short celobrojnoj promenljivoj.

Listing 3.5.: Demonstracija unešenja suviše velike wrednosti u signed celobrojnu vrednost

```
1   linclude <iostream.h>
2   int main()
3   {
4      short int smallNumber;
5      smallNumber = 32767;
6      cout « "mali broj:" « smallNumber « endl;
7      smal1Number++;
8      cout « "mali broj:" « smallNumber « endl;
9      smallNumber++;
10     cout « "mali broj:" « smallNumber « endl;
11         return 0;
12  }
```

```
mali broj:32767
mali broj:-32768
mali broj:-32767
```

U liniji 4 `smallNumber` je, `ovog` puta, deklarisan `kao signed short int` (ukoliko eksplicitno ne naznačite da je promenljiva `unsigned`, podrazumeva se da je `si gned`). Program zatim nastavlja poput prethodnog, ali izlaz je prilično različit. Da biste potpuno razumeli njegov izlaz, morate znati kako su `si gned` brojevi predstavljeni kao bitovi u dvobajtnoj celobrojnoj promenljivoj. Za detalje, pogledajte Dodatak C, "Binarni i heksadecinalni brojevi".

Poput `unsigned` celobrojnih promenljivih, i `signed` celobrojne promenljive prelaze sa svoje najveće pozitivne na najmanju negativnu vrednost.

Karakteri

Promenljive tipa karakter `(type char)` su obično dugačke jedan bajt, što je dovoljno da prime 256 vrednosti (pogledajte Dodatak C). char se može interpretirati kao mali broj (0-255), ili kao clan ASCII skupa. *ASCII* je skraćenica od *American Standard Code for Information Interchange* (Standardni američki kod za razmenu podataka). ASCII skup karaktera i njegov ISO (International Standards Organization - Medunarodna organizacija za standardizaciju) ekvivalent predstavljaju način za kodiranje svih slova, cifara i znakova interpunkcije.

^MAPOMEHA^ Računari ne znaju ništa o slovima, znacima za interpunkciju, ili rečenicama. Sve što oni razumeju su brojevi. U stvari sve što oni zaista znaju jeste da li ima, ili nema dovoljne kolidne elektriciteta u odredenom spoju žica. Ako ima, to je inferno predstavljeno kao 1, ako nema, predstavljeno je kao 0. Grupisanjem nula i jedinica, računar je u stanju da generiše obrasce, koji se mogu predstaviti kao brojevi, a oni se, zatim, mogu dodeliti slovima i interpunkcijskim znacima.

U ASCII kodu malom slovu "a" je dodeljena vrednost 97. Svim malim i velikim slovima, svim ciframa i svim interpunkcijskim znacima su dodeljene vrednosti između 1 i 128. Sledećih 128 znakova i simbola je rezervisano za proizvodača računara, mada je prošireni IBM-ov skup karaktera postao nešto standardno.

Karakteri i brojevi

Kada dodelite karakter, na primer, 'a' promenljivoj tipa char, ono što je zaista tamo je, zapravo, broj između 0 i 255. Prevodioci, medutim, znaju kako da povežu karaktere (predstavljene jednostrukim znakom navoda, slovom, brojem, znakom interpunkcije, zatvarajućim jednostrukim znakom navoda) sa njihovom ASCII vrednosti.

Odnos vrednost/slovo je odreden arbitrarno; nema posebnog razloga da se malom slovu "a" dodeli vrednost `97`. Sve dok se svi (tastatura, prevodilac i monitor) slažu, nema problema. Ipak, veoma je važno shvatiti razliku između vrednosti 5 i karaktera `'5'`. Simbol `'5'`, zapravo, ima vrednost `53`, baš kao što slovo 'a' ima vrednost `97`.

Listing 3.6.: Štampanje karaktera, zasnovano na brojevima

```
1:  linclude  <iostream.h>
2:  int  main()
3:  {
4:  for  (int i  = 32;  i<128;  i++)
5:          cout «  (char)  i;
6:          return 0;
7:  }
```

IZLAZ

```
 !"#$%G"()*+,./0123456789:;<>?@ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^abcdefghij klmnopqrstuvwxyz<|>-s
```

Ovaj jednostavan program starnpa vrednosti karaktera od 32 do 127.

Specijalni znaci za štampanje

C++ prevodilac prepoznaje neke specijalne karaktere za formatiranje. Tabela 3.2 prikazuje one koji se najčešće koriste. Možete ih koristiti u svom kodu, kucajući obrnutu kosu crtu (koja se naziva *specijalni karakter* - eng. *escape character),* praćenu karakterom. Na taj način, da biste uneli tab karakter u svoj kod, unećete jednostruki znak navoda, obrnutu kosu crtu, slovo t i zatim zatvarajući jednostruki znak navoda:

```
char  tabCharacter  =  '\t';
```

Ovaj primer deklariše char promenljivu (tabCharacter) i inicijalizuje ga karakterom vrednosti \t, odnosno tabom. Specijalni znaci za štampanje se koriste za štampanje na ekran, u datoteku, ili na neki drugi izlazni uredaj.

• *Specijalni karakter* menja značenje karaktera koji ga sledi. Na primer, obično karakter **n** znači slovo **n,** ali kada mu prethodi specijalni karakter to označava novu liniju.

Tabela 3.2.: Specijalni karakteri

Karakter	Šta znači
\n	nova linija (new line)
\t	tabulator (tab)
\b	povratak za jedan karakter nazad (backspace)
\"	dvostruki navodnik (double quote)
V	jednostruki navodnik (single quote)
\?	znak pitanja (question mark)
w	obrnuta kosa crta (backslash)

Konstante

Poput promenljivih, i konstante su lokacije za smeštanje podataka. Za razliku od promenljivih, kao što im i samo ime ukazuje, konstante se ne menjaju. Morate inicijalizovati konstantu kada je kreirate i ne možete joj kasnije dodeliti novu vrednost.

Slovne konstante

C++ ima dva tipa konstanti: slovne i simboličke.

Slovna konstanta {literal constant) je vrednost koja se direktno unosi u program tamo gde je potrebna. Na primer:

```
int myAge = 39;
```

`myAge` je promenljiva tipa int, `39` je slovna konstanta. Ne možete dodeliti vrednost u `39` i njena vrednost se ne može menjati.

Simboličke konstante

Simboličke konstante su predstavljene nazivom, baš kao što je promenljiva predstavljena. Za razliku od promenljivih, medutim, pošto se konstanta inicijalizuje, njena vrednost se ne može promeniti.

Ako program ima jednu celobrojnu promenljivu nazvanu students i drugu nazvanu classes, možete izračunati koliko studenata imate, ako je poznat broj odeljenja, ukoliko znate da jedno odeljenje ima 15 studenata:

```
students = classes * 15;
```

^umh IIIy * označava množenje.

U ovom primeru 15 je doslovna konstanta. Program bi bio lakši za čitanje i održavanje, ukoliko bismo zamenili tu vrednost simboličkom konstantom:

```
students = classes * studentsPerClass;
```

Ukoliko kasnije odlučite da promenite broj studenata u svakom odeljenju, to možete uraditi tamo gde ste defmisali konstantu studentsPerClass, a da ne morate menjati broj na svakom mestu gde ste ga koristili u programu.

Postoje dva načina za deklarisanje simboličke konstante u programskom jeziku C+ + . Stari, tradicionalni i sada zastareo način je korišćenjem pretprocesorske naredbe, #define.

Definisanje konstante sa #define

Da biste definisali konstantu na tradicionalan način, unesite sledeće:

```
#define studentsPerClass 15
```

Primetite da studentsPerClass nema odreden tip (int, char itd); Idefine obavlja jednostavnu zamenu teksta. Svaki put kada pretprocesor naiđe na reč studentsPerClass, on ga zamenjuje u tekstu sa 15.

Pošto se pretprocesor izvršava pre prevodioca, prevodilac sam nikada neće videti konstantu, već broj 15.

Definisanje konstante sa const

Mada #define radi, postoji novi, mnogo bolji način definisanja konstanti u programskom jeziku C + +:

```
const unsigned short int studentsPerClass = 15;
```

Ovaj primer, takode, deklariše simboličku konstantu nazvanu studentsPerClass, ali ovoga puta studentsPerClass se unosi kao unsigned short int. Ovaj metod ima prednost, utoliko što čini kod lakšim za održavanje i sprečava greške. Najveća razlika je u tome što ova konstanta ima tip i prevodilac može da proven da li se ona koristi u skladu sa njenih tipom.

yHAPOMIHAy- Konstante se ne mogu menjati, dok se program izvršava. Ukoliko morate da promenite vrednost za studentsPerClass, na primer, morac'ete da promenite kod i da ga ponovo prevedete.

4j| **PAZrn** Nemojte koristiti tip int. Koristite short i long da biste jasno označili koju veličinu broja želite.

Pazite na brojeve koji prelaze veličinu celobrojnih tipova promenljivih i koji dovode do pogrešnih vrednosti u promenljivim.

Koristite nazive promenljivih sa značenjem koje ukazuje na to za šta ih koristite.

Nemojte koristiti službene reči za nazive promenljivih.

Konstante nabrajanja

Konstante nabrajanja omogućavaju kreiranje novih tipova i, zatim, definisanje promenljivih tih tipova, čije su vrednosti ograničene na skup mogućih vrednosti. Na primer, možete definisati COLOR kao konstantu nabrajanja i da postoji pet mogućih vrednosti za COLOR: RED, BLUE, GREEN, WHITE i BLACK.

Sintaksa za definisanje konstanti nabrajanja je da se napiše službena reč enum, koju slede naziv tipa, otvorena vitičasta zagrada, svaka dozvoljena vrednost, odvojena zarezom, i zatvorena vitičasta zagrada. Na primer:

```
enum COLOR (RED, BLUE, GREEN, WHITE, BLACK );
```

Ova naredba obavlja dva posla:

1. Čini COLOR nazivom za konstante nabrajanja, odnqsno novim tipom.
2. Čini 4 simboličkom konstantom sa vrednošću 0, BLUE simboličkom konstantom sa vrednošću 1, GREEN simboličkom konstantom sa vrednošću 2 i tako dalje.

Svaka konstanta nabrajanja ima celobrojnu vrednost. Ukoliko drugačije ne odredite, prva konstanta će imati vrednost 0, a ostale će se računati od nje. Svaka pojedinačna konstanta može biti inicijalizovana sa odredenom vrednošću, a ona koja nije inicijalizovana će se računati od one pre nje. Tako da ako unesete

```
enum COLOR {RED=100, BLUE, GREEN=500, WHITE, BLACK=700};
```

RED će imati vrednost 100, BLUE 101, GREEN 500, WHITE 501 i BLACK 700.

Možete da definišete promenljive tipa COLOR, ali njima se mogu dodeliti samo vrednosti nabrajanja (u ovom slučaju RED, BLUE, GREEN, WHITE ili BLACK, odnosno 100, 101, 500, 501 ili 700). Možete dodeliti ma koju vrednost boje COLOR promenljivoj. U stvari, možete dodeliti ma koji ceo broj, čak i ako nije dozvoljena boja, ali će dobar prevodilac istaći upozorenje prilikom prevodenja programa. Važno je razumeti da su prebrojive promenljive, u stvari, tipa unsigned int i da su konstante nabrajanja jednake celobrojnim promenljivama. Medutim, veoma je poželjno imati mogućnost imenovanja tih vrednosti kada radite sa bojama, danima u nedelji, ili sličnim skupovima vrednosti. Program koji koristi tip nabrajanja je predstavljen u listingu 3.7.

Listing 3.7.: Demonstracija nenumerisane konstante

```
#include <iostream.h>
int  main()
{
     enum Days { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, _Saturday

     Days DayOff;
     int x;
```

nastavlja sc

Listing 3.7.: Demonstracija nenumerisane konstante *nastavak*

```
9:      cout « "Koji dan želite da Vam bude Slobodan (0-6)?
        cm   » x;
        DayOff = Days(x);

        if (DayOff == Sunday |j DayOff == Saturday)
        cout « "\nVikendom ste, ionako, slobodni! \n";
        else
        cout « "\nU redu, upisaću taj dan kao Slobodan. \n"
         return 0;
```

```
Koji dan želite da Vam bude Slobodan (0-6)? 1

U redu, upisaću taj dan kao Slobodan.

Koji dan želite da Vam bude Slobodan (0-6)? 0

Vikendom ste ionako slobodni!
```

ДЖШjj^ ^ ^ ^e ^enn^sana PreDr°Jiva konstanta DAYS, sa nekoliko vrednosti, računajući od nule. Korisnik se pita da unese broj dana u liniji 9. Izabrani broj, izmedu 0 i 6, se poredi u liniji 13 sa prebrojivim vrednostima za Sunday i Saturday i u zavisnosti od toga koji je broj unet, preduzima se odgovarajuća akcija. Naredba if će biti detaljno objašnjena u Danu 4, "Iskazi i izrazi".

Ne možete uneti reč "Sunday" kada vas program pita za dan; program ne zna kako da prevede karaktere u Sunday u jednu od nabrojanih vrednosti.

^MAPOALLIO^ U ovom i svim malim programima u ovoj knjizi nema provere ispravnosti unetih podataka. Na primer, ovaj program ne proverava, kao što bi to uradio pravi program da li je korisnik uneo broj izmedu 0 i 6. Ovaj problem je izostavljen da bi programi bili što manji i jednostavniji i da bi mogli da se usredsredimo na trenutnu temu.

Rezime

U ovom poglavlju su razmatrane numeričke i znakovne promenljive i konstante, koje koristi programski jezik C++ za čuvanje podataka, tokom izvršavanja programa. Numeričke promenljive mogu biti ili celobrojne (`char`, `short` i `long int`) ili realne (`float` i `double`). Numeričke promenljive mogu biti `signed` ili `unsigned`. Mada tipovi mogu biti različite dužine na različitim računarima, tip je uvek iste dužine na određenom računaru.

Promenljivu morate deklarisati, pre nego što je koristite i tada joj morate dodeljivati podatke onog tipa koji ste deklarisali kao ispravan za tu promenljivu. Ukoliko dodelite preveliki broj celobrojnoj promenljivoj, ona će pokazati neispravnu vrednost zbog prekoračenja.

U ovom poglavlju takođe su objašnjene slovne i simboličke konstante, kao i nabrojive konstante i pokazana dva načina za deklarisanje simboličkih konstanti: korišćenjem #def i ne i korišćenjem službene reči const.

Pitanja i odgovori

P **Ukoliko kod short int može doći do prekoračenja, zašto uvek ne koristiti long celobrojan tip?**

O I short i long celobrojne promenljive mogu prekoračiti maksimalnu moguću vrednost, ali kod long celobrojnih promenljivih će se to desiti kod mnogo većih brojeva. Na primer, kod unsigned short int će prekoracenje nastupiti posle 65.535, dok će kod si gned short i nt prekoracenje nastupiti posle 4.294.967.295. S druge strane, na većini računara 1 ong celobrojna promenjiva će zauzeti dva puta vise memorije svaki put kada je deklarišete (4 bajta prema 2 bajta kod short) i program sa 100 takvih promenljivih će zauzeti 200 bajtova RAM-a vise. Danas je ovo mnogo manji problem nego ranije, pošto se većina ličnih računara danas isporučuje sa mnogo hiljada (ako ne i miliona) bajtova memorije.

P **Šta će se dogoditi ako dodelite broj sa decimalnom tačkom celobrojnoj** promenljivoj, **umesto realnoj? Pogledajte sledeću liniju koda:**

```
int aNumber = 5.4;
```

O Dobar prevodilac će dati upozorenje, ali dodela je potpuno legalna, samo će broj koji dodajete biti odsečen u celobrojni. Tako će celobrojna promeljiva kojoj dodajete vrednost 5.4, imati na kraju vrednost 5. Informacija će, tako, biti izgubljena i ukoliko pokušate, zatim, da dodelite vrednost te promenljive realnoj promenljivoj tipa `float`, ona će imati vrednost 5.

P **Zašto ne koristiti doslovne konstante; zbog čega se treba brinuti o** korišćenju simboličkih **konstanti?**

O Ukoliko neku vrednost koristite na mnogo mesta u programu, simbolička konstanta Vam omogućava da promenite sve vrednosti, menjajući samo onu koja inicijalizuje konstantu. Simboličke konstante, takode, govore same za sebe. Može biti teško za razumevanje zašto se broj množi sa 360, ali je potpuno jasno šta se dešava ako se broj množi sa degreesInACircle (broj stepena u krugu).

P Sta se dogada kada dodelimo negativan **broj** promenljivih **tipa** unsigned? Pogledajte sledeću liniju koda:

```
unsigned int aPositiveNumber = -1;
```

O Dobar prevodilac će dati upozorenje prilikom prevodenja, ali dodela je potpuno moguća. Negativnom broju će se pristupiti kao nizu bitova i on će se dodeliti promeljivoj. Takva vrednost promenljive će potom biti interpretirana kao unsigned broj. Na taj način, -1, čiji obrazac bitova je 11111111 11111111 (oxFF u

heksadecimalnom obliku), biće određen kao neoznačena vrednost 65535. Ukoliko vas ovo zbunjuje, pogledajte Dodatak C.

P Da li mogu da radim sa programskim jezikom C + + bez razumevanja obrazaca bitova, binarne aritmetike i heksadecinalnih brojeva?

O Da, možete, ali ne tako efektivno kao što biste radili ako razumete ove pojmove. C + + ne radi tako dobar posao kao neki drugi programski jezici "štiteći" vas od onog što računar zaista radi. Ovo je u stvari prednost, pošto vam daje snagu koju drugi jezici ne omogućavaju. A kao i sa svakim moćnim alatom, da biste izvukli maksimum iz programskog jezika C+ + , morate razumeti kako funkcioniše. Programeri koji pokušaju da programiraju u programskom jeziku C + + , bez razumevanja osnova binarnog sistema često bivaju zbunjeni rezultatima.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje obradene teme i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i proverite da li razumete odgovore, pre nego što predete na sledeće poglavlje.

Test

1. Koja je razlika između celobrojne i realne promenljive?
2. Koja je razlika izmedu unsigned short int i long int?
3. Koje su prednosti korišćenja simboličkih nad slovnim konstantama?
4. Koje su prednosti korišćenja službene reči const nad #def i ne?
5. Šta čini naziv promenljive dobrim, ili lošim?
6. Posmatrajući ovu nabrojivu konstantu, koju vrednost ima BLUE?

```
enum COLOR {WHITE, BLACK=100, RED, BLUE, GREEN=300};
```

7. Koje od ovih promenljivih imaju dobre nazive, koje lose, a koje su nedozvoljene?

   a. Age

   b. !ex

   c. R79J

   d. Total Income

   e. Invalid

1. Koji bi tip promenljive bio pogodan za čuvanje sledećih informacija?
   a. starosti
   b. površine dvorišta
   c. broja zvezda u galaksiji
   d. prosečne količine padavina u mesecu januaru.
2. Nadite dobre nazive promenljivima za te informacije.
3. Deklarišite konstantu pi kao 3.14159.
4. Deklarišite promenljivu tipa float i inicijalizujte je upotrebom konstante pi.

Dan 4

Iskazi i izrazi

U osnovi, program je niz naredbi koje se izvršavaju odredenim redosledom. Snaga programa proističe iz mogućnosti da uzvrši jednu, ili drugu grupu komandi, u zavisnosti od toga da li je odredeni uslov ispunjen, ili ne. Danas ćete naučiti

- šta su iskazi
- šta su blokovi
- šta su izrazi
- kako se program grana u zavisnosti od uslova
- kada je uslov ispunjen i kako na to reagovati.

Iskaz

U programskom jeziku C+ + naredbe kotrolišu tok izvršavanja programa, vrednuju izraze, ili ne rade ništa *(prazan iskaz* - eng. *null statement)*. Svi iskazi programskog jezika C + + se završavaju tačka-zarezom (;), Čak i prazan iskaz, koji čini samo tačka-zarez i ništa vise. Jedan od najčešće korišćenih iskaza je iskaz dodele:

```
x = a + b;
```

Za razliku od algebre, gornji iskaz ne znači da je x jednako a + b. Iskaz se čita kao "Dodeli x vrednost zbira a i b" ili, "Dodeli x, a + b " . Iako ovaj iskaz obavlja dva posla, on je samo jedan iskaz i ima samo jedan tačka-zarez. Operator dodele dodeljuje sve što je na desnoj strani znaka jednakosti onome §to je na levoj strani.

IIIjIIMj *Prazan iskaz* je iskaz koji ne radi ništa.

Praznine

Praznine (tabulatori, prazna mesta i novi red) se, obično, ignorišu u iskazima. Iskaz dodele iz prethodnog primera bi mogao da se napiše potpuno korektno i kao

```
x=a+b;
```

ili kao

```
x                    =a
+          b         ;
```

Mada je poslednja verzija potpuno ispravna, ona je i veoma nerazumljiva. Praznine se mogu koristiti da bi program napravile razumljivim i lakšim za održavanje, ili da ga učine nerazumljivim. Kao i drugde, C + + obezbeduje snagu; procenu morate uraditi sami.

чШ^МШ} Znakovi praznina (prazna mesta, tabulaton i novi redovi) se ne mogu videti. Kada se ti karakteri odštampaju, možete videti samo prazan papir.

Blokovi i složeni iskazi

Na svakom mestu na kome možete postaviti iskaz, možete uneti i složeni iskaz, poznat kao blok. Blok počinje otvorenom vitičastom zagradom (() i završava se zatvorenom vitičastom zagradom (}). Iako se svaki iskaz u bloku mora završavati tačka-zarezom (;), sam blok se njime ne završava. Na primer,

```
{
     temp = a;
     a =b;
     b = temp;
}
```

Ovaj blok koda radi kao jedan iskaz i zamenjuje vrednosti promenljivih a i b.

4| PAzm Koristite zatvorenu vitičastu zagradu svaki put kada imate otvorenu vitičastu zagradu.

Zavrsavajte iskaze tačka-zarezom.

Pažljivo koristite praznine, da učinite svoj kod jasnijim.

Izrazi

Sve što se može izračunati u vrednosti je *izraz* u programskom jeziku C + + . Izraz vraća vrednost. Tako, na primer, 3 + 2 vraća vrednost 5 i zato predstavlja izraz. Svi izrazi su iskazi.

Mnoštvo delova koda, koji, zapravo, jesu izrazi, mogu Vas iznenaditi. Na primer,

```
3-2                  // vraća vrednost 3.2
PI                   // konstanta tipa float koja vraća vrednost 3.14
SecondsPerMinute     // konstanta tipa int koja vraća 60
```


Pretpostavljajući da je PI konstanta sa vrednošću 3,14 i da je SecondsPerMinute konstanta sa vrednošću 60, sva tri iskaza su izrazi.

Složeni izraz

x = a + b;

ne samo da sabira a i b i dodeljuje vrednost zbira x-u, već, takođe vraća vrednost zbira (vrednost promenljive x). Na taj način, gornji iskaz je takode izraz, pa se, zato, on se može nalaziti na desnoj strani operatora dodele:

```
y = x = a + b;
```

Ova linija se obrađuje na sledeći način:

Saberi a i b.
Dodeli zbir izraza a+b promenljivoj x.
Dodeli rezultat izraza dodeljivanja x=a+b promenljivoj y.

Ako su a, b, x i y celobrojne promenljive i ako a ima vrednost 2 i b ima vrednost 5, i x i y imaće dodeljenu vrednost 7.

Listing 4.1.:

```
      linclude <iostream.h>
2     int  main()
3     {
4         int a=0, b=0, x=0,  y=35;
5         cout « "a:  " « a « "  b: " « b;
6         cout «  "  x:  • « x « "  y: " « y «   endl;
7         a = 9;
8         b = 7;
9         y = x = a+b;
10:        cout « "a:  " « a « "  b: " « b;
11:        cout « " x:    « x « "  y: " « y «   endl;
12:           return 0;
13:   }

              a: 0 b: 0 x: 0 y: 35
              a: 9 b: 7 x: 16 y: 16
```

|.VHU/:^!^ U liniji 4 smo deklarisali i inicijalizovali četiri promenljive. Njihove vrednosti se štampaju u linijama 5 i 6. U liniji 7 promenljivoj a se dodeljuje vrednost 9. U liniji 8 promenljivoj b se dodeljuje vrednost 7. U liniji 9 vrednosti promenljvih a i b se sabiraju i rezultat se dodeljuje promenljivoj x. Izraz (x=a+b) se izračuna kao vrednost (zbir vrednosti promenljivih a i b) i ta vrednost se dodeli promenljivoj y.

Operator!

Operator je simbol koji tera prevodilac da preduzme akciju. Operatori deluju nad operandima, a u programskom jeziku C + + svi operandi su izrazi. U programskom jeziku C+ + razlikujemo nekoliko različitih kategorija operatora. Dve od ovih kategorija su

- operator dodele
- aritmetički operatori.

Operator dodele

Operator dodele (=) prouzrokuje da operand na levoj strani operatora dodele zameni svoju vrednost vrednošću na desnoj strani operatora dodele. Izraz

```
x • a + b;
```

dodeljuje vrednost, koja je rezultat sabiranja a i b, operandu x.

Operand koji se može naći na levoj strani operatora dodele (a da izraz bude ispravan) se naziva *l-vrednost* (eng. *lvalue)*. To znači da se na desnoj strani nazivaju (pogodili ste) *d-vrednost* (eng. *rvalue)*.

Konstante su d-vrednosti i ne mogu biti 1-vrednosti. Tako možete pisati

```
x = 35        // ispravno
```

ali ne možete kao ispravno napisati

```
35 = x;       // greška, nije l-vrednost!
```

L-vrednost je operand, koji se može nad na levoj strani izraza. D-vrednost je operand koji može biti samo na desnoj strani izraza. Primetite da su sve 1-vrednosti istovremeno i d-vrednosti, ali da sve d-vrednosti nisu istovremeno i 1-vrednosti. Primer d-vrednosti, koja istovremeno nije i l-vrednost, je konstanta, doslovna i simbolička. Tako možete pisati x=5;, ali ne i 5=x;.

Aritmetički operatori

Postoji pet aritmetičkih operatora: sabiranje (+), oduzimanje (-), množenje (*), deljenje (/) i modul (%).

Oduzimanje od unsigned celobrojnih promenljivih može dovesti do iznenađujućih rezultata, ako je rezultat negativan broj. Takve rezultate ste već mogli da vidite u prethodnom Danu, kada ste upoznavali posledice prekoračenja kod promenljivih. Listing 4.2 pokazuje šta se dogada kada oduzmete veliki od malog unsigned broja.

Listing 4.2.: Demonstracija korišćenja operatora oduzimanja i prekoracenja

```
// Listing 4.2 - demonstrira oduzimanje
// prekoračenje
#include<iostream.h>

int main()
{
   unsigned int difference;
   unsigned int bigNumber = 100;
   unsigned int small Number » 50;
   difference = bigNumber - small Number;
   cout « "Razlika je: " « diffrence;
   difference = small Number - bigNumber;
   cout « "\nSada je razlika: " « diffrence «end1;
      return 0;
15: }
```

```
JWWWW> Razlika je: 50
vxrr::x <r Sada je razlika: 4294967246
```

I^IJI^ji^ Operator oduzimanja se koristi u liniji 10 i rezultat se štampa u liniji 11, onakav kakav bismo i očekivali. Operator oduzimanja se ponovo koristi u liniji 12, ali ovog puta se oduzima veliki od malog unsigned broja. Rezultat će biti negativan, ali pošto se izračunavanje obavlja (i štampa) kao unsigned broj, rezultat operacije je prekoračenje, kao što je opisano u prethodnom Danu. Ova tema se detaljno razmatra u Dodatku A, "Prioritet operatora".

Celobrojno deljenje i ostatak deljenja

Celobrojno se razlikuje od uobičajenog deljenja. Kada delite 21 sa 4, rezultat je realan broj (broj sa razlomkom). Celobrojni brojevi nemaju razlomak, tako da se preostali deo jednostavno odbacuje. Rezultat je tako 5. Da biste dobili ostatak deljenja, morate da nadete 21 mod 4 (21 % 4) i rezultat je 1. Operator modul Vam daje ostatak celobrojnog deljenja.

Nalaženje ostatka celobrojnog deljenja može da bude veoma korisno. Na primer, možete poželeti da štampate poruku posle svake desete akcije. Svaki broj za koji dobijate vrednost 0, kada tražite modul 10 od tog broja, predstavlja tačan sadržalac broja 10. Tako je 1%10 1, 2%10 2 i tako redom, sve do 10%10, što daje rezultat 0.1 11% 10 je opet 1 i ovaj obrazac se nastavlja sve do sledećeg sadržaoca broja 10, to jest broja 20. Ovu tehniku ćemo koristiti kada budemo objašnjavali petlje u Danu 7, "Još o upravljanju programom".

4**UPOZORENJE**| Mnogi neiskusni C+ + programeri nenamerno stave tačku-zarez posle naredbe i f:

```
if (SomeValue < 10);
SomeValue = 10;
```

Ono što je programer nameravao je da testira da li je vrednost promenljive

SomeValue manja od 10 i, ako jeste, da joj dodeli 10, učinivši 10 minimalnom vrednošću promenljive SomeVal ue. Ali ako izvršimo gornji kod, primetićemo da je posle njega vrednost promenljive SomeVal ue uvek 10! Zašto? Zato sto se if iskaz završava tačka-zarezom, odnosno neće biti uradeno ništa na osnovu uslova iz i f.

Setite se da namere ne znace nista prevodiocu. Gornji deo koda se moze jasnije napisati kao

```
if (SomeValue < 10)          // test
;                            // ništa nemoj da uradiš
SomeValue =10;               // dodeli
```

Uklanjanje tačke-zareza učiniće zadnju liniju delom if iskaza i učiniće da kod radi ono što se nameravalo.

Mešanje operatora dodele i aritmetičkih operatora

Nije neubičajeno da se želi sabiranje neke vrednosti sa promenljivom, a zatim dodela rezultata istoj promenljivoj. Ako imate promenljivu `myAge` i želite da joj povećate vrednost za dva, možete napisati

```
int myAge = 5;
int temp;
temp = myAge + 2;       // saberi 5 + 2 i stavi rezultat u temp
myAge = temp;           // vrati ga u myAge
```

Iako je izraz tačan, ovaj način je izuzetno nepraktičan i rasipan. U C + + možete staviti istu promenljivu na obe strane operatora dodele i tako možemo svesti gornji kod na

```
myAge = myAge + 2;
```

što je mnogo bolje. U algebri ovaj izraz ne bi imao značenje, ali C + + ga tumači kao "dodaj dva na vrednost promenljive `myAge` i dodeli rezultat promenljivoj `myAge`".

Još jednostavnije, mada možda malo teže za praćenje, možemo napisati

```
myAge += 2;
```

Operator dodele vrednosti i sabiranja (+=) sabirad-vrednost sa 1-vrednošću i zatim dodeljuje rezultat ponovo u l-vrednost. Operator se čita kao "plus-jednako". Ako promenljiva `myAge` ima vrednost 4 na početku, trebalo bi da ima vrednost **6** posle ovog iskaza.

Postoje i operatori dodela vrednosti i oduzimanja (-=), deljenja (/=), množenja (*=) i modula (%=).

Inkrement i dekrement

Najčešca vrednost koja se dodaje (ili oduzima) vrednosti promenljive i zatim joj se ponovo dodeljuje je 1. U C+ + povećavanje vrednosti za 1 se naziva inkrementiranje, a smanjivanje vrednosti za 1 se naziva dekrementiranje. Postoje dva posebna operatora koji obavljaju ove akcije.

Inkrement operator (++) povećava vrednost promenljive za 1, a dekrement operator (--) smanjuje vrednost promenljive za 1. Ako imate promenljivu C i želite daje inkrementirate, koristiićete sledeći iskaz:

```
C++; // povećaj vrednost C za 1
```

Ovaj iskaz je potpuno ekvivalentan sa opširnijim iskazom

```
C = C + 1;
```

koji se može zameniti sa manje opširnim izrazom

```
C += 1;
```

Prefiks i postfiks

I operator inkrementa (++) i operator dekrementa (--) mogu se javiti u dva oblika: prefiksnom i postfiksnom. Prefiksni oblik se piiše pre naziva promenljive `(++myAge);` postfiksni oblik se piše posle naziva promenljive `(myAge++).`

U jednostavnom iskazu nije važno koji će se oblik koristiti, ali u složenim iskazima, kada inkrementirate (ili dekrementirate) promenljivu, a zatim njenu vrednost dodeljujete drugoj promenljivoj, to je veoma važno. Prefiksni operator će biti izvršen pre dodeljivanja, dok će se postfiksni izvršiti posle.

Semantika prefiksa je sledeća: inkrementiraj vrednost i zatim je dohvati. Semantika postfiksa je razlidta: dohvati vrednost, pa inkrementiraj original.

Ovo može biti zbunjujuće na prvi pogled, ali ako je x promenljive tipa int, čija je vrednost 5 i ako napišete

```
int a = ++x;
```

saopštili ste prevodiocu da inkrementira x (postaje **6)** i da, zatim, dohvati njegovu vrednost i dodeli je promenljivoj a. Tako je sada vrednost promenljive a, **6,** kao i promenljive x.

Ako, posle toga napišete

```
int b = x++;
```

saopštili ste prevodiocu da dohvati vrednost promenljive x i dodeli je promenljivoj b, a zatim da inkrementira x. Tako je sada vrednost promenljive b, **6,** ali je vrednost promenljive x, **7.** U listingu **4.3** se prikazuju upotreba i posledice upotrebe oba oblika.

Listing 4.3.: Demonstracija upotrebe prefiksnih i postfiksnih operatora

```
1: // Listing 4.3 - demonstrira upotrebu
2: // prefiksnih i postfiksnih inkremenet i
3: // dekrement operatora
4:  linclude <iostream.h>
5:  int mainQ
6: {
7:      int myAge = 39;       // initialize two integers
8:      int yourAge = 39;
9:   cout « "Imam: " « myAge « " godina. \n";
10:  cout « "Ti imaš: " « yourAge « " godina. \n";
11: myAge++;          //postfiks  inkrement
12: ++yourAge;        //prefiks  inkrement
13:  cout « "Prođe jedna godina... \n";
14:  cout « "Imam: " « myAge « " godina. \n";
15:  cout « "Ti imaš: " « yourAge « " godina. \nM;
16:  cout « "Prode još jedna godinaĐn";
17:  cout « "Imam: " « myAge++ « " godina. \n";
18:  cout « "Ti imaš: " « ++yourAge « " godina. \n";
19:  cout « "Hajde da to ponovo odštampamo\n";
20:  cout « "Imam: " « myAge « " godina. \nM;
21:  cout « "Ti imaš: " « yourAge « M godina. \nM;
22:      return 0;
23:}
```

```
JMHfiYJbfc  Imam      39 godina
            Ti ima§   39 godina
            Prode jedna godina
            Imam      40 godina
            Ti imaš   40 godina
            Prode joS jedna godina
            Imam      40 godina
            Ti imaš   41 godina
            Hajde da to ponovo odštampamo
            Imam      41 godina
            Ti imaš   41 godina
```

U linijama 7 i 8 smo deklarisali dve celobrojne promenljive i svaku smo inicijalizovali sa 39. Njihova vrednost je odštampana u linijama 9 i 10.

U liniji 11 `myAge` se inkrementira, korišćenjem postfiksnog inkrement operatora, a u liniji 12 `yourAge` se inkrementira korišćenjem prefiksnog inkrement operatora. Rezultati se štampaju u linijama 14 i 15 i oba su jednaka (40).

U liniji 17 `myAge` se inkrementira kao deo iskaza štampanja, korišćenjem postfiksnog inkrement operatora. Pošto je reč o postfiksu, inkrementiranje se obavlja posle štampanja, tako da se opet štampa vrednost `40`. Nasuprot tome, u liniji 18 `yourAge` se inkrementira, korišćenjem prefiksnog inkrement operatora. Pošto se inkrementiranje obavlja pre štampanja prikazana vrednost je 41.

Na kraju, u linijama 20 i 21, vrednosti promenljivih su opet štampane. Poštoje inkrementiranje već obavljeno, vrednost myAge je sada **41,** baš kao i vrednost promenljive yourAge.

Prioritet operatora

U složenom iskazu

```
x = 5 + 3 * 8;
```

šta se prvo obavlja, sabiranje, ili množenje? Ukoliko se prvo obavi sabiranje, rezultat je 8*8, odnosno 64. Ako se prvo obavi množenje, odgovor je 5+24, odnosno 29. Množenje ima viši proritet od sabiranja i tako je vrednost izraza 29.

Svaki operator ima svoj prioritet i kompletna lista prioriteta operatora je data u Dodatku A, "Prioritet operatora".

Kada dva aritmetička operatora imaju isti prioritet, oni se obrađuju s leva na desno. Tako se u donjem izrazu

```
x = 5 + 3 + 8 * 9 +6 *4;
```

prvo izračunaju množenja, sleva nadesno, odnosno, 8*9=72 i 6*4=24. Sada je izraz, u osnovi,

```
x = 5 + 3 + 72 + 24;
```

Potom obavljamo sabiranje, sleva nadesno: 5+3 = 8; 8+72=80;80+24=104.

Budite oprezni. Neki operatori, poput operatora dodele, izvršavaju se sa desna na levo! U svakom slučaju, šta učiniti, ako prioritet operatora ne odgovara Vašim potrebama? Pogledajte sledeći izraz

```
Total Seconds = NumMinutesToThink + NumMinutesToType * 60
```

U ovom izrazu, ne želimo da množimo NumMinutesToType promenljivu sa 60 i zatim da rezultat saberemo sa NumMinutesToThink. Želimo da saberemo dve promenljive, da bismo dobili ukupan broj minuta, i zatim da njega pomnožimo sa 60, da bismo dobili ukupan broj sekundi.

U ovom slučaju, da biste promenili prioritet operatora, koristite zagrade. Sve unutar zagrada se obraduje sa većim prioritetom, nego ma koji aritmetički operator. Tako će

```
TotalSeconds = (NumMinutesToThink + NumMinutesToType) * 60
```

biti ono što Vam treba.

Ugnježdavanje zagrada

U složenim izrazima ćete, možda, morati da se koristite ugnježdavanjem zagrada. Na primer, možda ćete imati potrebu ukupan broj sekundi i da izračunate ukupan broj ljudi, pre nego što pomnožite broj sekundi i broj ljudi:

```
TotalPersonSeconds = ((NumMinutesToThink + NumMinutesToType) * 60)*
  (PeopleInTheOffice + PeopleOnVacation))
```

U ovom složenom izrazu se prvo NumMinutesToThink sabira sa NumMinutesToType, zato što se nalaze u unutrašnjoj zagradi. Zatim se njihov zbir množi sa 60. Potom se PeopleInTheOffice sabira sa PeopleOnVacation. Na kraju se ukupan broj ljudi množi sa ukupnim brojem sekundi.

Ovaj izraz je lak za računare, ali je ljudima veoma teško da ga razumeju, ili modifikuju. Isti izraz možemo drugačije predstaviti, koristeći neke privremene celobrojne promenljive.

```
TotalMinutes = NumMinutesToThink + NumMinutesToType);
TotalSeconds = TotalMinutes * 60;
TotalPeople = PeopleInTheOffice + PeopleOnVacation;
TotalPersonSeconds = TotalPeople * TotalSeconds;
```

Treba vise vremena da bi se napisao ovaj primer i pri torn se koristi vise privremenih, promenljivih nego u prethodnom primeru, ali je zato mnogo lakši za razumevanje. Ako još dodate komentar ispred prvog izraza, koji objašnjava šta kod radi i zamenite 60 simboličkom konstantom, imaćete kod koji je lak za razumevanje i održavanje.

^J; **PAzm** *Ll?* Zapamtite da izrazi imaju vrednost.

Koristite prefiks operator *(^promenljiva)* da inkrementirate, ili dekrementirate promenljivu, pre nego što je upotrebite u izrazu.

Koristite postfiks operator *(promenljiva++)* da inkrementirate, ili dekrementirate promenljivu, pošto je upotrebite u izrazu.

Koristite zagrade, da biste promenili prioritet operatora.

Nemojte previše ugnježdavati zagrade, jer izraz postaje težak za razumevanje i odrzavanje.

Priroda istinitosti

U programskom jeziku C+ 4- nula predstavlja neistinitu vrednost *{eng. false),* dok se sve ostale vrednosti smatraju kao istinite (eng. *true),* mada je istinita vrednost, obično, predstavljena sa 1. Ako je izraz neistinit, on je jednak nuli, a ako je izraz jednak nuli, onda je neistinit. Ako je izraz istinit, znate jeste da nije jednak nuli i da je svaki izraz koji nije jednak nuli, istinit.

Relacioni operatori

Relacioni operatori se koriste u odredivanju da li su dva broja jednaka, ili je jedan veći ili manji od drugog. Svaki relacioni iskaz se iskazuje ili kao 1 (TRUE), ili kao 0 (FALSE). Lista relacionih operatora je data u tabeli 4.1.

Ako celobrojna promenljiva MyAge ima vrednost 39 i ako celobrojna promenljiva YourAge ima vrednost 40, možete odrediti da li ove dve promenljive imaju istu vrednost, korišćenjem operatora jednakosti:

```
myAge == yourAge; // da li je vrednost promenljive myAge ista kao promenljive yourAge?
```

Ovaj izraz ima vrednost 0, odnosno `false`, pošto promenljive nisu jednake. Izraz

```
myAge > yourAge; // da li je vrednost promenljive myAge veća od promenljive yourAge?
```

ima vrednost 0, ili `false`.

vUPOZORENJI| Mnogi neiskusni C-I- + programeri mešaju operator dodele (=) sa operatorom jednakosti (==). To može dovesti do grešaka u programima koji se teško nalaze.

Postoji šest relacionih operatora: jednako (==), manje (<), veće (>), manje ili jednako (<=), veće ili jednako (>=) i nije jednako (!=). Tabela 4.1 prikazuje svaki relacioni operator, njegovu upotrebu i primer koda.

Tabela 4.1.: Relacioni operatori

Naziv	Operator	Primer	Vrednost
jednako	==	100 == 50;	false
		50 == 50;	true
nije jednako	!=	100 != 50;	true
		50 != 50;	false
veće	>	100 > 50;	true
		50 > 50;	false
veće ili jednako	>=	100 >= 50;	true
		50 >= 50;	true
manje	<	100 < 50;	false
		50 < 50;	false
manje ili jednako	<=	100 <= 50;	false
		50 <= 50;	true

PAZITI *Ll,* Zapamtite da relacioni operatori vraćaju vrednost 1 (true) ili 0 (fal se).

Nemojte mešati operator dodele (=) sa relacionim operatorom jednakosti (==). Ovo je jedna od najčešćih programerskig grešaka u programskom jeziku C++.

Iskaz i f

Vaš program se, obično, izvršava liniju, po liniju, kako se one pojavljuju u Vašem izvornom kodu. Iskaz if Vam omogućava da ispitate uslov (poput: da li su dve promenljive jednake) i da "skočite" na različite delove programa, u zavisnosti od rezultata.

Najjednostavniji oblik if iskaza je

```
if   (izraz)
     iskaz;
```

Izraz u zagradi može biti ma koji, ali, obično, sadrži jedan od relacionih izraza. Ako izraz ima vrednost 0, smatra se da je neistinit i iskaz se preskače. Ako izraz ima ma koju vrednost osim nule, smatra se da je istinit i iskaz se izvršava. Obratite pažnju na sleded primer

```
if (bigNumber > smallNumber) bigNumber = smallNumber;
```

Ovaj kod poredi `bigNumber` i `smallNumber`. Ako je `bigNumber` ved, dodeljuje mu se vrednost promenljive `smal 1 Number.`

Pošto je blok iskaza unutar vitičastih zagrada ekvivalentan jednom iskazu, sleded oblik grananja može biti zaista veliki:

```
if   (izraz)
{
     iskazi;
     iskaz2;
     iskaz3\
}
```

Evo jednostavnog primera upotrebe:

```
if (bigNumber > smallNumber)
{
     bigNumber = smallNumber;
     cout « "Veliki broj: " « bigNumber « "Đn";
     cout « "Mali broj: " « smallNumber « "On";
}
```

Ovoga puta, ako je `bigNumber` ved, ne samo da mu se dodeljuje vrednost promenljive `smal 1 Number`, već se prikazuje i poruka o tome. U listingu 4.4 je dat detaljniji primer grananja, zasnovanog na relacionim operatorima.

Listing 4.4.: Pemonstrgcijq grananja, zasnovana na relacionim operatorima

```
1:  //Listing 4.4 - demonstracija iskaza if
2:  //korfšćenog sa relacionim operatorima
3:  linclude <iostream.h>
4:  int main()
5:  {
6:       int RedSoxScore, YankeesScore;
7:       cout « "Unesite rezultat za Red Sox: ";
8:       cin » RedSoxScore;
9:
10       cout « "\nUnesite rezultat za Yankees: ";
11       cin » YankeesScore;
12
13       cout « "\n";
14
15       if (RedSoxScore > YankeesScore)
16            cout « "Napred Sox! \n";
17
18       if (RedSoxScore < YankeesScore)
19       {
20            cout « "Napred Yankees! \n";
21            cout « "Sreća za New York! \n";
22       }
23
24       if (RedSoxScore     YankeesScore)
25       {
26            cout « "Nerešeno? To je nemoguće! \n";
27            cout « "Oaj mi pravi rezultat za Yankees:
28            cin » YankeesScore;
29
30            if (RedSoxScore > YankeesScore)
31                 cout « "Znao sam! Napred Sox!";
32
33            if (YankeesScore > RedSoxScore)
34                 cout « "Znao sam! Napred Yankees!";
35
36            if (YankeesScore == RedSoxScore)
37                 cout « "Stvarno je bilo nerešeno!"
38       }
39
40       cout « "\nHvala Ti što si mi rekao! \n";
41        return 0;
42
```

```
Unesite rezultat za Red Sox: 10

Unesite rezultat za Yankees: 10

Nerešeno? To je nemoguće!
Daj mi pravi rezultat za Yankees: 8
Znao sam! Napred Sox!
Hvala Ti što si mi rekao!
```

pMj^fll^ Ovaj program traži od korisnika da unesu rezultate dva bezbol tima, koji se čuvaju u celobrojnim promenljivama. Promenljive se porede, korišćenjem iskaza `i` f, u linijama 15, 18 i 24.

Ako je jedan rezultat veći od drugog, prikazuje se poruka sa informacijom o tome. Ako su rezultati jednaki, ulazi se u blok koda, koji počinje od linije 24 i završava se u liniji 38. Traži se ponovni unos drugog rezultata i ponovo se vrši poredenje.

Primetićete da ako je početni rezultat Yankeeja viši od rezultata Red Soxa, iskaz if u liniji 15 će imati vrednost FALSE i linija broj 16 se neće izvršiti. Ispitivanje u liniji 18 će biti tačno i izvršiće se iskazi u linijama 20 i 21. Zatim će se ispitati if iskaz u liniji 24 i on će biti netačan (ako je iskaz if u liniji 18 bio tačan). Tako će program preskodti čitav blok iskaza, nastavljajući tek od linije 39.

Stilovi uvlačenja

Listing 4.3 prikazuje jedan stil uvlačenja iskaza if. Ništa neće pre dovesti do "religioznog rata", nego pitati grupu programera koji je najbolji način postavljanja zagrada. Iako postoji čitav niz varijacija, razlikujemo tri osnovna stila:

- postavljanje početne vitičaste zagrade posle uslova i pozicioniranje zatvarajuće vitičaste zagrade ispod if radi zatvaranja bloka iskaza

```
if (izraz){
      iskazi
}
```

- pozicioniranje vitičastih zagrada ispod if i uvlačenje iskaza unutar bloka

```
if (izraz)
{
      iskazi
}
```

- uvlačenje i zagrada i iskaza

```
if (izraz)
        {
        iskazi
        }
```

U ovoj knjizi se koristi drugo rešenje, pošto smatram da je za razumevanje mesta gde blok iskaza počinje i završava lakše ako se zagrade nalaze jedna ispod druge. Nije značajno koji ćete stil koristiti, sve dok ste dosledni u njegovoj primeni.

else

Cesto ćete imati potrebu da izvršite jednu grupu naredbi, ako je Vaš uslov tačan, a drugu, ako je netačan. U listingu 4.3 želite da odštampate poruku (Napred Sox!), ako je prvo ispitivanje tačno (RedSoxSkore>Yankees), a drugu poruku, ako nije tačno (Napred Yanks!).

Do sada smo prvo ispitali jedan uslov, pa, zatim, drugi, čime se obavlja posao, ali dolazi do prenatrpanosti. Korišćenje službene reči else nas dovodi do daleko dtljivijeg koda:

```
if (izraz)
iskazi;
el se
iskaz2;
```

Listing 4.5 prikazuje korišćenje službene red else.

Listing 4.5.: Demonstracija upotrebesluibene reel else

```
1:   // Listing 4.5 - demonstrira iskaz if
2:   // sa else rečenicom
3:   linclude <iostream.h>
4:    int main()
5:   {
6:       int firstNumber, secondNumber;
7:       cout « "Unesite veliki broj, molim: ";
8:       cin » firstNumber;
9:       cout « "\nUnesite manji broj, molim: ";
10:      cin » secondNumber;
11:      if (firstNumber > secondNumber)
12:            cout « "\nHvala! \n";
13:      else
14:            cout « "\nUh! Drugi je veći!";
15:
16:        return 0;
17: }
```

```
ШЕЈПШј*  Unesite veliki broj, molim: 10
         Unesite manji broj, molim: 12
         Uh! Drugi je veći!
```

IDMIMfr* Iskaz if u liniji 11 se izvršava. Ako je tačan, izvršava se iskaz u liniji 12; ako nije tačan, izvršava se iskaz u liniji 14. Ako bi se uklonila reč else u liniji 13, iskaz 14 bi bio izvršen uvek, bez obzira da li je ili nije iskaz if tačan. Zapamtite da se oba iskaza if završava linijom 12. Ako se else tamo ne nalazi, linija 14 će biti samo sledeća linija programa.

Zapamtite da se svaki ili oba iskaza mogu zameniti blokom koda u vitičastim zagradama.

Iskaz if

Sintaksa za iskaz if je sledeća:

Oblik 1

```
if   (izraz)
       iskaz;
sledeći    iskaz;
```

Ako je izraz tačan, iskaz se izršava i program nastavlja sa sledećim iskazom. Ako izraz nije tačan, iskaz se preskače i program "skače" na sledeci iskaz. Zapamtite da iskaz može biti samo jedan iskaz koji se završava tačka-zarezom, ili blok unutar vitičastih zagrada.

Oblik 2

```
if   (izraz)
      iskazi;
else
      iskaz2;
sledeći    iskaz;
```

Ako je izraz tačan, iskazi se izvršava; u suprotnom, izvršava se iskaz2. Posle toga program nastavlja sa sledećim iskazom.

Primer **1**

```
Primer
if (SomeValue < 10)
     cout « "SomeValue je manje od 10");
else
     cout « "SomeValue nije manje od 10!")
cout « "Gotovo." « endl
```

Složeni if iskazi

Vredno je napomene da se unutar if iskaza, kao i kod dela else iskaza, može koristiti ma koji iskaz, čak i drugi if iskaz. Na taj način se može doći do složenih if iskaza u sledećem obliku

```
if   (izrazi)
{
     if   (izraz2)
           iskazi;
     else
     {
               if   (izraz3)
                     iskaz2;
               else
                     iskaz3;
     }
}
else
     iskaz4;
```

Ovaj složeni if iskaz govori: 'Ako su izrazi i izraz2 tačni, izvršite iskazl. Ako je izrazl tačan, ali i zraz2 nije, tada, ako je i zraz3 tačan, izvršite i skaz3.1 konačno, ako i zrazl nije tačan, lzvršite i skaz4." Kao što možete uočiti, složeni if iskazi mogu biti zbunjujući.

Listing 4.6 daje primer takvog složenog if iskaza.

Listing 4.6.: Složen, ugnjeiden if iskaz

```
//Listing 4.5 - složen, ugnježden
//if  iskaz
linclude <iostream.h>
int  main()
{
        //Tražite dva broja
        //Dodelite brojeve promenljivim bigNumber i littleNumber,
        //Ako je bigNumber veći od littleNumber,
        //proverite da li su međusobno deljivi
        //Ako jesu, proverite da li je u pitanju isti broj

        int firstNumber, secondNumber;
        cout « "Unesite dva broja. \nPrvi: ";
        cin » firstNumber;
        cout « "\nDrugi: ";
        cin » secondNumber;
        cout « "\n\n";

        //Ako jesu, proverite da li je u pitanju isti broj
        {
            if ( (firstNumber % secondNumber) == 0) // medusobno deljivi?
            {
            if (firstNumber == secondNumber)
                 cout « "Isti su! \n";
            else
                 cout « "Medusobno su deljivi! \n";
            }
            else
                 cout « "Nisu medusobno deljivi! \n";

        else
         cout « "Hej! Drugi broj je veći! \n";
          return 0;


        Unesite dva broja.
        Prvi: 10

        Drugi: 2

        Medusobno su deljivi!
```

*IIIII3^* :t•nrm: Unosimo dva broja, jedan, po jedan, i zatim ih uporedujemo. Prvi iskaz if, u liniji 19, pro vera va da li je prvi broj veći od drugog, ili da li je jednak drugom. Ako nije prelazi se na else u liniji 31.

Ako je prvo if tačno, izvršava se blok koda, koji počinje u liniji 20 i ispituje se drugi if iskaz, u liniji 21. Drugi if iskaz proverava da li postoji ostatak pri deljenju prvog broja sa drugim. Ako nema ostatka, brojevi su jednaki, ili je prvi broj deljiv drugim.

Iskaz i f, u liniji 23, provera da li su brojevi jednaki i prikazuje odgovarajuću poruku, zavisno od rezultata.

Ukoliko je iskaz i f, u liniji 21, netačan, izvršava se iskaz iza el se u liniji 31.

Korišćenje zagrada u ugnježdavanju i f iskaza

Iako je sasvim ispravno da se ne koriste zagrade kada if iskaz ima samo jedan iskaz i prilikom ugnježdavanja if iskaza, poput

```
if (x > y)              // ako je x veće od y
    if (x < z)          //i ako je x manje od z
        x = y;          //onda izjednači vrednost x i z
```

to, prilikom pisanja složenih ugnježdenih iskaza, može dovesti do velike konfuzije. Zapamtite: korišćenje praznih mesta i uvlačenje predstavljaju olakšicu za programera, a ništa ne znače za prevodilac. Veoma je lako pogrešiti i nenamerno dodeliti else iskaz pogrešnom if iskazu. Listing 4.7 ilustruje ovaj problem.

Listing 4.7.; Demonstrqcija lašto zagrode pomažu da se utvrdi koji se el se slaže sa kojim i f.

```
//Listing 4.7 - demonstrira zašto su
//zagrade važne u ugnježdenim if iskazima
finclude <iostream.h>
int main()
{
  int x;
  cout « "Unesite broj manji od 10, ili veći od 100:
  cin » x;
  cout « "\n"

   if (x > 10)
      if (x > 100)
           cout « "Veći od 100. Hvala! \n";
   else          //pogrešno else!
     cout « "Manji od 10. Hvala! \n";

      return 0;
```

```
Unesite broj manji od 10, ili veći od 100: 20

Manji od 10. Hvala!
```

Programer je nameravao da zatraži unos broja izmedu 10 i 100, da proved da li je uneta ispravna vrednost i zatim da odštampa zahvalu.

Ako je izraz u i f iskazu, u liniji 11, tačan, izvršava se sledeći iskaz (linija 12). U ovom slučaju, linija 12 se izvršava kada je uneti broj veći od 10. Linija 12, takode, sadrži iskaz i f. Izraz u i f iskazu je tačan, ako je uneti broj ved od 100. Ako je uneti broj veći od 100 izvršava se linija 13.

Ukoliko je uneti broj manji, ili jednak 10, iskaz if u liniji 10 je netačan. Program "skače" na prvu liniju, posle iskaza i f, u ovom slučaju na liniju 16. Ukoliko unesete broj manji od 10, na izlazu ćete dobiti sledeće:

```
Unesite broj manji od 10, ili veći od 100: 9
```

Iskaz el se u liniji 14 je trebalo da pripadne i f iskazu u liniji 11. Na žalost, u stvarnosti je else iskaz prikačen na if iskaz iz linije 12 i program ima jedva primetnu grešku, pošto se prevodilac neće buniti. Ovo je potpuno ispravan C + + program, ali on ne radi ono za šta je namenjen. Što je još gore, najveci broj puta prilikom testiranja programa, on će, najčešće, raditi potpuno ispravno. Sve dok je uneti broj veći od 100, program će raditi potpuno ispravno.

U listingu 4.8 problem se rešava korišćenjem neophodnih vitičastih zagrada.

Listing 4.8.: **Demonstracija pravilne upotrebe zagrada sa i f iskazom**

```
// Listing 4.8 demonstrira pravilnu upotrebu zagrada
// sa if iskazom
 #include <iostream.h>
 int main()
 {
   int x;
   cout « "Unesite broj manji od 10, ili veći od 100: ";
   cin » x;
   cout « "\n"

   if (x > 10)
   {
      if (x > 100)
           cout « "Veći od 100. Hvala! \n";
   }
   else                          //pogrešno else!
       cout « "Manji od 10. Hvala! \n";
        return 0;
```

```
аМГУ^J^  Unesite broj manji od 10, ili veći od 100: 20
```

ti!Mjl№|^» Vitičaste zagrade u linijama 12 i 15 cine sve između njih jednim iskazom i sada je else u liniji 16 dodeljeno iskazu if u liniji 11, kao što je i nameravano.

Ako korisnik unese 20, tada će iskaz i f u liniji 11 biti tačan dok će iskaz i f u liniji 13 biti netačan, tako da se ništa neće odštampati. Bilo bi bolje kada bi programer postavio još jedno el se posle linije 14, tako da bi se greška "uhvatila" i odgovarajuća poruka odštampala.

<*ф№ III III 1jp* **Programi prikazani u ovoj knjizi su pisani da bi se ilustrovale odredene feme koje se prikazuju. Oni su namerno učinjeni jednostavnim; nema načina da se program udni otpornim na greške korisnika. U kodu profesionalnog kvaliteta mora biti predvidena i odgovarajuće obradena svaka greška korisnika.**

Logički operatori

Često želite da postavite vise od jednog relacionog pitanja istovremeno: "Da li je tačno da je x veće od y i da li je tačno da je y veće od z?" Program mora da odredi da su oba uslova ispunjena, ili da je neki drugi uslov ispunjen, da bi preduzeo akciju.

Zamislite složen alarmni sistem koji ima sledeću logiku rada: "Ako se oglasio alarm sa yrata I ako je posle 18h I ako NIJE praznik, ILI ako je vikend, tada pozvati policiju." Tri navedena logička operatora programskog jezika C + + se koriste, da bi se omogućilo postavljanje takvog složenog uslova. Lista logičkih operatora je data u tabeli 4.2.

Tabela 4.2.: Logički operatori

Operator	Simbol	Primer
I (AND)	&&	*izrazl* **&&** *izraz2*
ILI (OR)	!!	*izrazl* 'J *izraz2*
NEGACIJA (NOT)	!	i izraz

Logičko I (AND)

Logički I izraz povezuje dva izraza i, ako su oba tačna, izraz sa logičkim I je, takode tačan. Ako je tačno da ste gladni I ako je tačno da imate novca, tada je tačno da možete sebi da kupite ručak. Na taj način će

```
if ((x == 5) && (y ==5))
```

biti tačno, ako je tačno da su i x i y jednaki `5`, a biće netačno ako je, bar jedan od niih različit od `5.`

Primetićete da je logičko I predstavljeno sa dva && simbola. Samo jedan & simbol je potpuno različit operator, o kome će biti red u Danu 21, "Šta je sledeće".

Logičko ILI (OR)

Logički ILI izraz se sastoji od dva izraza, povezana znakom j j. Ako je ma koji od njih tacan i ceo izraz je tačan. Ako imate novca, ILI imate kreditnu kartu, tada možete platiti račun. Nije potrebno da imate i novae i kreditnu kartu; dovoljno je da imate samo jedno od njih, mada je dobro ako posedujete i jedno i drugo. Na taj način će

```
if ((x == 5) 'j (y ==5))
```

biti tačno, ako su ili x, ili y jednaki `5,` ili ako su i x i y jednaki `5.`

Primetićete da je logično ILI predstavljeno sa dva |j simbola. Samo jedan j simbol je potpuno različit operator, o kome će biti reči u Danu 21, "Šta je sledeće".

Logička negacija (NOT)

Logički izraz sa logičkom negacijom je tačan, ako je izraz koji se negira netačan. Opet, ako je izraz koji se negira tačan, ceo izraz će biti netačan! Na taj način će

```
if (!(x == 5))
```

biti tačno, samo ako x nije jednako `5.` Drugačije ovo možemo zapisati kao

```
if (x != 5)
```

Prioritet relacija

Relacioni i logički operatori, koji cine C + + izraze, obavezno vraćaju vrednost: 1 (tačno - TRUE), ili 0 (netačno - FALSE). I oni imaju svoj prioritet (pogledajte Dodatak A) koji određuje koja će se relacija prva ispitati. Ova činjenica je veoma značajna pri traženju vrednosti ovakvih iskaza

```
if (x > 5 && y > 5 !i z > 5)
```

Moguće je da je programer želeo da ovaj izraz bude tačan, ako su i x i y veći od 5, ili ako je z veće od 5. S druge strane, programer je možda, želeo, da izraz bude tačan samo kada je x veće od 5 i ako je tačno da je y veće od 5, ili da je z veće od 5.

Ako x ima vrednost 3, a y i z imaju vrednost 10, prva interpretacija izraza će biti tačna (z je veće od 5, tako da je ceo izraz tačan, bez obzira na vrednost x i y), ali druga interpretacija će biti netačna (nije tačno da je x veće od 5, te je ceo izraz netačan, bez obzira što je z veće od 5).

Iako će prioritet odrediti koja će se relacija prva ispitivati, zagrade mogu da promene prioritet i da učine sve jasnijim

```
if ((x > 5) && (y > 5 ! ! z > 5))
```

Ako koristi napred date vrednosti, ovaj iskaz je netačan. Pošto nije tačno da je x veće od 5, leva strana I iskaza je netačna i, na osnovu toga, ceo iskaz je netačan. Zapamtite da I iskaz zahteva da njegove obe strane budu tačne.

yMAPOKUMAy Korišćenje dodatnih zagrada, zbog jasnijeg odredivanja šta želite da grupišete, |e uvek dobra ideja. Zapamtite: cilj je da se pišu programi koji rade i koji su laki za čitanje i razumevanje.

Još o istinitosti

U programskom jeziku C++ nula predstavlja netačnu vrednost, dok je ma koja druga vrednost tačna. Budući da izraz uvek ima vrednost, mnogi C++ programed koriste ovu osobinu u svojim if naredbama. Naredba poput

```
if (x)                          // ako je x tačno (različito od nule)
x = 0;
```

čita se kao "Ako je x različito od nule, postavi ga na nulu." Jasnije ovo može biti zapisano kao

```
if (x != 0)                     // ako je x različito od nule
x = 0;
```

Obe naredbe su ispravne, ali je poslednja jasnija. Dobra programerska praksa je da se za poslednji metod koristi za prvi metod koristi za prava logička ispitivanja, umesto za ispitivanje da li je vrednost promenljive različita od nule.

Sledeća dva iskaza su, takode, ekvivalentna:

```
if (!x)                 // ako je x netačno (nula)
if (x==0)               // ako je x nula
```

Drugi iskaz je jasniji za razumevanje.

<|. WUTi Koristite zagrade oko logičkih ispitivanja, da ih učinite jasnijim i da eksplicitno date prioritet.

Koristite zagrade u ugnježdenim if naredbama, da jasno naznačite kome pripada koja else naredba i da izbegnete greške.

Nemojte koristiti if(x) kao sinonim za if(x!=0); poslednji iskaz je jasniji

Nemojte koristiti if(!x) kao sinonim za if(x==0); poslednji iskaz je jasniji

yMAPOMENAy Uobičajeno je da se definiše sopsrveni prebrojivi Bulov (logički) tip sa enum Bool {FALSE, TRUE};. Ovo služi da postavi FALSE na 0, a TRUE na 1.

Uslovni (trostruki) operator

Uslovni operator (?:) je jedini trostruki operator u programskom jeziku C++; to je jedini operator sa tri člana.

Uslovni operator koristi tri izraza i vraća vrednost:

(izrazl) ? (izraz2): (izraz3)

Ova linija se tumači kao: "Ako je izrazl tačan, vrati vrednost od izraz2; inače, vrati vrednost od izraz3." Obično se ova vrednost dodeljuje promenljivoj.

U listingu 4.9 je prikazana upotreba uslovnog operatora.

Listing 4.9.: Demonstracija primene uslovnog operatora

```
1   // Listing 4.9 demonstrira primenu uslovnog operatora
2   //
3   #include <iostream.h>
4   int roain()
5   {
6      int x, y, z;
7      cout « "Unesite dva broja. \n";
8      cout «  "Prvi:
9      cin » x;
10     cout « "\nDrugi:  ";
11     cin » y;
12     cout « "\n";
13
14     if (x > y)
15       z = x;
16     else
17       z = y;
18
19     cout « "z:  " « z;
20     cout « "\n";
21
22     z =  (x > y) ? x : y;
23
24     cout « "z:  " « z;
25     cout « "\n";
26        return 0;
27
```

```
jWWjiifr  Unesite dva broja.
'          Prvi: 5

           Drugi: 8

           z: 8
           z: 8
```

T'HUI^ Kreiraju se tri celobrojne promenljive: x, y i z. Prve dve imaju vrednost koja je dobijena od korisnika. Naredba if, u liniji 14, ispituje koja promenljiva ima veću vrednost i dodeljuje je promenljivoj z. Ova vrednost se štampa u liniji 19.

Uslovni operator u liniji 22 vrši isto ispitivanje i dodeljuje veću vrednost promenljivoj z. To se tumači kao: "Ako je x veće od y, vrati vrednost od x; inače vrati vrednost od y." Vrednost koju vraća uslovni operator se dodeljuje promenljivoj z. Vrednost se štampa u liniji 24. Kao što možete videti, uslovna naredba je kraći ekvivalent od if...else naredbe.

Rezime

Ovo poglavlje je "pokrilo" dosta tema. Naučili ste šta su u programskom jeziku C + + **iskaz i izraz, šta** su **to** C + + **operatori i kako** C+ + if naredba radi.

Videli ste da blok naredbi unutar vitičastih zagrada može da se koristi gde se koristi samo jedna naredba.

Naučili ste da svaki izraz ima svoju vrednost i da ta vrednost može biti ispitana u i f **naredbi,** ili **korišćenjem uslovnih operatora.Takode ste** videli kako da **povežete** vise izraza, korišćenjem logičkih operatora, kako da poredite vrednosti, korišćenjem relacionih operatora i kako da dodeljujete vrednosti korišćenjem operatora dodele.

Istražili ste prioritet operatora i videli kako se zagrade mogu iskoristiti da promene redosled primene operatora i da tako učine redosled eksplicitnijim i, na taj način, lakšim za korišćenje.

Pitanja i odgovori

P Zašto koristiti nepotrebne zagrade kada prioritet operatora određuje redosled primene operatora?

O Iako je tačno da će prevodilac poznavati prioritet operatora i da programer može da koristi prioritet operatora, korišćenje zagrada omogućava veću čitljivost koda i njegovo lakše održavanje.

P Ako relacioni operatori uvek vraćaju 1, **ili** 0, **zašto se za druge vrednosti smatra da su tačne?**

O Relacioni operatori vraćaju 1, ili 0, ali izrazi vraćaju vrednosti koje se mogu ispitati u i f naredbi. Na primer,

```
if    ((x=a+b)==35)
```

Ovo je savršeno ispravan iskaz u programskom jeziku C+ + . Vrednost izraza se izračunava, čak i ako zbir a i b nije jednak 35. Takode ćete primetiti da je x dodeljena vrednost zbira a i b u svakom slučaju.

P Koje efekte tabovi, prazna mesta i novi redovi imaju na program?

O Tabovi, prazna mesta i novi redovi nemaju nikakav efekat na program, ali njihova pažljiva upotreba čini program lakšim za praćenje.

P Da li negativni brojevi imaju vrednost tačno ili netačno?

0 Svi brojevi koji nisu jednaki nuli, pozitivni i negativni, imaju vrednost tačno.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje predenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što pređete na sledeće poglavlje.

Kviz

1. Sta je to izraz?
2. Da li je x=5+7 izraz? Koja mu je vrednost?
3. Koja je vrednost od 201 / 4?
4. Koja je vrednost od 201 % 4?
5. Ako su myAge, a i b celobrojne int promenljive, koje su im vrednosti posle:

```
myAge=39;
a = myAge++;
b=++myAge;
```

6. Kolikoje 8+2*3?
7. Koja je razlika između x=3 i x==3?
8. Da li će sledeće vrednosti biti tačne (TRUE) ili netačne (FALSE)?

   a. 0

   b. 1

   c. -1

   d. x=0

   e. x==0 // Pretpostavite da x ima vrednost 0

Vežbe

1. Napišite jednu i f naredbu, koja ispituje dve celobrojne promenljive i menja vrednost veće vrednošću manje promenljive, koristed samo jedno el se.

   Proučite sledeći program. Zamislite da unoste tri broja i napišite koji izlaz očekujete.

```
1: linclude <iostream.h>
2: int main()
3: {
4: int a, b, c;
```

```
5:  cout « "Unesite tri  broja,  molim\n";
6:  cout « "a:  ";
7:  cin » a;
8:  cout « "\nb:  ";
9:  cin » b;
10:  cout « M\nc:  ";
11:  cin » c;
12:
13:  if (c = (a-b))
14:  scout « "a:  H;
15:  cout « a;
16:  cout « "minus  b:  ";
17:  cout « b;
18:  cout « "jednako je  c:  ";
19:  cout « c « endl ;}
20:  else
21:  cout « "a-b nije jednako c:  " « endl;
22:  return 0;
23:  }
```

3. Unesite u računar program iz druge vežbe; prevedite ga, povežite i izvršite. Unesite vrednosti **20,10** i **50.** Da li ste dobili izlaz koji ste očekivali? Zašto niste?

4. Proučite ovaj program i predvidite njegov izlaz:

```
1:  linclude <iostream.h>
2:   int  main()
3:  {
4:  int a = 1, b = 1, c;
5:  if (c = (a-b))
6:        cout « "Vrednost c je:  " « c;
7:  return 0;
8:  }
```

5. Unesite, prevedite, povežite i izvršite program iz četvrte vežbe. Šta je njegov izlaz? Zašto?

Dan 5

Funkcije

Iako je objektno-orijentisano programiranje skrenulo pažnju sa funkcija na objekte, funkcije, ipak, ostaju centralna komponenta svakog programa. Danas ćete naučiti

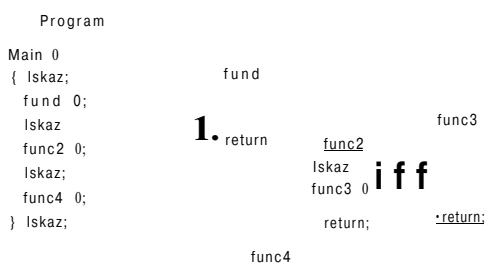
- šta je funkcija i koji su njeni delovi
- kako deklarisati i definisati funkcije
- kako funkcijama predati parametre
- kako vratiti vrednost iz funkcije.

v

Sta je funkcija?

Funkcija je, u stvari, potprogram, koji može raditi sa podacima i vratiti vrednost. Svaki C++ program ima bar jednu funkciju, main(). Kada Vaš program počne, automatski se poziva mai n(), koja će, možda, pozvati druge funkcije, od kojih će neke pozvati neke druge.

Svaka funkcija ima svoje ime i kada se ono susretne izvršenje programa se grana na njeno telo. Po povratku iz funkcije, izvršenje se nastavlja na sledećoj liniji pozivajuće funkcije. Ovaj tok je ilustrovan na slici 5.1.



Dobro dizajnirane funkcije obavljaju specifičan i lako razumljiv zadatak. Komplikovane zadatke bi trebalo rastaviti na vise funkcija, a onda se svaka može pozvati.

Postoje dva tipa funkcija: korisnički-definisane i ugradene. Ugradene funkcije su deo paketa Vašeg kompajlera - njih obezbeđuje proizvođač.

Deklarisanje i definisanje funkcija

Korišćenje funkcija u Vašem programu zahteva da prvo deklarišete funkciju, a da je, onda, definišete. Deklaracija saopštava kompajleru ime, povratni tip i parametre funkcije i kako funkcija radi. Ni jedna funkcija se ne može pozvati iz neke druge funkcije ako nije prvo deklarisana. Deklaracija se naziva *prototip*.

Deklarisanje funkcije

Postoje tri načina da se deklariše funkcija:

- Zapišite Vaš prototip u datoteku, a, onda, upotrebite direktivu #i ncl ude, da biste je uključili u Vaš program.
- Zapišite prototip u datoteku, u kojoj se koristi Vaša funkcija.
- Definišite funkciju, pre nego što je pozove neka druga funkcija. Kada ovo uradite, defmicija se ponaša kao njena deklaracija.

Iako možete definisati funkciju pre njenog korišćenja i tako izbeći potrebu kreiranja prototipa funkcije, ovo nije dobra programska praksa zbog tri razloga.

Prvo - zahtevati da se funkcije pojave u datoteci u određenom redosledu je loša ideja. Ovo otežava održavanje programa dok se zahtevi menjaju.

Drugo - može da zatreba sposobnost funkcije `A()` da pozove funkciju BO, ali i da bi B() trebalo da bude sposobna da pozove `A()`, pod odredenim okolnostima. Nije

moguće definisati A() pre definisanja B() i B() pre defmisanja A(), pa se, zato, bar jedna od njih mora deklarisati, u svakom slučaju.

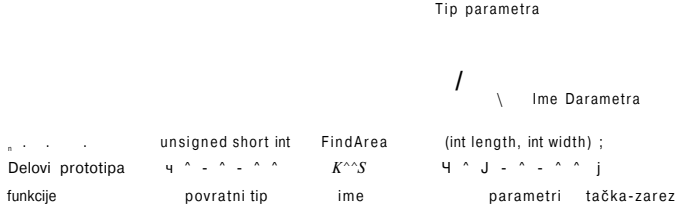
Treće - prototipovi funkcija su dobra i moćna tehnika dibagiranja. Ako Vaš prototip deklariše da Vaša funkcija prihvata odreden skup parametara, ili da vraća određen tip vrednosti, a onda Vaša funkcija ne odgovara prototipu, kompajler može označiti Vašu grešku, umesto da se čeka da se ona sama pokaže kada izvršite program.

Prototipovi funkcija

Za mnoge ugradene funkcije koje koristite već su napisani prototipovi u datotekama - uključujete ih u svoj program, korišćenjem #i ncl ude. Za funkcije koje sami pišete, morate uključiti prototip.

Prototip funkcije je iskaz, što znači da se završava znakom tačka-zarez. On se sastoji od povratnog tipa funkcije, imena i liste parametara.

Lista parametara je lista svih parametara i njihovih tipova, koji su odvojeni zarezima. Slika 5.2 ilustruje delove prototipa funkcije.



Prototip funkcije i defmicija funkcije se moraju potpuno slagati po povratnom tipu, imenu i listi parametara. Ako se ne slažu, dobićete grešku tokom kompajliranja. Uočite, ipak, da nije potrebno da prototip funkcije sadrži imena parametara, nego samo njihove tipove. Prototip koji ovako izgleda je perfektno istinit:

```
long Area(int, int);
```

Ovaj prototip deklariše funkciju nazvanu Area(), koja vraća long i ima dva parametra, oba celobrojna. Iako je ovo legalno, to nije dobra ideja. Dodavanje imena parametara čini Vaš prototip jasnijim. Ista funkcija sa imenovanim parametrima bi mogla biti:

```
Long Area(int length, int width);
```

Uočite da sve funkcije imaju povratni tip. Ako ni jedan nije eksplicitno naglašen, podrazumevani povratni tip je int. Vaši programi će, ipak, biti lakši za razumevanje, ako eksplicitno deklarisete povratni tip svake funkcije, uključujući i `main()`. Listing 5.1 demonstrira program koji uključuje prototip za funkciju Area().

Listing 5 1.: Deklaracija, defmicija i upotreba te funkcije

```
1   // Listing 5.1 - Demonstrira upotrebu prototipova funkcija
2
3   typedef unsigned short USHORT;
4   linclude<iostream.h>
5   USHORT FindArea(USHORT length, USHORT width); //prototip funkcije
6
7   int main()
8   {
9     USHORT lengthOfYard;
10    USHORT widthOfYard;
11    USHORT areaOfYard;
12
13    cout « "\nKoliko je široko Vase dvorište? ";
14    cin » widthOfYard;
15    cout « "\nKoliko je dugačko Vaše dvorište? ";
16    cin » lengthOfYard;
17
18    areaOfYard= FindArea(lengthOfYard,widthOfYard);
19
20    cout « "\nVaše dvorište je: ";
21    cout « areaOfYard;
22    cout « " kvadratnih stopa\n\n";
23          return 0;
24
25
26  USHORT FindArea(USHORT l, USHORT w)
27  {
28      return l * w;
29  }
```

```
Koliko je široko Vase dvorište? 100

Koliko je dugačko Vase dvorište? 200

Vaše dvorište je 20000 kvadratnih stopa
```

Prototip za funkciju FindArea() je u liniji 5. Uporedite prototip sa definicijom funkcije u liniji 26. Uočite da su isti ime, povratni tip i tipovi parametara. Kada bi se razlikovali, bila bi generisana greška kompajlera. U stvari, jedina zahtevana razlika je da se prototip funkcije završava znakom tačka-zarez i da nema telo.

Takođe uočite da su imena parametara u prototipu length i width, a imena u definiciji su l i w. Kao što je rečeno, imena u prototipu se ne koriste; ona su tu kao informacija za programera. Uključena, trebalo bi da odgovaraju implementaciji, kada je moguće. Ovo smanjuje konfuziju, ali se ne zahteva, kao §to to ovde vidite.

Argumenti se predaju funkciji u redu u kom su deklarisani i definisani, ali nema povezivanja imena. Da ste predali widthOfYard, za kojim sledi lenghtOfYard, funkcija FindArea() bi koristila vrednost u widthOfYard za length (širinu), a lenghtOfYard za

wi dth (dužinu). Telo funkcije se uvek zatvara u zagrade, čak i ako se sastoji iz samo " " ^ F * jednog iskaza, kao u ovom slučaju.

Definisanje funkcije

Defmicija funkcije se sastoji od funkcijskog zaglavlja i njenog tela. Zaglavlje je potpuno isto kao i prototip funkcije, izuzev što se parametri moraju imenovati i što nema završnog znaka tačka-zarez (;).

Telo funkcije je skup iskaza, koji su zatvoreni u zagrade. Slika 5.3 prikazuje zaglavlje i telo funkcije.

```
                povratni tip          jme              parametri

              Unsignd short int      FindArea      (int length, int width)


          - otvorena zagrada
{
    // iskazi

    return          ^Oe^Tgth^*jvi^

      \ ključna reč          \
                              povratna vrednost
Slika 5.3
Zaglavlje                 zatvorenazagrada
i telo funkcije
'
```

Funkcije

Sintaksa prototipa funkcije

povratni_tip *ime_funkcije* ([tip [imeParametra]]...);

Sintaksa definicije funkcije

povratni_tip *ime_funkcije* ([tip imeParametra]...)
{
    iskazi;
}

Prototip funkcije saopštava kompajleru povratni tip, ime i listu parametara. Ne zahteva se da funkcije imaju parametre, a, ako imaju, ne zahteva se da prototip prikaže njihova imena, nego samo njihove tipove. Prototip se uvek završava znakom tačka-zarez.

Definicija funkcije se mora slagati u povratnom tipu i listi parametara sa svojim prototipom. Ona mora obezbediti imena za sve parametre, a telo definicije funkcije mora biti okruženo zagradama. Svi iskazi unutar tela funkcije se moraju završavati

znakom tačka-zarez, ali sama funkcija **se ne završava tim znakom,** već zatvarajućom zagradom.

Ako funkcija **vraća** vrednost, trebalo **bi** da **se** završi **return** iskazom, iako **se** on može legalno pojaviti **bilo gde u telu** funkcije.

Svaka funkcija ima povratni tip. Ako ni jedan nije eksplicitno izabran, povratni tip će biti **int.** Osigurajte **da je** svakoj funkciji dat eksplicitni povratni tip. Ako funkcija ne vraća vrednost, njen povratni tip će biti void.

Primeri prototipova funkcija

```
long FindArea(long length, long width);  // vraća long, ima dva parametra
void PrintMessage(int messageNumber);    // vraća void, ima jedan parameter
int GetChoiceO;                          // vraća int, nema parametara
BadFunction();                           // vraća int, nema parametara
```

Primeri defmicija funkcija

```
long Area(long 1, long w)
{
     return 1 * w;
}
void PrintMessage(int whichMsg)
{
     if (whichMsg == 0)
          cout « "Zdravo.\n";
     if (whichMsg == 1)
          cout « "ZbogomAn";
     if (whichMsg > 1)
          cout « "Ja sam zbunjen.\n";
)
```

Izvršenje funkcija

Kada pozovete funkciju, izvršenje počinje od prvog iskaza posle otvarajuće zagrade ({). Grananje se može ostvariti korišćenjem iskaza if (i sličnih iskaza koji će biti opisani u Danu 7, "Vise o toku programa"). Funkcije mogu pozivati druge funkcije, a, čak, i same sebe (pogledajte sekciju "Rekurzija", kasnije u ovoj glavi).

Lokalne promenljive

Ne samo da možete predati promenljive funkciji, nego i deklarisati promenljive unutar tela funkcije. Ovo se obavlja korišćenjem lokalnih promenljivih, koje su tako nazvane jer postoje samo lokalno unutar same funkcije. Po povratku iz funkcije, lokalne promenljive nisu vise raspoložive.

Lokalne promenljive se definišu kao i sve druge promenljive. Parametri, koji se predaju funkciji, takode se smatraju lokalnim promenljivim i mogu se koristiti na isti

način kao da su bili definisani unutar tela funkcije. Listing 5.2 je primer korišćenja parametara i lokalno definisanih promenljivih unutar funkcije.

H* **Listing 5.2: Upotreba lokalnih promenljivih i parametara**

```
linclude <iostream.h>

float Convert(float);
int  main()
{
   float  TempFer;
   float  TempCel;

   cout « "Molim Vas, unesite temperaturu u Farenhajtovim stepenima:
   cin  » TempFer;
   TempCel  = Convert(TempFer);
   cout « "\nEvo temperature u Celzijusovim stepenima:  ";
   cout « TempCel « endl;
          return 0;


float  Convert(float  TempFer)
(
   float  TempCel;
   TempCel  = ((TempFer - 32) * 5) / 9;
   return TempCel;
```

```
Molim Vas, unesite temperaturu u Farenhajtovim stepenima: 212
Evo temperature u Celzijusovim stepenima:  100
Molim Vas, unesite temperaturu u Farenhajtovim stepenima: 32
Evo temperature u Celzijusovim stepenima: 0
Molim Vas, unesite temperaturu u Farenhajtovim stepenima: 85
Evo temperature u Celzijusovim stepenima: 29.4444
```

U liniji 6 i 7 deklarisane su dve float promenljive - jedna da čuva temperaturu u Farenhajtovim, a druga u Celzijusovim stepenima. Od korisnika se u liniji 9 traži da unese vrednost temperature u Farenhajtovim stepenima, i ta vrednost se predaje funkciji `Covert()`.

Izvršenje "skače" na prvu liniju funkcije `Convert` () u liniji 19, gde je deklarisana lokalna promenljiva, takođe nazvana `TempCel`. Uočite da ona nije isto što i promenljiva `TempCel` u liniji 7. Ova promenljiva postoji samo unutar funkcije `Convert()`. Vrednost koja se predaje kao parametar, `TempFer`, samo je lokalna kopija promenljive koja se predaje u funkciji `main()`.

Ova funkcija može imenovati parametar sa `FerTemp` i lokalnu promenljivu sa `CelTemp` i program bi radio jednako dobro. Vi možete ponovo uneti ova imena i rekompajlirati program, da biste videli ovaj rad.

Lokalnoj funkcijskoj promenljivoj `TempCel` se dodeljuje vrednost koja rezultira iz oduzimanja 32 od parametra `TempFer`, množenjem sa 5, a, cnda, deljenjem sa 9. Ova vrednost se onda vraća kao povratna vrednost funkcije i ona se u liniji 11 dodeljuje promenljivoj `TempCel` u funkciji `main()`. Vrednost se štampa u liniji 13.

Program se izvršava tri puta. Prvi put, vrednost 212 se predaje da bi se proverilo da tačka ključanja vode u Farenhajtovim stepenima (212) generiše ispravan odgovor u Celzijusovim stepenima (100). Drugi test je tačka smrzavanja vode. Treći test je slučajan broj izabran za generisanje frakcionog rezultata.

Kao vežbu, pokušajte da ponovo unesete program sa drugim imenima za promenljive, kao što je to ovde ilustrovano:

```
#include <iostream.h>

float Convert(float);
int main()
{
   float TempFer;
   float TempCel;

   cout « "Molim vas unesite temperaturu u Ferinhajtima:
   cin » TempFer;
   TempCel = Convert(TempFer);
   cout « "\nEvo temperature u Celzijusima: ";
   cout « TempCel « endl;


float Convert(float Fer)
{
   float Cel;
   Cel = ((Fer - 32) * 5) / 9;
   return Cel;
```

Trebalo bi da dobijete iste rezultate.

Promenljiva ima *opseg,* koji odreduje koliko dugo je ona raspoloživa Vašem programu i gde joj se može pristupiti. Promenljive, deklarisane unutar bloka ograničene su na taj blok; njima se može pristupiti samo unutar njega i "izlaze izvan postojanja" kada se on završi. Globalne promenljive imaju globalni opseg i na raspolaganju su u svakom delu programa.

Normalno je opseg očigledan, ali postoje neki zavaravajući izuzeci. Trenutno, promenljive, deklarisane unutar zaglavlja petlje for `(for int i = 0; i<SomeValue; i++)`, ograničene su na blok u kome se petlja `for` kreira, ali priprema se promena u službenom C++ standardu.

Ništa od ovoga stvari ne predstavlja problem, ako ste pažljivi da ne koristite ponovo imena Vaših promenljivih unutar date funkcije.

Globalne promenljive

Promenljive, koje su definisane izvan svake funkcije, imaju globalni opseg, pa su, tako, raspoložive u svakoj funkciji u programu, uključujući i `main()`.

Lokalne promenljive ne menjaju globalne promenljive i kad imaju ista imena - sakrivaju ih. Ako funkcija ima lokalnu promenljivu sa istim imenom kao i globalna promenljiva, ime se odnosi na lokalnu, ne globalnu, kada se koristi unutar funkcije. Listing 5.3 ilustruje ove činjenice.

Listing 5.3: Demonstriranje globalnih i lokalnih promenljivih

```
linclude <iostream.h>
void myFunction();          // prototip

int x = 5, y = 7;           // globalne promenljive
int main()


     cout « "x iz main:   « x «  "\n";
     cout « "y iz main:   « y «  "\n\n";
     myFunction();
     cout « "Povratak iz myFunction!\n\n";
     cout « "x iz main: " « x « "\n";
     cout « "y iz main: " « y « "\n";
     return 0;


void myFunction()
{
     int y = 10;

     cout « "x iz myFunction: " « x « "\n";
     cout « "y iz myFunction: " « y « "\n\n"
```

```
>ШБПШ^ x iz main:
....."      y iz main:

            x iz myFunction: 5
            y iz myFunction: 10

            Povratak iz myFunction

            x iz main: 5
            y iz main: 7
```

Ovaj jednostavan program ilustruje nekoliko ključnih i potencijalno zbunjujućih činjenica o lokalnim i globalnim promenljivim. U liniji 1 su

deklarisane **dve** globalne promenljive, x i **y.** Globalna promenljiva x se inicijalizuje vrednošću 5, **a** globalna promenljiva **y** vrednošću 7.

U linijama 8 i 9, u funkciji main(), **ove** vrednosti **se** štampaju na ekranu. Uočite da funkcija **mainQ** ne definiše ni jednu promenljivu; pošto su one globalne, već su raspoložive **za main().**

Kada **se pozove myFunction()** u liniji 10, izvršenje programa prelazi na liniju 18 i lokalna promenljiva, **y,** definiše **se** i inicijalizuje vrednošću 10. U liniji 21 **myFuncti on** () štampa vrednost promenljive x, a koristi se globalna promenljiva x, kao **što je to bio** slučaj u **main().** U liniji 22, ipak, kada se **koristi** ime promenljive **y,** koristi se lokalna promenljiva, sakrivajući globalnu promenljivu sa istim imenom.

Funkcijski poziv se završava i kontrola se vraća funkciji `main()`, koja ponovo štampa vrednosti u globalnim promenljivim. Uočite da na globalnu promenljivu y uopšte ne utiče vrednost koja je dodeljena lokalnoj promenljivoj y funkcije `myFuncti on()`.

Globalne promenljive: reč upozorenja

U C + + , globalne promenljive su legalne, ali se skoro nikada ne koriste. C+ + je izrastao iz C-a, čije globalne promenljive su opasna, ali neophodna alatka. One su neophodne, jer postoje trenuci kada je potrebno da programer učini podatak raspoloživim mnogim funkcijama, a ne želi da prosledi taj podatak, kao parametar, od funkcije do funkcije.

Globali su opasni, jer su oni deljeni podaci i jedna funkcija može promeniti globalnu promenljivu na način koji je nevidljiv za drugu funkciju. Ovo kreira bagove, koje je veoma teško pronaći.

U Danu 14, "Specijalne klase i funkcije", videćete moćnu alternativu za globalne promenljive, koju nudi C + + , ali nije raspoloživa u C-u.

Vise o lokalnim promenljivim

Za promenljive koje su deklarisane unutar funkcije se kaže da imaju "lokalni opseg". To znači da su one vidljive i mogu se koristiti samo unutar funkcije u kojoj su definisane. U stvari, u C+ + promenljive možete definisati bilo gde unutar funkcije, ne samo na njenom vrhu. Opseg promenljive je blok u kojem je ona definisana. Ako defmišete promenljivu unutar skupa zagrada unutar funkcije, ona je raspoloživa samo unutar tog bloka. Listing 5.4 ilustruje ovu ideju.

Listing 5.4: Promenljive koje su ograničene na blok.

```
1   // Listing 5.4 - demonstrira promenljive
2   // koje su ogranicene na blok
3
4   #include <iostream.h>
5
6   void myFunc();
7
8   int  main()
9   {
10     int x = 5;
11
12     cout « "\nU main x je: " « x;
13
14     myFunc();
15     cout « "\nPovratak u main, x je: " « x;
16       return 0;
17
18
19  void myFunc()
20  {
21
22     int x = 8;
23     cout « "\nU myFunc, lokalni x je: " « x « endl;
24
25
26        cout « "\nU bloku unutar myFunc, x je: " « x;
27
28        int x = 9;
29
30        cout « "\nIzrazito lokalni x: " « x;
31     }
32
33    cout « "\nIzvan bloka, u myFunc, x: " « x « endl;
34
```

```
U main x je: 5
U myFunc, lokalni x je: 8

U bloku unutar myFunc, x je: 8
Izrazito lokalni x: 9
Izvan bloka, u myFunc, x: 8

Povratak u main, x je: 5
```

Program počinje inicijalizacijom lokalne promenljive, x, u liniji 10, u `main()`. Izlaz u liniji 11 verifikuje da je x inicijalizovano vrednošću 5.

Poziva se `MyFunc()` i lokalna promenljiva nazvana x, inicijalizuje se vrednošću 8 u liniji 22. Njena vrednost se štampa u liniji 23.

Blok počinje u liniji 25 i ponovo se štampa promenljiva x iz funkcije u liniji 26. Kreira se nova promenljiva, takode nazvana x, ali lokalna za blok, u liniji 28 i inicijalizuje se vrednošću 9.

Vrednost najnovije promenljive x se štampa u liniji 30. Lokalni blok se završava u liniji 31, a promenljiva kreirana u liniji 28, odlazi "izvan opsega" i nije vise vidljiva.

U liniji 33 štampa se x koje je bilo deklarisano u liniji 22. Na njega nije uticalo x definisano u liniji 28; njegova vrednost je još uvek 8.

U liniji 34 myFunc () odlazi izvan opsega i njena lokalna promenljiva x postaje neraspoloživa. Izvršenje se vraća na liniju 15 i štampa se vrednost lokalne promenljive x, koja je kreirana u liniji 10. Na nju ne utiče ni jedna promenljiva definisana u myFunc().

Nepotrebno je posebno naglašavati da bi ovaj program bio daleko manje konfuzan da su ove tri promenljive dobile jedinstvena imena!

Funkcijski iskazi

Virtuelno ne postoji ograničenje za broj, ili tipove iskaza, koji se mogu naći u telu funkcije. Iako ne možete definisati drugu funkciju unutar funkcije, možete pozvati neku funkciju i, naravno, main() upravo to radi u skoro svakom C++ programu. Funkcije mogu, čak, pozivati same sebe.

Iako ne postoji ograničenje za veličinu funkcije u C++, dobro dizajnirane funkcije teže da budu male. Mnogi programeri preporučuju održavanje funkcija dovoljno kratkim da stanu na jedan ekran, tako da se može videti cela funkcija odjednom. Ovo je praktično rešenje koje, često, ne poštuju veoma dobri programeri, ali manje funkcije su lakše za razumevanje i održavanje.

Svaka funkcija bi trebalo da obavlja jedan Iako razumljiv zadatak. Ako Vaše funkcije postaju velike, potražite mesta gde ih možete podeliti na komponentne zadatke.

Argument! funkcije

Ne moraju svi argumenti funkcije biti istog tipa. Savršeno je razumno napisati funkciju koja prihvata celobrojnu vrednost, dva long-a i karakter, kao svoje argumente.

Svaki važeći C++ izraz može biti argument funkcije, uključujući konstante, matematičke i logičke izraze i druge funkcije koje vraćaju vrednost.

Korišćenje funkcija kao parametara za funkcije

Iako je legalno za jednu funkciju da prihvati, kao parametar, drugu funkciju koja vraća vrednost, to može biti razlog da kod postane težak za čitanje i dibagiranje.

Kao primer, recimo da imate funkcije double(), tripleO, squaref), i cube(), od kojih svaka vraća vrednost. Možete napisati

```
Answer = (double(triple(square(cube(myValue)))));
```

Ovaj iskaz prihvata promenljivu, my Value, i predaje je, kao argument, funkciji cube() njena povratna vrednost se predaje, kao argument, funkciji square(), čija se povratna vrednost predaje funkciji tripleO, a ta povratna vrednost funkciji double(). Povratna vrednost ovog udvostručenog, utrostručenog, kvadriranog i kubiranog broja se sada predaje funkciji Answer.

Teško je biti potpuno siguran šta ovaj kod radi (da li je vrednost utrostručena pre, ili posle kvadriranja?) i, ako je odgovor pogrešan, biće teško pronaći koja je funkcija izneverila.

Alternativa je da se svaki korak dodeli sopstvenoj prelaznoj promenljivoj:

```
unsigned long myValue = 2;
unsigned long cubed   =   cube(myValue);        // kubiran = 8
unsigned long squared = square(cubed);          // kvadriran = 64
unsigned long tripled = triple(squared);        // utrostrucen = 196
unsigned long Answer =  double(tripled);        // Answer = 392
```

Sada se svaki prelazni rezultat može ispitati, a redosled izvršenja je eksplicitan.

Parametri su lokalne promenljive

Argumenti koji se predaju funkciji su za nju lokalni. Promene pravljene na argumentima ne utiču na vrednosti u pozivajućoj funkciji. Ovo je poznato kao predavanje po vrednosti, što znači da se u funkciji pravi lokalna kopija svakog argumenta. Ove lokalne kopije se tretiraju kao i sve ostale lokalne promenljive. Listing 5.5 ilustruje ovu činjenicu.

Listing 5.5: Demonstrira predavanje po vrednosti.

```
// Listing 5.5 - demonstrira predavanje po vrednosti

#include <iostream.h>

void swap(int x, int y);

int main()
{
  int x = 5, y = 10;

    cout « "Main. Pre zamene, x: " « x « " y: " « y « "\n";
    swap(x.y);
    cout « "Main. Posle zamene, x: " « x « " y: " « y « "\n"
       return 0;
```

```
void swap (int x, int y)
{
  int temp;

  cout « "Swap. Pre zamene, x: " « x « " y: " « y « "\n";

  temp • x;
  x = y;
  y = temp;

  cout « "Swap.Posle zamene, x: " « x « " y: " « y « "\n";
```

```
Main. Pre zamene, x: 5 y: 10
Swap. Pre zamene, x: 5 y: 10
Swap. Posle zamene, x: 10 y: 5
Main. Posle zamene, x: 5 y: 10
```

Ovaj program inicijalizuje dve promenljive u mainQ, a onda ih predaje funkciji swap(), za koju se veruje da ih zamenjuje. Ipak, kada se ponovo ispitaju u main(), one su nepromenjene!

Promenljive se inicijalizuju u liniji 9, a njihove vrednosti se prikazuju u liniji 11. Poziva se swapQ i predaju se promenljive.

Izvršenje programa prelazi na funkciju swapQ, gde se u liniji 21 ponovo štampaju vrednosti. One su u istom redu u kome su bile u mainQ, kao što se i očekuje. Od linije 23 do 25 vrednosti se zamenjuju i ova akcija se potvrđuje izlazom u liniji 27. Tokom funkcije swap(), vrši se zamena ovih vrednosti.

Izvršenje se vraća na liniju 13, nazad u main(), gde vrednosti nisu vise zamenjene.

Kao što ste zaključili, vrednosti su predate funkciji swap() po vrednosti. To znači da su napravljene kopije vrednosti koje su lokalne za swap(). Ove lokalne promenljive se zamenjuju od linije 23 do 25, ali na promenljive u main() se ne utiče.

U Danima 8. i 10. videćete alternative za predavanje po vrednosti, koje će dozvoliti da vrednosti u mainQ budu promenjene.

Povratne vrednosti

Funkcije vraćaju vrednost, ili void. Void je signal kompajleru da nikakva vrednost neće biti vraćena.

Da biste vratili vrednost iz funkcije, napišite ključnu reč return - za njom sledi vrednost koju želite da vratite. Sama vrednost će, možda, biti izraz koji vraća vrednost. Na primer:


```
return 5;
return (x > 5);
return (MyFunction());
```

Ovo su legalni return iskazi, uz pretpostavku da funkcija MyFuncti on () vraća vrednost. Vrednost u drugom iskazu, return (x > 5), biće nula, ako x nije veće od 5, ili će biti 1. Ono što se vraća je vrednost izraza, 0 (neistina), ili 1 (istina), ne vrednost od x.

Kada se susretne ključna reč return, izraz koji sledi se vraća kao vrednost funkcije. Izvršenje programa se odmah vraća u pozivajuću funkciju, a iskazi koji slede posle return se ne izvršavaju.

Legalno je imati vise od jednog return iskaza u jednoj funkciji. Listing 5.6 ilustruje ovu ideju.

Listing 5.6: Demonstracija višestrukih return iskaza

```
1   // Listing 5.6 - demonstrira visestruke return
2   // iskaze
3
4   linclude <iostream.h>
5
6   int Doubler(int AmountToDouble);
7
8   int main()
9   {
10
11      int result = 0;
12      int input;
13
14      cout « "Unesite broj izmedju 0 i 10,000 za udvostrucenje: ";
15      cin » input;
16
17      cout « "\nPre pozivanja udvostrucivaca... ";
18      cout « "\nulaz: " « input « "    udvostrucen: " « result « "\n"
19
20      result = Doubler(input);
21
22      cout « "\nPovratak iz udvostrucivaca..An";
23      cout « "\nulaz: " « input « "    udvostrucen: " « result « "\n"
24
25
26      return 0;
27
28
29  int Doubler(int original)
30  {
31      if (original <= 10000)
32          return original * 2;
33      el se
```

```
        return -1;
    cout « "Ovde ne mozete!\n";


Unesite broj izmedju 0 i 10.000 za udvostrucenje: 9000

Pre pozivanja udvostrucivaca...
ulaz: 9000   udvostrucen: 0

Povratak iz udvostrucivaca...

ulaz: 9000   udvostrucen: 18000

Unesite broj izmedju 0 i 10,000 za udvostrucenje: 11000

Pre pozivanja udvostrucivaca...
ulaz: 11000 udvostrucen: 0

Povratak iz udvostrucivaca...
ulaz: 11000  udvostrucen: -1
```

jTOTFjd^ Broj se zahteva u linijama 14 i 15 i štampa u liniji 18, zajedno sa lokalnom promenljivom result. Funkcija Doubler() se poziva u liniji 20, i ulazna vrednost se predaje kao parametar. Rezultat će biti dodeljen lokalnoj promenljivoj result, a vrednosti će se ponovo odštampati u linijama 22 i 23.

U liniji 31, u funkciji Doubl er(), testira se parametar, da bi se videlo da li je veći od 10.000. Ako nije, funkcija vraća dvostruki originalni broj. Ako je veći od 10.000, funkcija vraća -1 kao vrednost greške.

Iskaz u liniji 35 se nikada ne dostiže, jer, bez obzira da li je vrednost veća, ili ne od 10.000, dolazi do povratka iz funkcije, pre nego što se dode do linije 35, ili na liniji 32, ili liniji 34. Dobar kompajler će upozoriti da se ovaj iskaz ne može izvršiti, i dobar programer će ga ukloniti!

Podrazumevani parametri

Za svaki parametar koji deklarišete u prototipu i definiciji funkcije pozivajuća funkcija mora predati vrednost. Predata vrednost mora biti deklarisanog tipa. Ako imate funkciju deklarisanu kao funkcija

```
long myFunction(int);
```

funkcija, u stvari, mora prihvatiti celobrojnu promenljivu. Ako se definicija funkcije razlikuje, ili ako ne predate celobrojnu vrednost, dobićete grešku kompajlera.

Izuzetak od ovog pravila je ako prototip funkcije deklariše podrazumevanu vrednost za parametar. Podrazumevana vrednost je vrednost koja se korsti ako ni jedna nije obezbedena. Prethodna deklaracija se može ponovo napisati kao


```
long myFunction (int x = 50);
```

Ovaj prototip kaže: **"myFunction**() vrača **long,** a prihvata celobrojni parametar. Ako argument nije obezbeden, upotrebi podrazumevanu vrednost **50".** Zato **što** se imena parametara ne zahtevaju u prototipovima funkcija, ova deklaracija je mogla biti napisana kao

```
long myFunction (int = 50);
```

Defmicija funkcije se ne menja deklarisanjem podrazumevanog parametra. Zaglavlje funkcijske definicije za ovu funkciju bi bilo

```
long myFunction (int x)
```

Kada pozivajuća funkcija ne bi uključila parametar, kompajler bi popunio x podrazumevanom vrednošću 50. Ime podrazumevanog parametra u prototipu ne treba da bude isto kao i ime u zaglavlju funkcije; podrazumevana vrednost se dodeljuje po poziciji, a ne po imenu.

Nekim, ili svim parametrima funkcije se mogu dodeliti podrazumevane vrednosti. Jedna restrikcija je: ako neki od parametara nema podrazumevanu vrednost, ni jedan predhodni parametar ne može imati podrazumevanu vrednost.

Ako prototip funkcije izgleda kao

```
long myFunction (int Paraml, int Param2, int Parat3);
```

možete dodeliti podrazumevanu vrednost parametru **Param2** samo ako ste dodelili podrazumevanu vrednost parametru **Parat3.** Možete dodeliti podrazumevanu vrednost parametru **Paraml,** samo ako ste dodelili podrazumevane vrednosti parametrima **Param2** i **Parat3.** Listing 5.7 demonstrira upotrebu podrazumevanih vrednosti.

Listing 5.7. Demonstracija podrazumevanih vrednosti parametara.

```
1:   // Listing 5.7 - demonstrira upotrebu
2:   // podrazumevanih vrednosti parametara
3:
4:  #include <iostream.h>
5:
6:   int AreaCube(int length, int width = 25, int height = 1);
7:
8:   int main()
9:  {
10:      int length = 100;
11:      int width = 50;
12:      int height = 2;
13:      int area;
14:
15:      area = AreaCube(length, width, height);
16:      cout « "Prva zapremina iznosi: " « area « "\n";
17:
```

```
    area ° AreaCube(length, width);

    cout « "U drugom pozivu zapremina iznosi: " « area « "\n"

    area * AreaCube(length);
    cout « "U trecem pozivu zapremina iznosi: " « area « "\n"
     return 0;

AreaCube(int length, int width, int height)
{

    return (length * width * height);
```

```
Prva zapremina iznosi: 10000
U drugom pozivu zapremina iznosi: 5000
U trecem pozivu zapremina iznosi: 2500
```

U liniji **6** prototip AreaCube() odreduje da funkcija AreaCubeQ prihvata tri celobrojna parametra. Poslednja dva imaju podrazumevane vrednosti.

Ova funkcija izračunava zapreminu kvadrata, čije se dimenzije predaju. Ako width nije predata, koristi se width od 25 i height od 1. Ako je predata width, ali ne i height, koristi se height od 1. Nije moguće predati height bez predavanja width.

U linijama 10-12 se inicijalizuju dimenzije length, height i width i predaju se funkciji AreaCubef) u liniji 15. Vrednosti se proražunavaju, i rezultat se štampa u liniji 16.

Izvršenje se vraća na liniju 18, gde se ponovo poziva AreaCubeQ, ali bez vrednosti za height. Koristi se podrazumevana vrednost i ponovo se dimenzije proračunavaju i štampaju.

Izvršenje se vraća na liniju 21 i ovogputa se ne predaje ni width, ni height. Izvršenje se grana, po treći put, na liniju 27. Koriste se podrazumevane vrednosti. Zapremina se proračunava, a onda štampa.

<jffPAaTS**§·** Zapamtite da se parametri funkcije ponašaju kao lokalne promenljive unutar funkcije.

Nemojte pokušati da kreirate podrazumevanu vrednost za prvi parametar ako nema podrazumevane vrednosti za drugi.

Nemojte zaboraviti da argumenti predati po vrednosti ne mogu uticati na promenljive u pozivajućoj funkciji.

Nemojte zaboraviti da promene globalne promenljive u jednoj funkciji menjaju tu promenljivu za sve funkcije.

Preklapajuće funkcije

C + + Vam omogućava da kreirate vise od jedne funkcije sa istim imenom. Ovo se naziva preklapanje funkcija. Funkcije se moraju razlikovati po svojim listama parametara, različitim tipom parametra, različitim brojem parametara, ili i jednim i drugim. Evo primera:

```
int myFunction (int, int);
int myFunction (long, long);
int myFunction (long);
```

myFuncti on () je preklopljena sa tri različite liste parametara. Prva i druga verzija se razlikuju po tipovima, a treća po broju parametara.

Povratni tipovi mogu biti isti, ili različiti u preklopljenim funkcijama. Trebalo bi da uočite da dve funkcije sa istim imenom i listom parametara, ali različitim povratnim tipovima, generišu grešku kompajlera.

ДРРJДJJ Preklapanje funkcija se takode naziva *funkcijski polimorfizam. Poly* znači vise, a *morph* znači forma: polimorfna funkcija je vise - formna.

Funkcijski polimorfizam se odnosi na sposobnost da se funkcija "optereti" sa vise od jednog značenja. Menjanjem broja, ili tipa parametara, možete dvema, ili trima funkcijama dodeliti isto funkcijsko ime, a ispravna će biti pozvana, na osnovu podudarnosti korišćenih parametara. Ovo Vam dozvoljava da kreirate funkciju koja može dati prosek celobrojnih, dvostrukih i drugih vrednosti, bez potrebe za kreiranjem individualnih imena za svaku funkciju, kao što su AveragelntsO, AverageDoublesf) i tako dalje.

Pretpostavimo da pišete funkciju koja udvostručuje svaki ulaz koji joj date. Želeli biste da možere da predate int, long, float, ili double. Bez preklapanja funkcija, morali biste da kreirate četiri funkcijska imena:

```
int  DoubleInt(int);
long  DoubleLong(long);
float DoubleFloat(float);
double  DoubleDouble(double);
```

Sa preklapanjem funkcija, napravićete ovu deklaraciju:

```
int  Double(int);
long  Double(long);
float Double(float);
double  Double(double);
```

Ovo je lakše za čitanje i korišćenje. Ne morate da brinete koju da pozovete; Vi samo predajete promenljivu, a ispravna funkcija se poziva automatski. Listing 5.8 ilustruje upotrebu preklapanja funkcija.

Listing 5.8: Pemonstradja funkcijskog poiimorfiima

```
1:    // Listing 5.8 - demonstrira
2:    // funkcijski  polimorfizam
3:
4:    linclude <iostream.h>
5:
6:    int  Oouble(int);
7:    long Oouble(long);
8:    float Double(float);
9:    double Double(double);
10:
11:   int  mainQ
12:   {
13:       int       mylnt = 6500;
14:       long      myLong = 65000;
15:       float     myFloat = 6.5F;
16:       double    myDouble = 6.5e20;
17:
18:       int        doubledInt;
19:       long      doubledLong;
20:       float     doubledFloat;
21:       double    doubledDouble;
22:
23:       cout « "mylnt:  " « mylnt « "\n";
24:       cout « "myLong:  " « myLong « "\n";
25:       cout « "myFloat:  " « myFloat « "\n";
26:       cout « "myDouble:  " « myDouble « "\n";
27:
28:       doubledInt = Double(mylnt);
29:       doubledLong = Double(myLong);
30:       doubledFloat = Double(myFloat);
31:       doubledDouble = Double(myDouble);
32:
33:       cout « "doubledInt:  " « doubledInt « "\n";
34:       cout « "doubledLong:  " « doubledLong « "\n";
35:       cout « "doubledFloat:  " « doubledFloat « "\n";
36:       cout « "doubledDouble:  " « doubledDouble « "\n";
37:
38:        return 0;
39:   }
40:
41:   int Double(int original)
42:  {
43:     cout « "U Double(int)\n";
44:     return 2 * original;
45:  }
46:
47:   long Double(long original)
48:  {
```



```
  cout « "U Double(long)\n";
  return 2 * original;


float  Double(float  original)
{
  cout « "U Double(float)\n";
  return 2 * original;


double Double(double original)
{
  cout « "U Double(double)\n";
  return 2 * original;
```

```
mylnt:  6500
myLong: 65000
myFloat:  6.5
myDouble: 6.5e+20
U  Double(int)
U Double(long)
U  Double(float)
U Double(double)
DoubledInt:  13000
DoubledLong: 130000
DoubledFloat:  13
DoubledDouble: 1.3e+21
```

Funkcija `Double()` je preklopljena sa `int`, `long`, float i `double`. Prototipovi su u linijama 6-9, a definicije su u linijama 41-63.

U telu glavnog programa deklarisano je osam lokalnih promenljivih. U linijama 13 - 16 inicijalizovane su četiri vrednosti, a u linijama 28-31, pomoću druge četiri dodeljeni su rezultati predavanja prve četiri funkciji `Double()`. Uočite da kada se pozove `Double()` pozivajuća funkcija ne odlučuje koju da pozove; ona samo predaje argument, a ispravna se poziva.

Kompajler ispituje argumente i bira koju od četiri `Double()` funkcija da pozove. Izlaz otkriva da je svaka od njih pozvana po redosledu.

Specijalne feme o funkcijama

Pošto su funkcije centralni pojam u programiranju, javlja se nekoliko specijalnih tema, koje mogu biti od interesa kada susretnete neobične probleme. Kada se koriste pametno, inline funkcije Vam mogu pomoći da dobijete taj poslednji delić performansi. Rekurzija funkcija je jedan od onih divnih, ezoteričnih delova programiranja koji, povremeno, mogu "preseći" težak problem.

Inline funkcije

Kada definišete funkciju, kompajler, normalno, kreira samo jedan skup instrukcija u memoriji. Kada pozovete funkciju, izvršenje programa "skače" na te instrukcije, a, po povratku iz funkcije, izvršenje se vraća na sledeću liniju u pozivajućoj funkciji.' Ako pozovete funkciju 10 puta, Vaš program "skače" na isti skup instrukcija svaki put. Ovo znači da postoji samo jedna kopija funkcije, a ne 10.

Postoji osobina koja je iznad skakanja na funkcije i iz funkcija. Ispostavlja se da su neke funkcije veoma male, samo linija, ili dva koda, i postiže se efikasnost ako program može izbeći ove skokove, da bi izvršio samo jednu, ili dve instrukcije. Kada programeri govore o efikasnosti, oni, obično, misle na brzinu: program se izvršava brže, ako se može izbeći funkcijski poziv.

Ako je funkcija deklarisana ključnom rečju `inline`, kompajler ne kreira pravu funkciju: on kopira kod iz inline funkcije, direktno u pozivajuću funkciju. Ne obavlja se nikakav skok; to je kao da ste napisali iskaze u pozivajućoj funkciji.

Uočite da cena korišćenja inline funkcija može biti visoka. Ako se funkcija pozove 10 puta, inline kod se kopira u pozivajuće funkcije za svaki od ovih 10 poziva. Sićušno poboljšanje brzine, koje ćete, možda, postići vise je nego "preplavljeno" povećanjem veličine izvršnog programa. Čak i povećanje brzine može biti prividno. Današnji optimizacioni kompajleri sami obavljaju odličan posao i skoro nikada nema velikog dostignuća deklarisanjem inline funkcije. Što je važnije, povećana brzina donosi sopstvenu cenu performansi.

Šta praktično uraditi? Ako imate malu funkciju, jedan, ili dva iskaza, ona je kandidat za inline. Kada ste u nedoumici, ipak, je izostavite. Listing 5.9 demonstrira `inline` funkciju.

Listing 5.9: Demonstrira inline funkciju

```
// Listing 5.9 - demonstrira inline funkciju

#include<iostream.h>

inline int Double(int);

int main()
{
  int target;

  cout « "Unesite broj: ";
  cin » target;
  cout « "\n";

  target = Double(target);
  cout « "Rezultat: " « target « endl;

  target = Double(target);
  cout « "Rezultat: " « target « endl;


  target = Double(target);
  cout « "Rezultat: " « target « endl;
     return 0;
}

int Double(int target)
{
  return 2*target;
```

```
Unesite broj: 20

Rezultat: 40
Rezultat: 80
Rezultat: 160
```

fOTTHIfc^ U liniji 5 Double() je kao inline funkcija, koja prihvata `int` parametar i :u::rn"""*^ vraća `int`. Deklaracija je kao i svaki drugi prototip, izuzev što je ključna reč `inline` dodata neposredno pre povratne vrednosti.

Ovo se kompajlira u kod, koji je isti kao da ste napisali sledeće:

```
target = 2 * target;
```

na svakom mestu gde ste uneli:

```
target = Double(target);
```

Do vremena izvršenja Vašeg programa, instrukcije su već na mestu, kompajlirane u OBJ datoteku. Ovim se izbegava "skakanje" pri izvršenju koda, po cenu većeg programa.

^**NAPOMENA**^ Inline |e predlog kompajleru da ugradi funkciju. Kompajler ima slobodu da ignoriše predlog i napravi pravi funkcijski poziv.

Rekurzija

Funkcija može pozvati samu sebe. Ovo se naziva rekurzija; može biti direktna, ili indirektna. Direktna je kada funkcija poziva samu sebe; indirektna je kada funkcija poziva drugu funkciju, koja, onda, poziva prvu funkciju.

Neki problemi se najlakše rešavaju rekurzijom, obično oni u-kojima radite sa podacima a onda na isti način i sa rezultatom. Oba tipa rekurzije, direktna i indirektna, dobijaju se dve varijacije: one koje se konačno završavaju i proizvode odgovor i one koje se nikada ne završavaju i prouzrokuju pad u vreme izvršenja. Programeri smatraju da je druga prilično "zabavna" (kada se dešava nekom drugom).

Važno je uočiti da se, kada funkcija poziva samu sebe, izvršava nova kopija te funkcije. Lokalne promenljive u drugoj verziji su nezavisne od lokalnih promenljivih u prvoj. Ne mogu direktno uticati jedne na druge vise nego što lokalne promenljive u main() mogu uticati na lokalne promenljive u bilo kojoj funkciji koju ona poziva, kao što je bilo ilustrovano u listingu 5.4.

Da bismo ilustrovali rešavanje problema, korišćenjem rekurzije, razmotrite Fibonacci-jevu seriju:

1,1,2,3,5,8,13,21,34...

Svaki broj, posle drugog, je zbir dva broja pre njega. Fibonacci - jev problem bi mogao biti traženje 12. broja u seriji.

Jedan od načina da se reši ovaj problem je da se ispita serija pažljivo. Prva dva broja su 1. Svaki naredni broj je zbir predhodna dva broja. Uopšteno govoreći, n-ti broj je zbir n - 2 i n -1, sve dok je $n > 2$.

Rekurzivne funkcije treba da zaustave uslov. Nešto se mora dogoditi da bi program prestao sa rekurzijom, ili se nikada neće završiti. U Fibonacci - jevoj seriji $n < 3$ je uslov za zaustavljanje.

Algoritam koji treba upotrebiti je ovaj:

1. Pitajte korisnika za poziciju u seriji.
2. Pozovite funkciju `fib()` sa torn pozicijom, predajući joj vrednost koju je korisnik uneo.
3. Funkcija `fib()` ispituje argument (n). Ako je n < 3, ona vraća 1; inače, `fib()` poziva samu sebe (rekurzivno), predajući n - 2, poziva sebe ponovo, predajući n - 1, i vraća sumu.

Ako pozovete `fib(l)`, ona vraća 1. Ako pozovete `fib(2)`, ona vraća 1. Ako pozovete `fib(3)`, ona vraća zbir pozivanja `fib(2)` i `fib(l)`. Zato što `fib(2)` i `fib(l)` vraćaju po 1, `fib(3)` će vratiti 2.

Ako pozovete `fib(4)`, ona vraća zbir pozivanja `fib(3)` i `fib(2)`. Pokazali smo da `fib(3)` vraća2, (pozivajući `fib(2)` i `fib(l))`, ida `fib(2)` vraća jedan, tako daće `fib(4)` sabirati ove brojeve i vratiti 3, što je četvrti broj u seriji.

Praveći ovaj korak dalje, ako pozovete `fib(5)`, ona će vratiti zbir `fib(4)` i fib(3). Pokazali smo da `fib(4)` vraća 3, a `fib(3)` vraća 2, pa će vraćena zbir biti 5.

Ovaj metod nije najefikasniji način za resavanje problema (za `fib(20)` funkcija `fib()` se poziva 13.529 puta!), ali funkcioniše. Budite pažljivi: ako unesete preveliki broj, nestaće Vam memorije. Svaki put kada se pozove `fib()`, memorija se rezerviše. Po povratku iz nje, memorija se oslobada. Sa rekurzijom, memorija nastavlja da se rezerviše, pre nego što se oslobodi, i ovaj sistem može da iskoristi memoriju veoma brzo. Listing 5.10 implementira funkciju `fib()`.

{upozoftfiufjr Kada pokrenete listing 5.10, upotrebite mali broj (manji od 15). Posto on koristi rekurzija, može se potrošiti mnogo memorije.

Listing 5.10; Demonstrironje rekurzije korišćenjem Fibonacci - jeve serije

```
// Listing 5.10 - demonstrira rekurziju
//  Fibonacci-jevo pretrazivanje.
// Pronalazi n-ti Fibonacci - jev broj
// Koristi ovaj algoritam: Fib(n) = fib(n-l) + fib(n-2)
// Uslovi za zaustavljanje: n = 2 J| n = 1

#include<iostream.h>

int fib(int n);

int main()
{

  int n, answer;
  cout « "Unesite broj koji treba pronaci: ";
  cin » n;

  cout « "\n\n";

  answer = fib(n);

  cout « answer « · je " « n « "-ti Fibonacci - jev broj\n";
     return 0;


int fib (int n)
{
  cout « "U procesu fib(" « n « ")... ";

  if (n < 3 )
  {
     cout « "Vraca l!\n";
     return (1);
  }
  else
  {
    cout « "Poziva se fib(" « n-2 « ") i fib(" « n-1 « »).\n"
    return( fib(n-2) + fib(n-l));
```

```
Unesite broj koji treba pronaci:  5

U procesu fib(5)... Poziva se fib(3) i fib(4).
U procesu fib(3)... Poziva se fib(l) i fib(2).
```

```
U procesu fib(l)... Vraca 1!
U procesu fib(2)... Vraca 1!
U procesu fib(4)... Poziva se fib(2) i fib(3).
U procesu fib(2)... Vraca 1!
U procesu fib(3)... Poziva se fib(l) i fib(2).
U procesu fib(l)... Vraca 1!
U procesu fib(2)... Vraca 1!
5 je 5-ti Fibonacci-jev broj
```

jШШ&» Program traži broj za pronalaženje u liniji 15 i dodeljuje ga promenljivoj target. Onda poziva fib() sa target. Izvršenje se grana na funkciju fib(), gde, u liniji 28, ona štampa svoj argument.

Argument n se testira, da bi se videlo da li je jednak 1, ili 2 u liniji 30; ako je tako, vraća se iz fib(). Inače, ona vraća sume vrednosti, koje su vraćene pozivanjem fib() za n - 2 i n - 1.

U primeru n je 5, pa se iz mainQ poziva fib(5). Izvršenje "skače" na funkciju fib() i n se testira za vrednost manju od 3 u liniji 30. Test je neistinit, pa fib(5) vraća sumu vrednosti, koje su vraćene iz fib(3) i fib(4). To znači, da se fib() poziva za n - 2 (5 - 2 = 3) i n - 1(5 - 1 - 4), fib(4) će vratiti 3, a fib(3) će vratiti 2, pa će konačni odgovor biti 5.

Zato što fib(4) predaje argument koji nije manji od 3, fib() će se pozvati ponovo, ovoga puta sa 3 i 2, dok će fib(3) redom pozvati fib(2) i fib(l). Konačno, pozivi fib(2) i fib(l) će vratiti 1, jer su oni uslovi za zaustavljanje.

Izlaz prati ove pozive i povratne vrednosti. Kompajlirajte, povežite i izvršite ovaj program, unoseći prvo 1, onda 2, pa 3, sve do 6, i pažljivo posmatrajte izlaz. Onda, radi zabave, probajte broj 20. Ako ne ostanete bez memorije, imaćete dobar "sou".

Rekurzija se ne koristi često u C + + programiranju, ali može biti moćna i elegantna alatka za odredene potrebe.

^KAPOMENA^, Rekurzija je veoma težak deo naprednog programiranja. Ona je ovde predstavljeno, jer Vam to može pomod da razumete osnove njenog rada, ali nemojte previse brinuti ako ne razumete potpuno sve detalje.

Kako funkcije rade - pogled ispod haube

Kada pozovete funkciju, kod se grana na pozvanu funkciju, parametri se predaju, izvršava se telo funkcije. Kada se funkcija završi, vraća se vrednost (osim ako funkcija vraća void), i kontrola se vraća pozivajućoj funkciji.

Kako se ovaj zadatak ostvaruje? Kako kod zna gde da se grana? Gde se promenljive čuvaju prilikom predaje? Šta se dešava sa promenljivim koje su deklarisane u telu funkcije? Kako se vraća povratna vrednost? Kako kod zna gde da nastavi?

Većina uvodnih knjiga ne pokušava da odgovori na ova pitanja, ali, bez razumevanja ovih informacija, programiranje će za Vas ostati nejasna misterija. Objašnjenje zahteva kratak osvrt na diskusiju o kompjuterskoj memoriji.

Nivoi apstrakcije

Jedna od glavnih prepreka za nove programere je borba sa mnogim slojevima intelektualne apstrakcije. Kompjuteri su, naravno, samo elektronske mašine. Oni ne znaju ništa o prozorima i menijima, o programima i instrukcijama, čak ni o jedinicama i nulama. Sve što se stvarno dešava je to da se na različitim mestima meri napon u integrisanom kolu. Čak i ovo je apstrakcija: elektricitet je samo jedan intelektualan koncept, koji tumači ponašanje podatomskih delova.

Programera uznemirava bilo koji nivo detalja ispod ideje vrednosti u RAM-u. Ipak, nije potrebno da znate fiziku, da biste vozili auto, napravili tost, ili udarili lopticu za bejzbol - - isto tako, nije potrebno da znate elektroniku kompjutera, da biste ga programirali.

Ipak, potrebno je da znate kako je memorija organizovana. Bez jake mentalne slike o tome gde su Vaše promenljive kada se kreiraju i kako se vrednosti predaju medu funkcijama, to će sve ostati nedokučiva misterija.

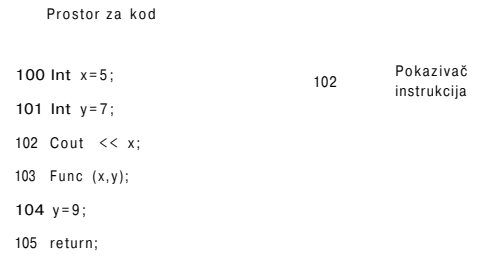
Particioniranje RAM-a

Kada započnete Vaš program, Vaš operativni sistem (kao što su DOS, ili Microsoft Windows) priprema različita područja memorije, bazirajući se na zahtevima Vašeg kompajlera. Kao C + + programer, često ćete biti zabrinuti prostorom globalnih imena, slobodnim skladištem, registrima, prostorom za kod i stekom.

Globalne promenljive su u prostoru globalnih imena. O prostoru globalnih promenljivih i slobodnom skladištu ćemo vise govoriti u narednim danima, a za sada ćemo fokusirati registre, prostor za kod i stek.

Registri su specijalno područje memorije koje je izgrađeno direktno u centralnoj procesnoj jedinici (engl. Central Processing Unit, CPU). Oni se brinu o internom domadnstvu. Mnogo onoga što se odvija u registrima izlazi iz opsega ove knjige, ali ono sto nas interesuje je skup registara koji su odgovorni za pokazivanje, u svakom trenutku, na sledeću liniju koda. Mi ćemo nazvati ove registre, zajedno, pokazivač instrukcija. Posao pokazivača instrukcija je da pamti koja sledeća linija koda treba da se izvrši.

Sam kod se nalazi u*prostoru za kod,* a to je deo memorije rezervisan za čuvanje binarnog oblika instrukcija, koje ste kreirali u Vašem programu. Svaka linija izvornog koda se prevodi u seriju instrukcija, a svaka od ovih instrukcija se nalazi na određenoj adresi u memoriji. Pokazivač instrukcija ima adresu sledeće instrukcije koju treba izvršiti. Slika 5.4 ilustruje ovu ideju.

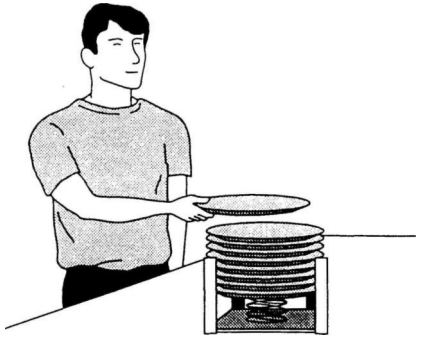


Slika 5.4.
Pokazivač instrukcija

Stek je posebno područje memorije, locirano za čuvanje podataka, koje zahtevaju sve funkcije u Vašem programu. Nazvan je tako, jer je to red sa osobinom zadnji-unutra, prvi-napolje, slično kao stek tanjira u kafeteriji, kao što je prikazano na Slici 5.5.

Zadnji-unutra, prvi-napolje znači da će ono što je zadnje dodato na stek biti prva stvar koja se uzima. Većina redova je kao red u pozorištu: prvi u redu je prvi koji ulazi. Stek je vise kao stek novčića: Vi poređate u nizu 10 penija na vrhu stola, a onda neke uzmete, zadnja tri će biti prva tri koja ćete uzeti.

Kada se podaci "gurnu" u stek, on raste; kako se podaci "vade" sa steka, on se skuplja. Nije moguće izvaditi tanjir iz steka bez vadenja svih tanjira koji su stavljeni posle njega.



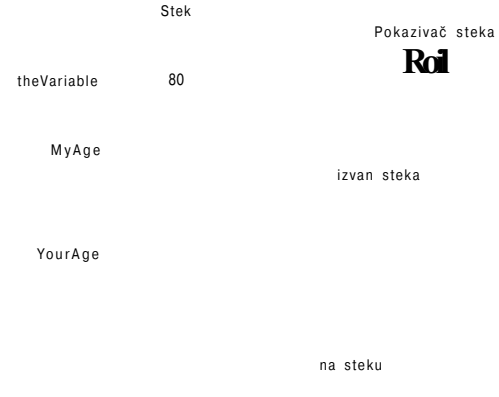
Slika 5.5.
Stek

Stek tanjira je opšta analogija. On je ispravan sve dok se podudara, ali je fundamentalno pogrešan. Tačnija mentalna slika je serija kockica, koje su poravnate od vrha do dna. Vrh steka je svaka kockica na koju pokazivač steka (kao još jedan registar) pokazuje.

Svaka kockica ima sekvencijalnu adresu, a jedna od tih adresa se čuva u registru koji je pokazivač steka. Sve ispod te magične adrese, koja je poznata kao vrh steka, smatra se da je na steku. Sve iznad vrha se smatra da nije na steku i da je nevažeće. Slika 5.6 ilustruje ovu ideju.

Slika 5.6.
Pokazivač steka

Kada se podatak stavlja na stek, on se stavlja u kockicu koja se nalazi iznad pokazivača steka, a onda se pokazivač pomera na nov podatak. Kada se podatak uzima sa steka, sve što se stvarno dešava je to da se adresa pokazivača menja pomeranjem na dole. Slika 5.7 razjašnjava ovo pravilo.



Slika 5.7.
Pomeranje pokazivača steka

Stek i funkcije

Evo sta se dešava kada se program, koji se izvršava na PC-u pod DOS-om, grana na funkciju:

1. Adresa u pokazivaču instrukcija se uvećava na sledeću instrukciju posle poziva funkcije. Ta adresa se onda stavlja na stek i ona će biti povratna adresa pri povratku iz funkcije.

2. Na steku se pravi prostor za povratni tip, koji ste deklarisali. Na sistemima sa dvo-bajtnom celobrojnom vrednošću, ako je povratni tip deklarisan sa int, dva bajta se dodaju na stek, ali se nikakva vrednost ne stavlja u ove bajtove.

3. Adresa pozvane funkcije, koja se čuva u specijalnom podrucju memorije, rezervisanom za tu svrhu, učitava se u pokazivač instrukcija, tako da će sledeća izvršena instrukcija biti u pozvanoj funkciji.

4. Tekući vrh steka se beleži i čuva u specijalnom pokazivaču, koji se naziva okvir steka. Sve što se od sada doda na stek do povratka iz funkcije će se smatrati "lokalnim" za funkciju.

5. Svi argumenti za funkciju se stavljaju na stek.

6. Sada se izvršava instrukcija u pokazivaču instrukcija, čime se izvršava prva instrukcija u funkciji.

7. Lokalne promenljive se stavljaju na stek čim se definišu.

Kada je funkcija spremna za povratak, povratna vrednost se stavlja u područje steka, koje je rezervisano u koraku 2. Stek se onda potpuno predaje okviru steka, što efektivno uništava sve lokalne promenljive i argumente za funkciju.

Povratna vrednost se uzima sa steka i dodeljuje kao vrednost poziva funkcije, a adresa, koja je sačuvana u koraku 1, vraća se i stavlja u pokazivač instrukcija, tako da se program nastavlja odmah posle poziva funkcije, sa vraćenom vrednošću funkcije.

Neki detalji ovog procesa se menjaju od kompajlera do kompajlera, ili izmedu kompjutera, ali esencijalne ideje su konzistentne među okruženjima. Uopšteno, kada pozovete funkciju, povratna adresa i parametri se stavljaju na stek. Tokom života funkcije, lokalne promenljive se dodaju na stek. Po povratku iz funkcije, one se ukljanjaju, preuzimanjem steka.

U narednim danima pogledaćemo druga mesta u memoriji za čuvanje podataka, koji moraju postojati i posle života funkcije.

Rezime

Funkcija je, u stvari, potprogram, kome možete predati parametre i iz kojeg možete vratiti vrednost. Svaki C + + program počinje u funkciji `main()`, a ona dalje može pozvati druge funkcije.

Funkcija se deklariše prototipom, koji opisuje povratnu vrednost, ime funkcije i tipove njenih parametara. Funkcija opciono može biti deklarisana kao inline. Prototip funkcije takode može deklarisati podrazumevane vrednosti za jedan, ili vise parametara.

Definicija funkcije mora odgovarati prototipu funkcije po povratnom tipu, imenu i listi parametara. Imena funkcija se mogu preklopiti menjanjem broja, ili tipa parametara; prevodilac pronalazi ispravnu funkciju, baziranjem na listu argumenata.

Lokalne promenljive funkcije i argumenti, koji su predati funkciji, su lokalni za blok u kome su deklarisani. Parametri predati po vrednosti su kopije i ne mogu uticati na vrednost promenljivih u pozivajućoj funkciji.

Pitanja i odgovori

P Zašto ne učiniti sve promenljive globalnim?

O Nekada se upravo tako i radilo. Ipak, kako su programi postajali kompleksniji, postalo je veoma teško pronaći bagove u programima jer je podatke mogla pokvariti svaka funkcija - globalni podaci mogu biti promenljeni bilo gde u programu. Godine iskustva su ubedile programere da bi podatke trebalo čuvati kao lokalne koliko, je to moguće, a pristup zbog promene tih podataka bi trebalo da bude usko definisan.

P Kada bi trebalo koristiti ključnu reč inl ine u prototipu funkcije?

O Ako je funkcija veoma mala, ne vise od linije, ili dve, a ako se neće pozivati sa mnogo mesta u Vašem programu, ona je kandidat za inline.

P Zašto se promene vrednosti argumenata funkcije ne reflektuju u pozivajućoj funkciji?

O Argumenti koji se predaju funkciji se predaju po vrednosti. To znači da je argument u funkciji, u stvari, kopija originalne vrednosti. Ovaj koncept je detaljno objašnjen u sekciji "Dodatno objašnjenje", koja sledi posle Radionice.

P Ako se argumenti predaju po vrednosti, šta da radim ako mi je potrebno da se promene reflektuju u pozivajućoj funkciji?

O U Danu 8, biće objašnjeni pokazivači. Upotreba pokazivača će rešiti ovaj problem, a takođe i obezbediti prevazilaženje ograničenja vraćanja samo jedne vrednosti iz funkcije.

P Šta se dešava ako imam sledeće dve funkcije?

```
int Area (int width, int length = 1); int Area (int size);
```

Da li će se one preklopiti? Postoji razlika u broju parametara, ali prvi ima vrednost koja se podrazumeva.

0 Deklaracije će se kompajlirati, ali, ako pozovete Area sa jednim parametrom, dobićete grešku u vreme kompajliranja: ambiguity between `Area (int, int)` i `Area (i nt)` - dvosmislenost između `Area (int, int)` i `Area (int)`.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje predenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Kakva je razlika izmedu prototipa i definicije funkcije?
2. Da li imena parametara moraju da se slažu u prototipu, definiciji i pozivu funkcije?
3. Ako funkcija ne vraća vrednost, kako deklarišete funkciju?
4. Ako ne deklarišete povratnu vrednost, koji tip te vrednosti se pretpostavlja?
5. Šta je lokalna promenljiva?
6. Šta je opseg?
7. Šta je rekurzija?
8. Kada bi trebalo koristiti globalne promenljive?
9. Šta je preklapanje funkcija?
10. Šta je polimorfizam?

Vežbe

1. Napišite prototip za funkciju nazvanu Perimeter (), koja vraća unsigned 1 ong i nt i prihvata dva parametra, oba unsigned short int.
2. Napišite definiciju funkcije Peri meter (), kao što je objašnjeno u Vežbi 1. Dva parametra predstavljaju dužinu i širinu pravougaonika. Neka funkcija vrati obim (dvostruka dužina plus dvostruka širina).
3. ISTERIVAČ BAGOVA: Šta nije u redu sa funkcijom u sledećem kodu?

```
#include <iostream.h>
void myFunc(unsigned short int x);
int main()
{
unsigned short int x, y;
y = myFunc(int);
cout « "x: " « x « " y " « y « "\n"-
)
```



```
void myFunc(unsigned short int x)                                    mmmmw
{
return (4*x);
}
```

4. ISTERIVAČ BAGOVA: Šta nije u redu sa funkcijom u sledećem kodu?

```
#include <iostream.h>
int myFunc(unsigned short int x);
int main()
{
unsigned short int x, y;
y = myFunc(x);
cout « "x: " « x « " y: " « y « "\n";
}

int myFunc(unsigned short int x);
{
return (4*x);
}
```

5. Napišite funkciju koja prihvata dva unsigned short celobrojna argumenta i vraća rezultat deljenja prvog i drugog. Nemojte obaviti deljenje, ako je drugi broj nula, nego vratite -1.
6. Napišite program koji od korisnika traži dva broja i poziva funkciju koju ste napisali u Vežbi 5. Odštampajte odgovor, ili odštampajte poruku o grešci, ako dobijete -1.
7. Napišite program koji traži broj i stepen. Napišite rekurzivnu funkciju, koja diže broj na stepen. Tako, ako je broj 2, a stepen 4, funkcija će vratiti 16.

Dan 6

Osnovne klase

Klase proširuju ugradene sposobnosti C + + , koje Vam pomažu u predstavljanju i rešavanju kompleksnih problema iz stvarnog sveta. Danas ćete naučiti:

- šta su klase i objekti
- kako definisati novu klasu i kreirati objekte te klase
- šta su funkcije članice i podaci članovi
- šta su konstruktori i kako se koriste.

Kreiranje novih tipova

Već ste naučili nekoliko tipova promenljivih, uključujući i neoznačene celobrojne tipove i karaktere. Tip promenljive Vam govori puno o njoj. Na primer, ako deklarišete `Height` i `Width` kao neoznačene celobrojne tipove, znate da svaki od njih može čuvati broj između 0 i 65.535, pretpostavljajući da celobrojna vrednost ima dva bajta. To je objašnjenje šo se oni nazivaju neoznačeni celobrojni tipovi. Pokušaj da se čuva bilo šta drugo u ovim promenljivim prouzrokuje grešku. Ne možete čuvati Vaše ime u neoznačenoj kratkoj celobrojnoj promenljivoj; i nemojte ni pokušavati.

Samo deklarisanjem ovih promenljivih kao neoznačenih kratkih celobrojnih tipova, moguće je da `Height` dodate promenljivoj `Width` i da dodelite taj broj drugom broju.

Tip ovih promenljivih Vam govori

- njihovu veličinu u memoriji
- kakve informacije one mogu čuvati
- koje akcije se nad njima mogu preduzeti

Uopšteno govoreći, tip je kategorija. Poznati tipovi uključuju automobil, kuću, osobu, voce i oblik. U C++-U, programer može kreirati bilo koji potreban tip i svaki od ovih novih tipova može imati svu funkcionalnost i snagu ugradenih tipova.

Zašto kreirati novi tip?

Programi se, obično, pišu da bi se rešili problemi iz stvarnog sveta, kao što je čuvanje zapisa o zaposlenima, ili simuliranje rada grejnog sistema. Iako je moguće rešiti kompleksne probleme, korišćenjem programa koji su napisani samo sa celobrojnim promenljivim i karakterima, daleko je lakše boriti se sa velikim, kompleksnim problemima, ako možete kreirati reprezentacije objekata o kojima govorite. Drugim rečima, simuliranje rada grejnog sistema je lakše ako možete kreirati promenljive koje predstavljaju sobe, grejne senzore, termostate i bojlere. Što ove promenljive bliže odgovaraju realnosti, to je lakše napisati program.

Klase i članovi

Novi tip pravite deklarisanjem klase. Klasa je samo kolekcija promenljivih, često različitih tipova, koji su kombinovani sa skupom povezanih funkcija.

Jedan način da se razmišlja o automobilu je kao o točkovima, vratima, sedištima, prozorima i tako dalje. Drugi način je da razmišljate o tome šta automobil može da uradi: da se pomera, ubrza, uspori, stane, parkira i tako dalje. Klasa Vam omogućava da enkapsulišete (zamotate) ove različite delove i različite funkcije u kolekciju, koja se naziva objekat.

Enkapsuliranje svega što znate o automobilu u jednu klasu ima nekoliko prednosti za programera. Sve se nalazi na jednom mestu, što olakšava upućivanje na njega, kopiranje i manipulisanje podacima. Isto tako, klijenti Vaše klase, to jest, delovi programa koji koriste Vašu klasu, mogu koristiti Vaš objekat, a da se, pri torn, ne brinu šta se nalazi u njemu, ili kako on radi.

Klasa se može sastojati od bilo koje kombinacije tipova promenljivih i tipova drugih klasa. Na promenljive u klasi se upućuje sa promenljive članice, ili podataka članova. Klasa Car bi mogla imati promenljive članice koje predstavljaju sedišta, tip radija, gume i tako dalje.

Promenljive članice, takođe poznate kao podaci članovi, su promenljive u Vašoj klasi. Promenljive članice su deo Vaše klase, kao što su točkovi i motor deo Vašeg automobila.

Funkcije u klasi, obično, manipulišu promenljivim članicama. Na njih se upućuje sa funkcije članice, ili metode klase. Metode klase Car bi mogle uključiti Start() i Brake() (engl. start = kreni, brake = zakoči). Klasa Cat (engl. cat = mačka) bi mogla imati podatke članove koji predstavljaju starost i težinu; njene metode bi mogle uključiti SleepO, Meow() i ChaseMice() (engl. sleep = spavaj, meow = mijau, chase mice = hvataj miševe).

Funkcije članice, takode poznate kao metode, su funkcije u Vašoj klasi. Funkcije članice su deo Vaše klase isto koliko i promenljive članice. One određuju šta objekti Vaše klase mogu uraditi.

Deklarisanje klase

Da biste deklarisali klasu, upotrebite ključnu reč class, praćenu otvarajućom zagradom, a onda dolazi lista podataka članova i metoda klase. Završite deklaraciju zatvarajućom zagradom i znakom tačka-zarez. Evo deklaracije klase koja je nazvana Cat:

```
class Cat
{
unsigned int   itsAge;
unsigned int   itsWeight;
Meow();
};
```

Deklarisanje ove klase ne alocira memoriju za Cat. Ono samo saopštava kompajleru šta je to Cat, koje podatke sadrži (itsAge i itsWeight) i šta ova deklaracija može da uradi (Meow()). Ono takođe saopštava kompajleru koliko je Cat velika, to jest, koliko prostora kompajler mora rezervisati za svaku Cat koju kreirate. U ovom primeru, ako celobrojna vrednost ima dva bajta, Cat ima samo četiri bajta: itsAge ima dva bajta, a itsWeight ima druga dva bajta. Meow() ne zauzima prostor, jer se nikakav skladišteni prostor ne rezerviše za funkcije članice (metode).

Reč o konvencijama imenovanja

Kao programer, morate imenovati sve svoje promenljive članice, funkcije članice i klase. Kao što ste naučili u Danu 3, "Promenljive i konstante", ovo bi trebalo da budu Iako razumljiva imena, koja imaju neko značenje. Cat, Rectangle i Employee su dobra imena klasa. Meow(), ChaseMiceQ i StopEngine() su dobra imena funkcija, jer Vam govore šta funkcije rade. Mnogi programeri imenuju promenljive članice prefiksom its, kao u i tsAge, i tsWeigt i i tsSpeed. Ovo pomaže u razlikovanju promenljivih članica od promenljivih nečlanica.

C++ razlikuje mala i velika slova, pa bi imena svih klasa trebalo da prate isti šablon. Na ovaj način, nikada neće biti potrebno da proverite kako da napišete ime Vaše klase: da li Rectangle, rectangle ili RECTANGLE? Neki programeri vole da stave odredeno slovo kao prefiks imenu svake klase; na primer, cCat ili cPerson. Drugi,

pak, koriste u imenu samo velika, ili mala slova. Konvencija koju ja koristim je da imenujem sve klase rx>četnim velikim slovom, kao u Cat i Person.

Slično, mnogi programeri započinju sve funkcije velikim, a sve promenljive malim slovima. Reči se, obično, rastavljaju podvlakom, kao u ChaseMice, ili započinjanjem svake reči velikim slovom, na primer, ChaseMice, ili DrawCircle.

Važna ideja je da bi trebalo da izaberete jedan stil i da ostanete njemu dosledni kroz svaki program. Vremenom, Vaš stil će evoluirati i uključiti ne samo konvencije imenovanja, nego i uvlake, poravnanje zagrada i stil komentarisanja.

^IAPOMEHA^ Uobičojeno je za razvojne kompanije da imaju kućne standarde za mnoge osobine stila. Ovo osigurava da svi oni koji se have razvojem mogu lako da čitaju kod svake firme.

Definisanje objekta

Objekat Vašeg novog tipa defmišete kao što definišete celobrojnu promenljivu:

```
unsigned int GrossWeight;          // definise neoznacenu celobrojnu promenljivu
Cat Frisky;                        // definise Cat
```

Ovaj kod definise promenljivu nazvanu GrossWight, čiji tip je neoznačena celobrojna vrednost. On, takode, definise objekat Frisky, čija je klasa (ili tip) Cat.

Klase protiv objekata

Defmicija mačke nije nikada Vaš ljubimac: individualne mačke su Vaši ljubimci. Vi podvlačite razliku između mačke, kao ideje, i određene mačke, koja se upravo sada seta po Vašoj sobi. Na isti način, C+ + pravi razliku izmedu klase Cat, koja je ideja mačke, i svakog individualnog Cat objekta. Frisky je objekat tipa Cat na isti način na koji je GrossWeight promenljiva tipa unsigned int.

.^JLIt!Ž3 *Ofy'kat* je individualni primerak klase.

Pristup članovima klase

Pošto definišete stvarni Cat objekat, na primer, Frisky, operator tačka (.) koristite da biste pristupili članovima tog objekta. Zato, da biste dodelili 50 promenljivoj `Weight`, koja je članica objekta Frisky, napisali biste

```
Frisky.Weight = 50;
```

Na isti način, da biste pozvali funkciju `MeowQ`, napisali biste

```
Frisky.Meow();
```

Kada koristite metodu klase, Vi je pozovete. U ovom primeru, Vi pozivate `MeowQ` za Frisky.

Dodeljivanje objektima, a ne klasama

U C+ +-u Vi ne dodeljujete vrednosti tipovima, već promenljivim. Na primer, nikada ne biste napisali

```
int = 5;                  // pogresno
```

Kompjuter bi označio ovo kao grešku, jer ne možete dodeliti **5** celobrojnom tipu. Umesto toga, morate definisati celobrojnu promenljivu i dodeliti **5** toj promenljivoj. Na primer,

```
int  x;                   // definise x kao int
x = 5;                    // postavlja vrednost u x na 5
```

Ovo je simbolički prikaz rečenice, "Dodeli **5** promenljivoj x, koja je tipa int". Isto tako, Vi ne biste napisali

```
Cat.age=5;                // pogresno
???
```

Kompajler bi označio ovo kao grešku, jer ne možete dodeliti **5** delu Cat koji predstavlja starost. Umesto toga, morate definisati Cat objekat i dodeliti **5** torn objektu. Na primer,

```
Cat  Frisky;              // isto kao i  int x;
Frisky.age = 5;           // isto kao i  x = 5;
```

Ako ne deklarišete objekat, Vaša klasa ga neće imati

Probajte ovaj eksperiment: trogodišnjem detetu pokažite mačku. Onda recite: "Ovo je Frisky, koji zna jedan trik. Frisky laje." Dete će se nasmešiti i red: "Ne, šašavi, mačke ne mogu da laju."

Kada biste napisali

```
Cat   Frisky;             // napravi Cat nazvanu Frisky
Frisky.Bark()             // reci Friskyju da laje
```

kompajler bi rekao: "Ne, šašavi, mačke ne mogu da laju". Kompajler zna da Frisky ne može da laje jer, klasa Cat nema funkciju BarkQ. Kompajler čak ne bi dozvolio Fri skyju da mijauče ako ne biste imali funkciju Meow().

<| Wtfirc ^ Upotrebite kljucnu rec class, da biste deklarisali klasu.

Nemojte pomešati deklaraciju sa definicijom. Deklaracija govori šta je klasa, a definicija rezerviše memoriju za objekat.

Nemojte pomešati klasu sa objektom.

Nemojte dodeliti vrednosti klasi. Dodelite vrednosti podacima clanovima objekta.

Upotrebite operator tačka (.) za pristup clanovima i funkcijama klase.

Privatno protiv javnog

U deklaraciji klase se koriste i druge ključne reči. Dve od najvažnijih su publ i c (engl. public = javni) i private (engl. private = privatni).

Podrazumevano je da su privatni svi članovi klase, podaci i metodi. Privatnim clanovima se može pristupiti samo unutar metoda same klase. Javnim clanovima se može pristupiti kroz svaki objekat klase. Razlika je i važna i zbunjujuća. Da bismo je učinili malo jasnijom, razmotrite primer od ranije iz ovog poglavlja:

```
class Cat
{
unsigned int  itsAge;
unsigned int  itsWeight;
Meow();
};
```

U ovoj deklaraciji itsAge, itsWeight i Meow() su privatni, zato što su svi članovi klase podrazumevano privatni. Ovo znači da su oni privatni, osim ako ne odredite drugačije.

Ipak, ako napišete

```
Cat  Boots;
Boots.itsAge=5;       // greska! ne mozete prostupiti privatnim podacima!
```

kompajler ovo označava kao grešku. U stvari, rekli ste prevodiocu: "Ja ću pristupati clanovima itsAge, itsWeight i MeowQ, samo iz funkcija članica klase Cat." Ali ovde ste pristupili promenljivoj članici i tsAge objekta Boots izvan metode klase Cat. Samo zato što je Boots objekat klase Cat, to ne znači da možete pristupiti delovima objekta Boots koji su privatni.

Ovo je izvor beskonačne konfuzije za nove C + + programere. Ja skoro da mogu da čujem kako "kukate": "Hej! Ja sam malopre rekao da je Boots mačka. Zašto Boots ne može pristupiti svojoj starosti?" Odgovor je: Boots može, ali Vi ne možete. Boots, u sopstvenim metodima, može pristupiti svim svojim delovima, javnim i privatnim. Čak i pošto ste kreirali klasu Cat, to ne znači da možete videti, ili promeniti njene delove, koji su privatni.

Način korišćenja Cat tako da možete pristupiti podacima clanovima je

```
class Cat
I
public:
unsigned int  itsAge;
unsigned int  itsWeight;
Meow();
};
```

Sadasu itsAge, itsWeight i Meow() javni. Boots.itsAge = 5 se prevodi bez problema.

Listing 6.1 prikazuje deklaraciju klase Cat sa javnim promenljivim članicama. "^*[81]!^

Listing 6.1: Pristup javnim clanovima jednosfavne klase

```
// Demonstrira deklaraciju klase i
// definiciju objekta klase,

#include <iostream.h>   // za cout

class Cat               // deklarisi objekat klase
{
 public:                // clanovi koji slede su javni
   int  itsAge;
   int  itsWeight;



void main()
{
   Cat Frisky;
   Frisky.itsAge = 5;    // dodeli promenljivoj clanici
   cout « "Frisky je macka koja ima " ;
   cout « Frisky.itsAge « " godina.\n";
```

```
Frisky je macka koja ima 5 godina.
```

*J^^jgf^* Linija 6 sadrži ključnu reč class. Ovo govori kompajleru da je ono što sledi deklaracija. Ime nove klase dolazi posle ključne reči class. U ovom slučaju, to je Cat.

Telo deklaracije počinje otvarajućom zagradom u liniji 7, a završava se zatvarajućom zagradom i znakom tačka-zarez u liniji 11. Linija 8 sadrži ključnu reč public, što pokazuje da je sve što sledi javno, do reči private, ili kraja deklaracije klase.

Linije 9 i 10 sadrže deklaracije članova klase itsAge i itsWeight.

Linija 14 započinje glavnu funkciju programa. Fri sky je definisan u liniji 16 kao primerak klase Cat, to jest, kao Cat objekat. Friskyjeva starost se postavlja u liniji 17. U linijama 18 i 19 promenljiva članica itsAge se koristi za štampanje poruke o Friskyju.

MAPOMEHAy Probajte da pretvorite u komentar liniju 8 i probajte da ponovo kompajlirate. Dobićete grešku u liniji 17, jer i tsAge vise neće imati javni pristup. Za klase se podrazumeva privatni pristup.

Učinite podatke članove privatnim

Kao opšte pravilo dizajna, trebalo bi da čuvate podatke članove klase kao privatne. Zato, morate kreirati javne funkcije, poznate kao metode pristupa za postavljanje i uzimanje privatnih promenljivih. Ovi metodi pristupa su funkcije članice, koje drugi delovi Vašeg programa pozivaju da bi dobili i postavili Vaše privatne promenljive.

^IIIIIIIIII *Javni metod* pristupa je funkcija članica klase, koja se koristi ili za čitanje vrednosti privatne promenljive, ili za postavljanje njene vrednosti.

Zašto dosadivati ovim dodatnim nivoom indirektnog pristupa? Ipak, jednostavnije je i lakše koristiti podatke, nego raditi kroz funkcije pristupa.

Funkcije pristupa Vam omogućavaju da odvojite detalje o tome kako se podaci čuvaju od toga kako se koriste. Ovo Vam omogućava da promenite način čuvanja podataka bez potrebe za ponovnim pisanjem funkcija koje koriste podatke.

Ako funkcija koja treba da zna starost Cat pristupa promenljivoj itsAge direktno, biće potrebno tu funkciju ponovo napisati, ako Vi, kao autor klase Cat, odlučite da promenite način kako se taj podatak čuva. Posedovanjem poziva funkcije GetAge(), Vaša klasa Cat može Iako vratiti pravu vrednost, bez obzira kako ste došli do starosti. Pozivajuća funkcija ne treba da zna da li je čuvate kao neoznačenu celobrojnu vrednost, ili kao long, ili je proračunavate.

Ova tehnika olakšava održavanje Vašeg programa. Daje Vašem kodu duži život, jer promene dizajna ne prouzrokuju zastarevanje vašeg programa.

Listing 6.2 prikazuje klasu Cat, koja je modifikovana tako da uključuje privatne podatke članove i javne metode pristupa. Uočite da ovo nije izvršiv listing.

Listing 6.2: Klasa sa metodama pristupa

```
1:       // Deklaracija klase Cat
2:       // podaci clanovi su privatni, javni metodi pristupa
3:       // posreduju u postavljanju i uzimanju vrednosti privatnih podataka
4:
5:  class Cat
6:  {
7:   public:
8:       // javni metodi pristupa
9:     unsigned int GetAgeQ;
10:    void SetAge(unsigned int Age);
11:
12:    unsigned int GetWeightQ;
13:    void SetWeight(unsigned int Weight);
14:
15:      // javne funkcije clanice
16:    Meow();
17:
18:      // privatni podaci clanovi
19: private:
20:    unsigned int  itsAge;
21:    unsigned int  itsWeight;
22:
23: };
```


Ova klasa ima pet javnih metoda. Linije 9 i 10 sadrže metode pristupa za itsAge. Linije 12 i 13 sadrže metode pristupa za itsWeight. Ove funkcije pristupa postavljaju promenljive članice i vraćaju njihove vrednosti.

Javna funkcija članica MeowQ je deklarisana u liniji 16. MeowQ nije funkcija pristupa. Ona ne daje i ne postavlja promenljivu članicu; obavlja drugu službu za klasu - štampanje reči Meow.

Same promenljive članice su deklarisane u linijama 20 i 21.

Da biste postavili Friskyjevu starost, predali biste vrednost metodu SetAgeQ, kao u

```
Cat  Frisky;
Frisky.SetAge(5);    // postavi Friskyjevu starost, koriscenjem javnog metoda pristupa
```

Privatnost protiv bezbednosti

Deklarisanje metoda, ili podataka kao privatnih omogućava kompajleru da pronade programske greške, pre nego što one postanu bagovi. Svaki programer, koji ceni svoje konsultacione troškove, može zaobići privatnost, ako želi. Stroustrup, pronalazač C++-a, je rekao: "Mehanizam kontrole pristupa u C++-u obezbeduje zaštitu protiv neželjenog pristupa, ne protiv prevare." (ARM, 1990)

Ključna rec class

Sintaksa za ključnu reč class je sledeća:

```
class  class_name
{
Ij ovde dolaze kljucne reci za kontrolu pristupa
// ovde dolaze deklarisane promenljive i deklarisani metode klase
```

Ključnu reč class koristite da biste deklarisali nove tipove. Klasa je kolekcija podataka članova klase, koji mogu biti različitih tipova, uključujući i druge klase. Klasa, takođe, sadrži funkcije klase, ili metode - one su funkcije koje se koriste za manipulisanje podacima u klasi i za izvršavanje drugih usluga za klasu.

Vi definišete objekte novog tipa na isti način na koji definišete neku promenljivu. Navedete tip (klasu), a onda ime promenljive (objekat). Clanovima klase i funkcijama pristupate korišćenjem operatora tačka (.).

Ključne reči za kontrolu pristupa koristite da biste deklarisali sekcije klase kao javne, ili privatne. Podrazumevano za kontrolu pristupa je privatno. Svaka ključna reč menja kontrolu pristupa od te tačke do kraja klase, ili do sledeće ključne reči kontrole pristupa. Deklaracije klasa se završavaju zatvarajućom zagradom i znakom tačka-zarez.

Primer 1

```
class Cat
{
public:
unsigned int Age;
unsigned int Weight;
void MeowQ;
};

Cat Frisky;
Frisky.Age = 8;
Frisky.Weight = 18;
Frisky.MeowQ;
```

Primer 2

```
class Car
{
public:                              // sledecih pet su javni

void Start();
void Accelerate();
void BrakeO;
void SetYear(int year);
int GetYear();

private:                             // ostatak je privatni

int Year;
Char Model {255};
};                                   // kraj deklaracije klase

Car OldFaithful;                     // napravi primerak automobila
int bought;                          // lokalna promenljiva tipa int
OldFaithful.SetYear(84) ;            // godini dodeli 84
bought = OldFaithful.GetYear();      // postavi bought na 84
OldFaithful.Start();                 // pozovi metodu za startovanje
```

j PAZITI Deklarišite promenljive članice kao privatne.

Upotrebite javne metode pristupa.

Nemojte pokušati da koristite privatne promenljive članice izvan klase.

Pristupite privatnim promenljivim članicama iz funkcija članica klase.

Implementiranje metoda klase

Kao što ste videli, funkcija pristupa obezbeduje javni interfejs za privatne podatke članove klase. Svaka funkcija pristupa, zajedno sa svakom drugom metodom klase

koju deklarišete, mora imati implementaciju. Implementacija se naziva definicija funkcije. Definicija funkcije članice počinje imenom klase, posle čega slede dvostruke dve tačke, ime funkcije i njeni parametri. Listing 6.3 prikazuje kompletnu deklaraciju jednostavne klase Cat i implementaciju njene funkcije pristupa i jedne opšte funkcije članice klase.

Listing 6.3: Implementiranje metoda jednostavne klase

```
1  // Demonstrira deklaraciju klase i
2  // definiciju metoda klase,
3
4  linclude <iostream.h>      // za cout
5
6  class Cat                  // zapocinje deklaraciju klase
7  {
8    public:                  // zapocinje javnu sekciju
9      int GetAge();          // funkcija pristupa
10     void SetAge (int age); // funkcija pristupa
11     void Meow();           // opsta funkcija
12   private:                 // pocinje privatnu sekciju
13     int itsAge;            // promenljiva clanica
14
15
16 // GetAge, Javna funkcija pristupa
17 // vraca vrednost clana itsAge
18 int Cat:-.GetAge()
19 {
20    return itsAge;
21
22
23 // definicija SetAge, javna
24 // funkcija pristupa
25 // postavlja clan itsAge
26 void Cat::SetAge(int age)
27
28    // postavi promenljivu clanicunjegovu starost na
29    // vrednost koja je predata u parametru age
30    itsAge = age;
31 }
32
33 // definicija metode Meow
34: // vraca void
35; // parametri: Nema
36 // akcija: Na ekranu stampa "mijau"
37 void Cat::Meow()
38 {
39    cout « "Mijau.\n";
40
41
```

nastavljase

Listing 6.3: Implementiranje metoda jednostavne klase *nastavak*

```
// kreira macku, postavlja njenu starost, pravi
// mijauk, saopstava nam njenu starost, a onda ponovo pravi mijauk.
int main()
{
   Cat Frisky;
   Frisky.SetAge(5);
   Frisky.MeowQ;
   cout « "Frisky je macka koja ima " ;
   cout « Frisky.GetAgeQ « " godinaAn";
   Fri sky. MeowQ;
    return 0;
```

```
Mi jau.
Frisky je macka koja ima 5 godina.
Mi jau.
```

Linije 6-14 sadrže definiciju klasne Cat. Linija 8 sadrži ključnu reč publ ic, što govori kompajleru da je ono što sledi skup javnih članova. Linija 9 ima deklaraciju javnog metoda pristupa GetAge(). GetAgeQ obezbeduje pristup privatnoj promenljivoj članici itsAge, koja je deklarisana u liniji 13. Linija 10 ima javnu funkciju pristupa SetAge Q. SetAge Q prihvata celobrojnu vrednost kao argument i postavlja itsAge na vrednost tog argumenta.

Linija 11 ima deklaraciju klasnog metoda MeowQ. MeowQ nije funkcija pristupa. Ovde je opšti metodi koji na ekranu štampa reč "Mijau".

Linija 12 zapocinje privatnu sekciju, koja uključuje samo deklaraciju u liniji 13 privatne promenljive članice itsAge. Deklaracija klase se završava zatvarajućom zagradom i znakom tačka-zarez u liniji 14.

Linije 18-21 sadrže definiciju funkcije članice GetAgeQ. Ovaj metod ne prihvata parametre: on vraća celobrojnu vrednost. Uočite da metodi klase uključuju ime klase, praćeno dvostrukom dvotačkom i imenom funkcije (Linija 18). Ova sintaksa saopstava kompajleru da je funkcija GetAgeQ, koju ovde definišete, ona koju ste deklarisali u klasi Cat. Izuzev ovog zaglavlja, funkcija GetAgeQ se kreira kao i bilo koja druga funkcija.

Funkcija GetAgeQ zauzima samo jednu liniju; ona vraća vrednost u itsAge. Uočite da funkcija mainQ ne može pristupiti promenljivoj itsAge, jer je itsAge privatna za klasu Cat. Funkcija mainQ ima pristup javnoj metodi GetAgeQ. Zato što je GetAge funkcija članica klase Cat, ona ima potpuni pristup promenljivoj itsAge. Ovaj pristup omogućava funkciji GetAgeQ da vrati vrednost od itsAge funkciji mainQ.

Linija 26 sadrži definiciju funkcije članice SetAge (). Ona prihvata celobrojni parametar i postavlja vrednost promenljive i tsAge na vrednost tog parametra u liniji 30. Zato što je članica klase Cat, SetAgeQ ima direktan pristup promenljivoj članici itsAge.

Linija 37 zapocinje definiciju, ili implementaciju metoda MeowQ klase Cat. To je jednolinijska funkcija, koja na ekranu štampa reč "Mijau", praćenu novom linijom.

f Zapamtite da karakter \n štampa novu liniju na ekranu.

Linija 44 zapocinje telo programa poznatom funkcijom mainQ . U ovom slučaju, ona ne prihvata argumente. U liniji 46 mainQ deklariše Cat, nazvanu Frisky. U liniji 47 vrednost 5 se dodeljuje promenljivoj članici itsAge, korišćenjem metoda pristupa SetAge(). Uočite da se metod poziva korišćenjem imena objekta (Frisky), za kojim slede operator člana (.) i ime metoda (SetAge()). Na isti način možete pozvati bilo koji drugi metod u klasi.

Linija 48 poziva funkciju članicu MeowQ, a linija 49 štampa poruku, koristeći metod pristupa GetAgeQ. Linija 51 poziva ponovo MeowQ .

Konstruktori i destruktori

Postoje dva načina za definisanje celobrojne promenljive. Vi možete definisati promenljivu, a onda joj, kasnije, u programu dodeliti vrednost. Na primer,

```
int Weight;          // definise promenljivu
                     // ovde dolazi ostali kod
Weight = 7;          // dodeli joj vrednost
```

Ili možete definisati celobrojnu promenljivu i odmah je inicijalizovati. Na primer,

```
int Weight = 7;      // definise, i inicijalizuje na 7
```

Inicijalizacija kombinuje definiciju promenljive sa njenim inicijalnim argumentom. Ništa Vas ne sprečava da kasnije promenite tu vrednost. Inicijalizacija osigurava da Vaša promenljiva uvek ima vrednost koja ima značenje.

Kako da inicijalizujete podatke članove klase? Klase imaju specijalnu funkciju članicu, koja je nazvana konstruktor. Konstruktor može prihvatiti parametre, prema potrebi, ali ne može imati povratnu vrednost, čak ni voi d. On je metod klase sa istim imenom kao i sama klasa.

Uvek kada deklarišete konstruktor, takođe ćete želeti da deklarišete destruktor. Kao što konstruktori kreiraju i inicijalizuju objekte Vase klase, destruktori čiste za Vašim objektom i oslobadaju memoriju, koju ste možda alocirali. Destruktor uvek ima ime klase, kome prethodi tilda (-). Destruktori ne prihvataju argumente i nemaju povratnu vrednost. Zato, deklaracija klase Cat uključuje

```
-Cat();
```

Podrazumevani konstruktori i destruktori

Ako ne deklarišete konstruktor, ili destruktor, kompajler će napraviti jednog od njih za Vas. Podrazumevani konstruktor i destruktor ne prihvataju argumente i ne rade ništa.

Koliko je koristan konstruktor koji ne radi ništa? Delimično, to je stvar forme. Svi objekti moraju biti konstruisani i dekonstruisani, pa se ove funkcije, koje ne rade ništa, pozivaju u odgovarajuće vreme. Ipak, da biste deklarisali objekat bez predavanja parametara, kao

```
Cat Rags;           // Rags ne dobija ni jedan parametar
```

morate imati konstruktor u obliku

```
Cat();
```

Kada defmišete objekat klase, poziva se konstruktor. Kada bi konstruktor klase Cat prihvatao dva parametra, Vi biste, možda, definisali Cat objekat pisanjem

```
Cat Frisky (5,7);
```

Kada bi konstruktor prihvatao jedan parametar, pisali biste

```
Cat Frisky (3);
```

U slučaju kada konstruktor ne prihvata parametre uopšte, izostavljate zagrade i pišete

```
Cat Frisky;
```

Ovo je izuzetak od pravila, koje zahteva da sve funkcije imaju zagrade, čak i ako ne privataju ni jedan parametar. Ovo je razlog zbog koga ste u mogućnosti da napišete

```
Cat Frisky;
```

što je poziv podrazumevanog konstruktora. On ne obezbeduje zagrade i izostavlja ih. Ne morate koristiti podrazumevani konstruktor kojeg je obezbedio kompajler. Uvek imate slobodu da napišete sopstveni konstruktor bez parametara. Čak i konstruktori bez parametara mogu imati funkcijsko telo, u kome inicijalizuju svoje objekte, ili obavljaju druge poslove.

Ako deklarišete konstruktor, deklarišite i destruktor, čak i ako Vaš destruktor ne radi ništa. Iako bi podrazumevani destruktor radio ispravno, neće biti suvišno ako deklarišete sopstveni. To čini Vas kod jasnijim.

Listing 6.4 prepisuje klasu Cat, zbog upotrebe konstruktora za inicijalizaciju Cat objekta, postavljajući svoju starost na svaku inicijalnu starost koju obezbedite, i demonstrira gde se poziva destruktor.

Listing 6.4: Korišćenje konstruktora i destruktora.

```
// Demonstrira deklaraciju konstruktora i
// destruktora za klasu Cat

linclude <iostream.h>       // za cout

class Cat                   // zapocinje deklaraciju klase
{
 public:                    // zapocinje javnu sekciju
   Cat(int initialAge);     // konstruktor
   -Cat();                  // destruktor
   int Get Age ();          // funkcija pristupa
   void SetAge(int age);    // funkcija pristupa
   void Meow();
 private:                   // zapocinje privatnu sekciju
   int itsAge;              // promenljiva clanica
};

// konstruktor klase Cat,
Cat::Cat(int initialAge)
{
   itsAge = initialAge;


Cat::-Cat()                 // destruktor, ne radi nista
{
}

// GetAge, Javna funkcija pristupa
// vraca vrednost clana itsAge
int Cat::GetAge()
{
   return itsAge;


// Definicija za SetAge, javna
// funkcija pristupa

void Cat::SetAge(int age)
{
   // postavi promenljivu clanicu njenu starost na
   // vrednost koja je predana u parametru age
   itsAge = age;
}

// definicija metode Meow
// vraca: void
// parametri: Nema
// akcija: Na ekranu stampa "mijau"
void Cat::Meow()

   cout « "Mijau.\n";
}

// kreira macku, postavlja njenu starost, pravi
// mijau, saopstava nam njenu starost, a onda ponovo pravi mijauk.
int main()

```

nastavljase

Listing 6.4: Korišćenje konstruktora i destruktora. ***nastavak***

```
    Cat Frisky(5);
    Frisky.Meow();
    cout « "Frisky je macka koja ima • ;
    cout « Frisky.GetAgeO « " godina.\n";
    Frisky.Meow();
    Frisky.SetAge(7);
    cout « "Sada Frisky ima " ;
    cout « Frisky.GetAgeO « " godina.\n";
    return 0;
```

```
Mijau.
Frisky je macka koja ima 5 godina.
Mijau.
Sada Frisky ima 7 godina.
```

Listing 6-4 je sličan listingu 6.3, osim što linija 9 dodaje konstruktor za prihvatatanje celobrojne vrednosti. Linija 10 deklariše destruktor, koji ne prihvata ni jedan parametar. Destruktori nikada ne prihvataju parametre, a ni konstruktori ni destruktori ne vraćaju vrednost, čak ni void.

Linije 19-22 prikazuje implementaciju konstruktora. Ona je slična implementaciji funkcije pristupa SetAge(), gde nema povratne vrednosti.

Linije 24-26 prikazuju implementaciju destruktora ~Cat(). Ova funkcija ne radi ništa, ali morate uključiti definiciju funkcije, ako je označite u deklaraciji klase.

Linija 58 sadrži definiciju Cat objekta, Frisky. Vrednost 5 se predaje Friskyjevom konstruktoru. Nema potrebe da se pozove SetAge **0,** jer je Frisky kreiran sa vrednošću 5 u svojoj promenljivoj članici itsAge, kao što je prikazano u liniji 61. U liniji 63 promenljivoj i tsAge objekta Frisky se dodeljuje 7. Linija 65 štampa novu vrednost.

PANT| Koristite konstruktore za inicijalizaciju Vaših objekata.

Nemojte davati konstruktorima, ili destruktorima povratnu vrednost.

Nemojte davati parametre destruktoru.

const funkcije članice

Ako deklarišete metod klase sa const, podrazumeva se da metod neće promeniti vrednost ni jednog člana klase. Da biste deklarisali metod klase kao konstantu, stavite ključnu reč const posle zagrada, ali pre znaka tačka-zarez. Deklaracija konstantne funkcije članice SomeFunctionO ne prihvata argumente i vraća void. Ona izgleda ovako:

```
void SomeFunctionO const;
```


Funkcije pristupa se često deklarišu kao konstantne funkcije, korišćenjem modifikatora const. Klasa Cat ima dve funkcije pristupa:

```
void SetAge(int anAge);
int GetAgeO;
```

SetAge() ne može biti const, jer menja promenljivu članicu itsAge. GetAgeO, sa druge strane, može i trebalo bi da bude const, jer ne menja klasu uopšte. GetAgeO, jednostavno, vraća tekuću vrednost promenljive članice itsAge. Zato bi deklaracija ovih funkcija trebalo da bude napisana ovako

```
void SetAge(int anAge);
int GetAgeO const;
```

Ako deklarišete funkciju sa const, a implementacija te funkcije menja objekat, menjajući vrednost nekog od njegovih članova, kompajler će to označiti kao grešku. Na primer, kada biste napisali GetAgeO na takav način da on broji koliko je puta od Cat tražena njena starost, to bi generisalo grešku kompajlera, zato što biste menjali objekat Cat, pozivajući ovaj metod.

^|IUi4)MSNA|, Koristite const uvek kada je to moguće. Deklarišite funkcije članice sa const uvek kada one ne bi trebalo da promene objekat, omogućavajući kompajleru da Vam pomogne u pronalaženju grešaka; to je brže i jeftinije nego da sami tražite.

Dobra programerska praksa je da deklarišete onoliko const metoda koliko je to moguće. Svaki put kada to uradite, Vi omogućavate kompajleru da uoči Vaše greške, umesto da dopustite da one postanu bagovi, koji će se pojaviti prilikom izvršenja Vašeg programa.

Interfejs protiv implementacije

Kao što ste naučili, klijenti su delovi programa, koji kreiraju i koriste objekte Vaše klase. Interfejs Vase klase možete smatrati kao ugovor sa ovim klijentima. Ugovor ukazuje kakve podatke ima Vaša klasa i kako će se ona ponašati.

Na primer, u deklaraciji klase Cat, kreirate ugovor da će svaka Cat imati promenljivu članicu itsAge koja se može inicijalizovati u njenom konstruktoru, a kojom se dodeljuje vrednost njenom funkcijom pristupa SetAge(), i pročitati funkcijom pristupa GetAge. Takođe obećavate da će svaka Cat znati da MeowQ (engl. meow = mijaukati).

Ako GetAgeO učinite const funkcijom, kao što bi trebalo, ugovor, takođe, obećava da GetAgeO neće promeniti Cat za koju se poziva.

C + + je strogo tipiziran, što znači da kompajler prisiljava na poštovanje ovih ugovora, ukazujući Vam na grešku kada ih ne poštujete. Listing 6.5 demonstrira program koji se ne kompajlera, zbog nepoštovanja ovih ugovora.

Listing 6.5 se ne kompajlira!

Listing 6.5: Demonstracija nepoštovanja interfejsa

```
// Demonstrira greske kompajlera

linclude <iostream.h>          // za cout

class Cat
{
 public:
    Cat(int initialAge);
    -Cat();
    int GetAgeO const;          // konstantna funkcija pristupa
    void SetAge (int age);
    void Meow();
 private:
   int itsAge;
};

   // konstruktor za Cat,
   Cat::Cat(int initialAge)
   {
      itsAge = initialAge;
      cout « "Konstruktor klase Cat\n";
   }

   Cat::-Cat()                  // destruktor, ne radi nista
   {
      cout « "Destruktor klase Cat\n";
   }
  // GetAge, konstantna funkcija
  // ali mi ne postujemo konstantnost!
  int Cat::GetAge() const
  {
     return (itsAge++);         // ne postuje konstantnost!
  }

  // definicija za SetAge, javna
  // funkcija pristupa

  void Cat::SetAge(int age)
  {
     // postavi promenljivu clanicu njenu starost na
     // vrednost koja je predana u parametru age
     itsAge = age;


// definicija metode Meow
// vraca: void
// parametri: Nema
// akcija: Na ekranu stampa "mijau"
void Cat::Meow()
{
   cout « "Mijau.\n";
}

// demonstrira razlicita ne postovanja
// interfejsa i rezultuje predstavljaju greske kompajlera
int main()
{
   Cat Frisky;              // ne odgovara deklaraciji
   Frisky.Meow();
   Frisky.Bark();           // Ne, sasavi, macke ne mogu da laju.
   Frisky.itsAge = 7;       // itsAge je privatna
    return 0;
```

Ovaj program se ne kompajlira, pa, zbog toga, nema izlaza.

Bilo je zabavno napisati ovaj program, jer u njemu ima mnogo grešaka.

Linija 11 deklariše GetAgeQ kao const funkciju pristupa, što bi ona i trebalo da bude. U telu GetAgeQ, ipak, u liniji 32 se uvećava promenljiva članica itsAge. Pošto je ova metoda deklarisana kao const, ona ne sme promeniti vrednost od itsAge. Zbog toga se ovo označava kao greška prilikom kompajliranja programa.

U liniji 13 MeowQ nije deklarisana kao const. Iako ovo nije greška, to je loša programska praksa. Bolji dizajn uračunava da ovaj metod ne menja promenljive članice klase Cat. Zato bi MeowQ trebalo da bude const.

Linija 58 prikazuje definiciju Cat objekta, Frisky. Cat sada ima konstruktor, koji prihvata celobrojnu vrednost kao parametar. Ovo znači da morate predati parametar. Zato što u liniji 58 nema nema parametra, ona je označena kao greška.

Linija 60 prikazuje poziv metoda klase BarkQ. BarkQ nikada nije bio deklarisan. Pa je zato ovo netačno.

Linija 61 prikazuje itsAge, kojoj se dodeljuje vrednost 7. Zato što je itsAge privatni podatak clan, to se označava kao greška prilikom kompajliranja programa.

Zašto koristiti kompajler za hvatanje grešaka?

Bilo bi predivno napisati kod koji je stoprocentno čist od bagova, ali vrlo malo programera je sposobna da to uradi. Ipak, mnogi programeri su razvili sistem koji pomaže u minimiziranju bagova, njihovim hvatanjem i sredivanjem u ranom procesu.

Iako je ukazivanje kompajlera na greške izuzetno dosadno i predstavlja prokletstvo egzistencije programera, one su daleko bolje nego alternativa. Slabo tipiziran jezik omogućava Vam da ne poštujete Vaše ugovore bez oglašavanja kompajlera, ali zbog toga će Vaš program pasti u vreme izvršenja, kada, na primer, Vaš šef posmatra.

Greške **u** vreme kompajliranja, to jest, greške pronadene prilikom kompajliranja daleko su podnošljivije od grešaka **u** vreme izvršenja, to jest, grešaka pronadenih prilikom izvršenja programa. Zbog toga se greške **u** vreme kompajliranja mogu daleko pouzdanije pronaći. Moguće je izvršiti program mnogo puta bez prolaska kroz svaku moguću putanju koda. Greška **u** vreme izvršenja se može sakriti neko vreme. Greške u vreme kompajliranja se pronalaze svaki put kada kompajlirate. Njih je lakše identifikovati i srediti. Cilj kvalitetnog programiranja je osigurati da kod nema bagova u vreme izvršenja. Jedna isprobana-i-prava tehnika za ostvarenje ovoga je korišćenje kompajlera da uhvati Vaše greške rano u procesu razvoja.

Gde staviti deklaracije klasa i definicije metoda?

Svaka funkcija koju deklarišete za Vašu klasu mora imati definiciju. Definicija se, takode, naziva implementacija funkcije. Kao i druge funkcije, definicija metoda klase ima funkcijsko zaglavlje i funkcijsko telo.

Definicija mora biti u datoteci koju kompajler može pronaći. Većina C++ kompajlera želi da se ta datoteka završava sa .C, ili .CPP. Ova knjiga koristi .CPP, ali proverite Vaš kompajler, da biste videli šta on vise "voli".

^NAPOMEN^ Mnogi kompajleri podrazumevaju da su datoreke koje se C programi završavaju sa . C, a da se C++ programske datoteke završavaju sa .CPR Vi možete koristiti bilo koju ekstenziju, ali .CPP će minimizirati konfuziju.

Vi imate slobodu da, takode, u ovu datoteku stavite deklaraciju, ali to nije dobra programska praksa. Konvencija, koju većina programera usvaja, je smeštanje deklaracije u ono što se naziva *zaglavlje,* obično sa istim imenom, ali sa završetkom .H, .HP, ili .HPP. Ova knjiga imenuje zaglavlja sa .HPP, ali proverite šta Vaš prevodilac, da biste videli šta on vise "voli".

Na primer, Vi stavite deklaraciju klase Cat u datoteku nazvanu CAT.HPP, a definicije metoda klase u datoteku nazvanu CAT.CPP. Onda povezujete zaglavlje sa .CPP datotekom, stavljenjem sledećeg koda u početku datoteke CAT.CPP:

```
linclude  Cat.hpp
```

Ovo govori kompajleru da učita CAT.HPP u datoteku, baš kao da ste Vi ukucali njen sadržaj na ovom mestu. Zašto preterivati sa njihovim rastavljanjem, kada ćete ih svakako ponovo učitatiP^ Klijenti Vaše klase, uglavnom, ne brinu o implementacionim specifičnostima. Čitanje zaglavlja im saopstava sve što oni treba da znaju; oni mogu ignorisati implementacione datoteke.

^MAPOMIMA^ Deklaracija klase govori kompajleru šta je klasa, koje podatke čuva i koje funkcije ima. Deklaracija klase se naziva interfejs, jer govori korisniku kako da kontaktira sa klasom. Interfejs se, obično, čuva u .HPP datoteci, na koju se upućuje sa zaglavlja.

Definicija funkcije govori kompajleru kako funkcija radi. Definicija funkcije se naziva implementacija metoda klase, a čuva se u .CPP datoteci. Implementacioni detalj:

klase su briga samo autora klase. Klijenti klase, to jest, delovi programa koji koriste klasu, ne treba da znaju kako su funkcije implementirane.

Inline implementacija

Kao što možete tražiti od kompajlera da učini regularnu funkciju `inline`, možete metode klase učiniti `inline`. Ključna reč `inline` se pojavljuje pre povratnog tipa. Inline implementacija funkcije GetWeightQ, na primer, izgleda ovako:

```
inline int Cat::GetWeight()
{
return itsWeight;          // vraca podatak clan Weight
)
```

Takode možete staviti definiciju funkcije u deklaraciju klase, što automatski čini funkciju i nl i ne. Na primer:

```
class Cat
{
public:
int GetWeight() { return itsWeight; }     // inline
void  SetWeight(int  aWeight);
};
```

Uočite sintaksu definicije funkcije GetWeightQ. Telo inline funkcije počinje odmah nakon deklaracije metoda klase; nema znaka tačka-zarez posle zagrada. Kao i kod svake funkcije, definicija počinje otvarajućom, a završava se zatvarajućom zagradom. Kao i obično, beline nisu važne; deklaraciju ste mogli napisati i kao

```
class Cat
{
public:
int   GetWeight()
{
return itsWeight;
}                              // inline
void  SetWeight(int  aWeight);
};
```

Listinzi 6.6 i 6.7 obnavljaju klasu Cat, ali stavljaju deklaraciju u CAT.HPP, a implementaciju funkcija u CAT.CPP. Listing 6.7 takode menja funkcije pristupa i funkciju Meow(), tako da su one sada i nl ine.

Listing 6.6: Deklaracija klase Cat u CAT.HPP

```
#include <iostream.h>
class Cat
 (
publi c:
 Cat (int initialAge);
```

nastavlja se

Listing 6.6: Deklaracija klase Cat u CAT.HPP

nastavak

```
    ~Cat();
     int GetAgeO  { return itsAge;}            // inline!
     void SetAge (int age) { itsAge = age;}    // inline!
     void MeowQ  { cout « "Mijau.\n";}          // inline!
    private:
   int itsAge;
```

Listing 6.7: Implementacija klase Cat u CAT.CPP

```
1:   // Demonstrira inline funkcije
2:   // i ukljucenje zaglavlja
3:
4:   linclude "cat.hpp"   // proverite da li ste ukljucili zaglavlja!
5:
6:
7:   Cat::Cat(int initialAge)    //konstruktor
8:
9:      itsAge = initialAge;
10:
11:
12:  Cat::-Cat()              //destruktor, ne radi nista
13:
14:
15:
16:  // Kreira macku, postavlja njenu starost, pravi
17:  // mijauk, saopstava nam njenu starost, a onda ponovo pravi mijauk
18:   nt main()
19:
20:     Cat Frisky(5);
21:     Frisky.MeowQ;
22:     cout « "Frisky je macka koja ima " ;
23:     cout « Frisky.GetAgeO « " godina.\n";
24:     Frisky.Meow();
25:     Frisky.SetAge(7);
26:     cout « "Sada Frisky ima " ;
27:     cout « Frisky.GetAgeO « " godina.\n";
28:      return 0;
29:
```

```
Mijau.
Frisky je macka koja ima 5 godina.
Mijau.
Sada Frisky ima 7 godina.
```

Kod koji je predstavljen u listinzima 6.6 i 6.7 sličan je kodu u listingu 6.4, što su tri metoda napisana kao `inline` u deklaracionoj datoteci i deklaracija je izdvojena u CAT.HPP.

GetAgeO je deklarisana u liniji 7 i obezbedena je njena inline implementacija. Linije 8 i 9 obezbeduju još neke `inline` funkcije, ali upotrebljivost ovih funkcija je nepromenjena u odnosu na prethodne "outline" (engl. outline = izvanlinijske) implementacije.

Linija 4 u listingu 6.7 prikazuje linclude "cat.hpp", koja donosi listinge iz CAT.HPP. §|и Uključivanjem cat. hpp, saopštili ste pretkompajleru da učita cat. hpp u datoteku, kao da je ona tu bila ukucana, počevši od linije 5.

Ova tehnika Vam dozvoljava da stavite Vaše deklaracije u neku drugu datoteku, a ne samo u Vašu implementaciju - imaćete tu deklaraciju na raspolaganju kada je potrebna kompajleru. Ovo je opšta tehnika u C + + programiranju. Tipično, deklaracije klasa su u .HPP datoteci, koja se, često, uključuje sa linclude u pridruženu CPP datoteku.

Linije 18-29 ponavljaju glavnu funkciju iz listinga 6.4. To pokazuje da proglašavanje ovih funkcija i nl ine ne menja njihovo delovanje.

Klase sa drugim klasama kao podacima clanovima

Nije neuobičajeno izgraditi kompleksnu klasu deklarisanjem jednostavnijih klasa i njihovim uključivanjem u deklaraciju komplikovanije klase. Na primer, Vi ćete, možda, deklarisati klasu točka, klasu motora, klasu transmisije i tako dalje, a onda ih kombinovati u klasu automobila. Ovo deklariše ima vezu. Automobil ima motor, ima točkove i ima transmisiju.

Razmotrite drugi primer. Pravougaonik se sastoji od linija. Linija se definise sa dve tačke. Tačka se definise sa koordinatama x i y. Listing 6.8 prikazuje kompletnu deklaraciju klase Rectangle, onako kako bi se mogla pojaviti u RECTANGLE.HPP. Zato što se pravougaonik definise kao četiri linije, koje povezuju četiri tačke, a svaka tačka se odnosi na koordinatu na grafiku, mi prvo deklarišemo klasu Point, koja će čuvati x, y koordinate svake tačke. Listing 6.9 prikazuje kompletnu deklaraciju obe klase.

Listing 6.8: Deklarisanje kompletne klase

```
1:   // Pocetak Rect.hpp
2:   linclude <iostream.h>
3:   class Point      // sadrzi x,y koordinate
4:   {
5:      // nema konstruktora, koristi se podrazumevani
6:      public:
7:         void SetX(int x) { itsX = x; }
8:         void SetY(int y) { itsY = y; }
9:         int GetX()const { return itsX;}
10:        int GetY()const { return itsY;}
11:     private:
```

nastavljase

Listing 6.8: Deklarisanje kompletne klase

nastavak

```
12          int  itsX;
13          int  itsY;
14          // kraj deklaracije klase Point
15
16
17    class   Rectangle
18    {
19       public:
20          Rectangle (int top, int left, int bottom, int right);
21          čRectangle () {}
22
23          int GetTopO const { return itsTop; }
24          int GetLeft() const { return itsLeft; }
25          int GetBottom() const { return itsBottom; }
26          int GetRight() const { return itsRight; }
27
28          Point   GetUpperLeft() const { return itsUpperLeft; }
29          Point   GetLowerLeft() const { return itsLowerLeft; }
30          Point   GetUpperRight() const   return itsUpperRight; }
31          Point   GetLowerRight() const   return itsLowerRight; }
32
33          void SetUpperLeft(Point Location)  {itsUpperLeft = Location;}
34          void SetLowerLeft(Point Location)  {itsLowerLeft = Location;}
35          void SetUpperRight(Point Location)  {itsUpperRight = Location;
36          void SetLowerRight(Point Location)  {itsLowerRight = Location;
37
38          void SetTop(int top) { itsTop = top; }
39          void SetLeft (int left) { itsLeft = left; }
40          void SetBottom (int bottom) { itsBottom = bottom; }
41          void SetRight (int right) { itsRight = right; }
42
43          int GetArea() const;
44
45       private:
46          Point   itsUpperLeft;
47          Point   itsUpperRight;
48          Point   itsLowerLeft;
49          Point   itsLowerRight;
50          int     itsTop;
51          int     itsLeft;
52          int     itsBottom;
53          int     itsRight;
54    };
55    // kraj datoteke Rect.hpp
```

Listing 6.9: RECT.CPP.

```
1:  // Pocetak rect.cpp
2:  linclude "rect.hpp"
3:  Rectangle::Rectangle(int top, int left, int bottom, int right)
4:  {
5:        itsTop = top;
6:        itsLeft = left;
7:        itsBottom = bottom;
8:        itsRight = right;
9:
10        itsUpperLeft.SetX(left);
11        itsUpperLeft.SetY(top);
12
13        itsUpperRight.SetX(right);
14        itsUpperRight.SetY(top);
15
16        itsLowerLeft.SetX(left);
17        itsLowerLeft.SetY(bottom);
18
19        itsLowerRight.SetX(right);
20        i ts LowerRi ght.SetY(bottom);
21
22
23
24  // proracunava povrsinu pravougaonika, pronalazenjem temena,
25  // kreira sirinu i visinu, a onda mnozi
26  int Rectangle::GetArea() const
27  {
28        int Width = itsRight-itsLeft;
29        int Height = itsTop - itsBottom;
30        return (Width * Height);
31
32
33  int  main()
34
35        //inicijalizuje  lokalnu  Rectangle  promenljivu
36        Rectangle MyRectangle (100, 20, 50, 80 );
37
38        int Area = MyRectangle.GetArea();
39
40        cout « "Povrsina:  " « Area « "On";
41        cout « "Gornja leva X koordinata: ";
42        cout «  MyRectangle.GetUpperLeft().GetX();
43     return 0;
44
```

```
Povrsina: 3000
Gornja leva X koordinata: 20
```

jИЖJj^ Linije 3-14 u listingu 6.8 deklarišu klasu `Point`, koja se koristi za čuvanje specifičnih x, y koordinata na grafiku. Kao što je napisano, ovaj pro gram ne koristi mnogo `Poi nt.` Ipak, druge metode crtanja zahtevaju `Poi nt.`

Unutar deklaracije klase Point, deklarišete dve promenljive članice (itsX i itsY) u linijama 12 i 13. Ove promenljive čuvaju vrednosti koordinata. Kako se x-koordinata uvećava, Vi se na grafiku pomerate udesno. Kako se y-koordinata uvećava, Vi se na grafiku pomerate nagore. Drugi grafici koriste druge sisteme. Neki prozorski programi, na primer, uvećavaju y-koordinatu kako se pomerate nadole u prozoru.

Klasa Point koristi inline funkcije pristupa za dobijanje i postavljanje x i y tačaka, koje su deklarisane u linijama 7-10. Point koristi podrazumevani konstruktor i destruktor. Zato, morate postaviti njihove koordinate eksplicitno.

Linija 17 zapocinje deklaraciju klase Rectangle. Rectangle se sastoji od četiri tačke, koje predstavljaju temena od Rectangle (engl. rectangle = pravougaonik).

Konstruktor za Rectangl e (linija 20) prihvata četiri celobrojne vrednosti, poznate kao top, left, bottom i right. Četiri parametra za konstruktor se kopiraju u četiri promenljive članice (listing 6.9), a onda se kreiraju četiri Pointa.

Pored uobičajenih funkcija pristupa, Rectangle ima funkciju GetAreaO, koja je deklarisana u liniji 43. Umesto čuvanja površine kao promenljive, funkcija GetAreaO proračunava površinu u linijama 28-29 u listingu 6.9. Da bi ovo uradila, ona proračunava širinu i visinu pravougaonika, a, onda, množi ove dve vrednosti.

Uzimanje x-koordinate gornjeg levog ugla pravougaonika zahteva da pristupite tački UpperLeft i zatražite njenu X vrednost. Zato što je GetUpperleft () metod klase Rectangle, on može direktno pristupiti privatnim podacima klase Rectangle, uključujući i itsUpperLeft. Zato što je itsUpperLeft klase Point, a vrednost itsX klase Point privatna, GetUpperLeft() ne može direktno pristupiti ovom podatku. Umesto toga, mora se koristiti javna funkcija pristupa GetX(), da bi se dobila ta vrednost.

Linija 33 iz listinga 6.9 je početak tela programa. Do linije 36 nikakva memorija nije bila alocirana i ništa se nije dogodilo. Jedino ste saopštili kompajleru kako da napravi tačku i pravougaonik, u slučaju da je to potrebno.

U liniji 36 definišete Rectangle, predavanjem vrednosti za Top, Left, Bottom i Right.

U liniji 38 pravite lokalnu promenljivu, Area, tipa i nt. Ova promenljiva čuva površinu od Rectangle, koga ste kreirali. Inicijalizujete Area vrednošću koja je vraćena iz funkcije GetAreaO klase Rectangle.

Klijent klase Rectangle bi mogao kreirati Rectangle objekat i dobiti njegovu površinu bez poznavanja implementacije mfunkcije GetAreaO.

RECT.HPP je prikazana u listingu 6.8. Samo posmatranjem zaglavlja, koje sadrži deklaraciju klase Rectangle, programer zna da GetAreaO vraća int. Kako GetAreaO

ostvaruje svoju magiju nije briga korisnika klase Rectangle. U stvari, autor klase Rectangle bi mogao da promeni GetAreaO, bez uticanja na programe koji koriste tu klasu.

Strukture

Veoma blizak "rodak" ključnoj reči class je ključna reč struct, koja se koristi za deklarisanje strukture. U C++-u struktura je isto što i klasa, osim što su njeni clanovi podrazumevano javni. Možete deklarisati strukturu baš kao što deklarišete i klasu i možete joj dodeliti potpuno iste podatke članove i funkcije. U stvari, ako sledite dobru programersku praksu da uvek eksplicitno deklarišete privatne i javne sekcije Vaše klase, u torn slučaju neće biti nikakve razlike.

Pokušajte da unesete sledeće promene u listing 6.8:

- U liniji 3 promenite class Point u struct Point.
- U liniji 17 promenite class Rectangle u struct Rectangle.

Sada ponovo izvršite program i uporedite izlaz. Ne bi trebalo da bude bilo kakvih promena.

Zašto dve ključne reci rade istu stvar

Vi se verovatno pitate zašto dve ključne reči rade istu stvar. Kada je C+ 4- razvijen, on je izgraden kao proširenje jezika C. C ima strukture, iako one nemaju metode klase. Bjarne Stroustrup, kreator C++-a, temeljio je izgradnju na struct, ali je promenio ime u class, da bi predstavio novu, proširenu funkcionalnost.

<j| **PAZITI** j£ Stavite deklaraciju Vaše klase u HPP datoteku, a funkcije članice u CPP datoteku.

Koristite const uvek kada to možete.

Proučite klase, pre nego što nastavite.

Rezime

Danas ste naučili kako da kreirate nove tipove podataka koji se nazivaju klase. Naučili ste kako da definišete promenljive ovih novih tipova, nazvanih objekti. Klasa ima podatke čTanove, koji su promenljive razlidtih tipova, uključujući i druge klase. Klasa, takođe, uključuje funkcije članice, takode poznate i kao metodi. Ove funkcije članice koristite za manipulisanje podacima clanovima i obavite neke druge službe.

Članovi klase, i podaci i funkcije, mogu biti javni, ili privatni. Javni clanovi su pristupačni svakom delu Vašeg programa. Privatni članovi su pristupačni samo funkcijama članicama klase.

Dobra programska praksa je izolacija interfejsa, ili deklaracije, ili klase u zaglavlje. Ovo, obično, radite u datoteci sa `.HPP` ekstenzijom. Implementacija metoda klase se piše u datoteku sa `.CPP` ekstenzijom.

Konstruktori klase inicijalizuju objekte. Destruktori klase uništavaju objekte a, često se koriste i da oslobode memoriju koju su alocirali metodi klase.

Pitanja i odgovori

P **Koliki** je objekat klase?

O Veličina objekta klase u memoriji se određuje sumom veličina njegovih promenljivih članica. Metodi klase ne zauzimaju prostor kao deo memorije koji se rezerviše za objekat.

P **Ako** deklarišem klasu **Cat** privatnim članom **itsAge,** a, onda, definišem dva **Cat** objekta, Frisky i Boots, da **li Boots** može pristupiti Friskyjevoj promenljivoj članici **itsAge?**

O Da. Privatni podatak je raspoloživ funkcijama članicama klase i različiti primerci klase mogu pristupiti podacima drugih primeraka. Drugim rečima, ako su i Frisky i Boots primerci klase Cat, Friskyjeve funkcije članice mogu pristupiti Friskyjevim i Bootsovim podacima.

P Zašto da ne proglasim sve podatke članove javnim?

O Proglašenje podataka članova privatnim omogućava klijentu klase da koristi podatke bez brige o tome kako se oni čuvaju, ili proračunavaju. Na primer, ako klasa Cat ima metod GetAgeO, klijenti klase Cat mogu tražiti starost mačke bez znanja, ili brige o tome da li mačka čuva svoju starost u promenljivoj članici, ili je proračunava u preletu.

P **Ako** korišćenje **const** funkcije, koja menja klasu, prouzrokuje grešku kompajlera, zašto ne bih samo izostavio reč **const** i izbegao javljanje greške?

0 Ako Vaša funkcija članica logički ne bi trebalo da menja klasu, korišćenje ključne reči const je dobar način da zatražite od kompajlera pomoć u pronalaženju grubih grešaka. Na primer, GetAgeO možda ne bi imala razlog da promeni klasu Cat, ali Vaša implementacija ima ovu liniju:

```
if (itsAge = 100) cout « "Hey! You're 100 years old\n";
```

Deklarisanje GetAgeO kao const prouzrokuje da ovaj kod bude označen kao greška. Vi ste želeli da proverite da li je itsAge jednako 100, ali, umesto toga, nenamerno ste dodelili 100 promenljivoj itsAge. Zato što ova dodela menja klasu, Vi ste rekli da ovaj metod neće promeniti klasu, kompajler je sposoban da nađe grešku.

Ova vrsta greške može biti veoma teška za pronalaženje samo skeniranjem koda. **Oko Često** vidi samo ono što očekuje da vidi. Što **je** važnije, može izgledati da program radi ispravno, ali itsAge **je** sada bila postavljena na pogrešan broj. **Ovo će** prouzrokovati probleme, pre, ili kasnije.

P **Da li ikada postoji razlog za korišćenje strukture u C++ programu?**

O Mnogi **C++** programeri rezervišu ključnu reč struct za klase koje nemaju funkcije. Ovo **je** ponavljanje starih **C** struktura, koje nisu mogle imati funkcije. Iskreno, za mene **je** to konfuzna i siromašna programska praksa. Današnjoj strukturi bez metoda će metodi možda, biti potrebni sutra. Onda ćete biti prisiljeni ili da promenite tip u class, ili da prekršite Vaše pravilo i završite sa strukturama sa metodima.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Šta je operator tačka i za šta se koristi?
2. Šta rezerviše memoriju - deklaracija, ili definicija?
3. Da li je deklaracija klase njen interfejs, ili implementacija?
4. Kakva je razlika između javnih i privatnih podataka članova?
5. Da li funkcija članica može biti privatna?
6. Da li podatak član može biti javan?
7. Ako deklarisete dva `Cat` objekta, da li oni mogi imati različite vrednosti u svojim podacima clanovima `itsAge`?
8. Da li se deklaracije klasa završavaju znakom tačka-zarez? A definicije metoda?
9. Kako bi izgledalo zaglavlje za `Cat` funkciju, `Meow`, koja ne prihvata ni jedan parametar, a vraća `void`?
10. Koja funkcija se poziva za inicijalizaciju klase?

Vežbe

1. Napišite kod koji deklariše klasu nazvanu Employee (engl. employee = zaposleni) sa ovim podacima clanovima: age (engl. age = starost), YearsOfServi ce (engl. years of service = radni staž) i Sal ary (engl. salary = primanja).
2. Prepravite klasu Employee, tako da podaci clanovi budu privatni, i obezbedite javne metode pristupa za dobijanje i postavljanje svakog podatka člana.
3. Napišite program sa klasom Employee, koji pravi dva Employeeja: postavlja njihov age, YearsOfServi ce i Salary i štampa njihove vrednosti.
4. Nastavljajući Vežbu 3, obezbedite metod klase Employee, koja izveštava koliko hiljada dolara zaposleni zaraduje, zaokruženo na najbližih 1.000.
5. Promenite klasu Employee, tako da možete inicijalizovati age, YearsOfServi ce i Salary prilikom kreiranja zaposlenog.
6. ISTERIVAČ BAGOVA: Šta nije u redu sa sledećom deklaracijom?

```
class Square
{
public:
    int Side;
}
```

7. ISTERIVAČ BAGOVA: Zašto sledeća deklaracija klase nije veoma korisna?

```
class Cat
{
    int GetAge()const;
private:
    int itsAge;
};
```

8. ISTERIVAČ BAGOVA: Koja će tri baga kompajler naći u ovom kodu?

```
class TV
{
public:
    void SetStation(int Station);
    int GetStation() const;
private:
    int itsStation;
};
main()
{
    TV myTV;
    myTV.itsStation = 9;
    TV.SetStation(IO);
    TV my0therTv(2);
}
```


Dan 7

Vise programskog toka

Programi ostvaruju veći deo svog posla, grananjem i upetljavanjem. U Danu 4, "Izrazi i iskazi", naučili ste kako da granate svoj program, korišćenjem if iskaza. Danas učite

- šta su petlje i kako se koriste
- kako izgraditi različite petlje
- koja je alternativa za duboko-ugnježdene i f/el se iskaze.

Upetljavanje

Mnogi problemi programiranja se rešavaju ponavljanjem delova na iste podatke. Postoje dva načina da se ovo uradi: rekurzija (opisana na Dan **5,** "Funkcije") i iteracija. Iteracija znači obavljanje iste stvari ponovo i ponovo. Glavni metod iteracije je petlja.

Koreni upetljavanja goto

U vreme rane kompjuterske nauke programi su bili nezgodni, brutalni i kratki. Petlje su se sastojale od labele, nekih iskaza i skoka.

U C++ -u labela je samo ime, praćeno znakom dve tačke (:). Labela se stavlja sa leve strane, u odnosu na legalan C++ iskaz, a skok se ostvaruje pisanjem goto, posle koga sledi ime labele. Listing 7.1 ilustruje ovo.

Listing 7.1: Upetljavanje sa ključnom reel goto

```
// Listing 7.1
// Upetljavanje sa goto

linclude <iostream.h>

int main()
{
      int counter = 0;       // inicijalizuje brojac
loop: counter ++;            // vrh petlje
        cout « "brojac: " « counter « "\n";
       if (counter < 5)             // testiraj vrednost
           goto loop;                  // skoci na vrh

       cout « "Zavrseno. Brojac: " « counter « ".\n";
   return 0;


      brojac: 1
      brojac: 2
      brojac: 3
      brojac: 4
      brojac: 5
      Zavrseno. Brojac: 5.
```

U liniji 8 brojač se inicijalizuje na 0. Labela loop je u liniji 9 označavajući vrh petlje. Brojač se uvećava i štampa se njegova nova vrednost. Vrednost se testira u liniji 11. Ako je manji od 5, iskaz if je istinit i izvršava se iskaz goto. Ovo prouzrokuje da izvršenje programa "skoči" na liniju 9. Program nastavlja upetljavanje sve dok brojač ne postane jednak 5; u torn trenutku on "propada kroz" petlju i štampa se finalni izlaz.

Zašto se goto izbegava?

Naredba goto je kasno primila lošu kritiku. Iskazi goto mogu prouzrokovati skok na bilo koju lokaciju u Vašem izvornom kodu, unazad, ili napred. Nepažljiva upotreba goto iskaza je prouzrokovala zapetljane, nemoguće-za-čitanje programe, poznate kao "špageta kod". Zbog ovoga, učitelji kompjuterskih nauka su proveli poslednjih 20 godina udarajući jednom lekcijom po glavama svojih studenata: "Nikada, nikada, nikada nemojte koristiti goto! To je zlo!"

Da bi se izbegla upotreba goto, uvedene su sofisticirane, cvrsto kontrolisane komande upetljavanja: for, while i do.. .while. Njihovim korišćenjem, prave se programi, koji su lakši za razumevanje, a goto se generalno izbegava, ali neko može tvrditi da je slučaj malo prenaglašen. Kao i svaka alatka, pažljivo korišćena i u pravim rukama, goto može biti korisna ideja, a ANSI komitet je odlučio da je zadrži u jeziku, jer je njena upotreba legitimna.

Iskaz goto

Da biste upotrebili iskaz goto, potrebno je da napišete reč goto, posle koje sledi ime labele. Ovo prouzrokuje bezuslovan skok na labelu.

Primer

```
if (value > 10)      goto end;if (value < 10)      goto end;cout « "vrednost je
^10!";end:cout « "uradjeno";
```

ф р о г о к и ф Upotreba goto je, skoro uvek, znak lošeg dizajna. Najbolji savet je da izbegavate njeno korišćenje. Za 10 godina programiranja, meni je bila potrebna samo jednom.

Petlje while

Petlja while prouzrokuje da Vaš program ponavlja sekvencu iskaza, sve dok je početni uslov istinit. U primeru za goto, u listingu 7.1, brojač je bio uvećavan, dok njegova vrednost nije dostigla 5. Listing 7.2 prikazuje isti program, koji je prepravljen, da bi imao prednosti petlje loop.

Listing 7.2: whi 1 e petlje

```
// Listing 7.2
// Upetljavanje sa while

linclude <iostream.h>

int main()
{
  int counter = 0;               // inicijalizuje uslov

   while(counter < 5)      // testira da li je uslov jos uvek istinit
    {
       counter++;              // telo petlje
       cout « "brojac: " « counter « "\n";


   cout « "Zavrseno. Brojac: " « counter « ".\n";
    return 0;


       brojac: 1
       brojac: 2
       brojac: 3
       brojac: 4
       brojac: 5
       Zavrseno. Brojac: 5.
```

Ovaj jednostavan program demonstrira fundamente petlje whi 1 e. Uslov se testira i, ako je istinit, izvršava se telo petlje whi 1 e. U ovom slučaju, uslov koji je

testiran u liniji 10 je da li je brojač manji od 5. Ako je uslov istinit, izvršava se telo petlje; u liniji 12 uvećava se brojač, a u liniji 13 štampa se vrednost. Kada uslovni iskaz u liniji 10 ne uspe (kada brojač nije vise manji od 5), celo telo petlje whi l e (linije 11-14) se preskače. Izvršenje programa pada na liniju 15.

Iskaz while

Sintaksa za iskaz while je sledeća:

```
while ( uslov )
iskaz;
```

uslov je bilo koji C++ izraz, a *iskaz* je bilo koji važeći C++ iskaz, ili blok iskaza. Kada *uslov* proračunava TRUE(l), izvršava se *iskaz,* o onda se *uslov* ponovo testira. Ovo se nastavlja, sve dok test *uslova* ne bude FALSE, kada se petlja while završava, a izvršenje nastavlja na prvoj liniji ispod *iskaza.*

Primer

```
// brojanje do 10
int x = 0;
while (x < 10)
cout « "X: " « x++;
```

Komplikovaniji iskazi whi 1 e

Uslov koji se testira u petlji while može biti kompleksan koliko i bilo koji legalan C++ izraz. Ovo može uključiti izraze proizvedene korišćenjem logičkih operatora && (I), !! (ILI) i ! (NE). Listing 7.3 je nešto komplikovaniji iskaz while.

Listing 7.3: Kompleksne whi le petlje.

```
1:  // Listing 7.3
2:  // Kompleksni while iskazi
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    unsigned short smal1;
9:    unsigned long  large;
10:    const unsigned short MAXSMALL=65535;
11:
12:    cout « "Unesite mali broj: ";
13:    cin » smal1;
14:    cout « "Unesite veliki broj: ";
15:    cin » large;
16:
17:     cout « "mali: " « small « "...";
18:
```


```
// za svaku iteraciju, testiraju se tri uslova
while (small < large && large > 0 && small < MAXSMALL)

{
   if (small % 5000 == 0)  // napisi tacku svakih 5k linija
     cout « ".";

   small++;

   large-=2;


cout « "\nMali: " « small « " Veliki: " « large « endl;
return 0;
```

```
Unesite mali broj: 2
Unesite veliki broj: 100000
mali: 2.........
Mali: 33335 Veliki: 33334
```

JJШJTJ^ Ovaj program je igra. Unesite dva broja, jedan mali, a jedan veliki. Manji broj će se uvećavati za jedan, a veći će se smanjivati za dva. Cilj igre je pogoditi kada će se oni susresti.

U linijama 12-15 unose se brojevi. Linija 20 priprema petlju while, koja će se nastaviti samo dok se susreću tri uslova:

- smal **1** nije veći od large
- **1** arge nije negativan
- smal l ne prelazi veličinu male celobrojne vrednosti (MAXSMALL).

U liniji 23 vrednost u small se proračunava po modulu 5.000. Ovo ne menja vrednost u small; ipak, ovo vraća vrednost 0 samo kada je small sadržalac broja 5.000. Uvek kada je ovo slučaj, tačka (.) se štampa na ekranu, da bi se prikazao progres. U liniji 26 smal **1** se uvećava, a u liniji 28 large se smanjuje za 2.

Kada jedan od tri uslova u petlji whi le ne uspe, petlja se završava, a izvršenje programa se nastavlja posle zatvarajuće zagrade petlje whi le u liniji 29.

^HAPOMIHft^ Operator moduo (%) i složeni uslovi su obradeni Danu 3, "Promenljive i konstante".

continue i break

Pre izvršenja celog skupa iskaza u petlji whi **1** e, bićete vraćeni na vrh te petlje. Iskaz continue skače na vrh petlje.

U drugim slučajevima, možda ćete želeti da izađete iz petlje, pre nego što se su-sretnu izlazni uslovi. Iskaz break trenutno izlazi iz petlje while, a izvršenje programa se nastavlja posle zatvarajuće zagrade.

Listing 7.4 demonstrira upotrebu ovih iskaza. Ovog puta je igra postala komplikovanija. Korisnik se poziva da unese broj small i broj large, broj skip i broj target. Broj smal 1 će biti uvećavan za jedan, a broj 1 arge će biti umanjivan za 2. Dekrement će biti preskočen svaki put kada je broj smal 1 sadržalac broja skip. Igra se završava, ako small postane veći od large. Ako broj large dođe tačno do target, iskaz se štampa, a igra zaustavlja.

Listing 7.4: break i continue

```
1   // Listing 7.4
2   // Demonstrira break i continue
3
4   finclude <iostream.h>
5
6   int main()
7   {
8     unsigned short small;
9     unsigned long  large;
10     unsigned long  skip;
11     unsigned long target;
12     const unsigned short MAXSMALL=65535;
13
14     cout « "Unesite mali broj: ";
15     cin » smal1;
16     cout « "Unesite veliki broj: ";
17     cin » large;
18     cout « "Unesite broj koji treba preskociti: ";
19     cin » skip;
20     cout « "Unesite trazeni broj: ";
21     cin » target;
22
23   cout « "\n";
24
25    // postavi tri uslova za zaustavljanje petlje
26     while (small < large && large > 0 && small < 65535)
27
28     {
29
30       smal1++;
31
32        if (small % skip == 0)  // preskoci dekrement?
33        {
34          cout « "preskakanje na " « small « endl;
35          continue;
36        }
37
```



```
38      if (large == target)     // postignut cilj?
39      {
40        cout « "Cilj ostvaren!";
41        break;
42
43      large-=2;
44                    // kraj petlje while
45
46    cout « "\nMali: " « small « " Veliki: " « large « endl;
47  return 0;
```

```
Unesite mali broj: 2
Unesite veliki broj: 20
Unesite broj koji treba preskociti: 4
Unesite trazeni broj: 6

preskakanje na 4
preskakanje na 8

Mali: 10 Veliki: 8
```

t-wmfjbfe U ovoj igri korisnik gubi; small postaje veći od large, pre nego što se dostigne ciljni broj 6.

U liniji 26 testira se while uslov. Ako je smal 1 i dalje manji od large, 1 arge veći od 0, a smal 1 nije prekoračio maksimalnu vrednost za mali i nt, ulazi se u telo petlje whi 1 e.

U liniji 32 proračunava se ostatak vrednosti smal 1 po modulu vrednosti skip. Ako je small sadržalac vrednosti skip, dolazi se do iskaza continue i izvršenje programa skače na vrh petlje na liniju 26. Ovo efektivno preskače test za target i dekrement promenljive large.

U liniji 38 target se testira u odnosu na vrednost za large. Ako su isti, korisnik je pobedio. Poruka se štampa i dostiže se iskaz break. Ovo prouzrokuje trenutni izlazak iz petlje while, a izvršenje programa se nastavlja u liniji 46.

NAPOMENA^ I continue i break bi trebalo da budu korišćene veoma pažljivo. One su sledeće najopasnije komande posle goto, iz veoma sličnog razloga. Programi koji iznenada menjaju pravac su teži za razumevanje, a slobodna upotreba continue i break će učiniti čak i malu while petlju nečitljivom.

Iskaz continue

countinue; prouzrokuje da petlja while, ili for počne ponovo od vrha petlje.

Primer

```
if (value > 10)
    goto end;

if (value < 10)
    goto end;

cout « "vrednost je 10!";

end:

cout « "obavljeno";
```

Iskaz break

break; prouzrokuje trenutni završetak petlje while, ili for. Izvršenje skače za zatvarajuću zagradu.

Primer

```
while (condition)
{
    if (condition2)
        break;
    // iskazi;
}
```

Petlje while (1)

Uslov koji se testira u petlji while može biti bilo koji važeći C + + izraz. Sve dok je taj uslov istinit, petlja while će se izvršavati. Možete kreirati petlju koja se nikada neće završiti korišćenjem broja 1 za uslov koji će biti testiran. Kako je 1 uvek istina, petlja se nikada neće završiti, osim ako se ne naide na iskaz break. Listing 7.5 demonstrira brojanje do 10, korišćenjem ove ideje.

Listing 7.5: while (1) petlje

```
// Listing 7.5
// Demonstrira uvek istinitu petlju while

linclude <iostream.h>

int main()
{
  int counter = 0;

   while (1)
   {
      counter ++;
      if (counter > 10)
         break;
   }
   cout « "Brojac: " « counter « "\n"
    return 0;


      Brojac: 11
```

JSZEtefr U liniji 10 postavlja se petlja whi 1 e, sa uslovom koji nikada ne može biti neistinit. Petlja uvećava promenljivu brojač u liniji 12, a onda u liniji 13 testira da li je brojač prošao 10. Ako nije, petlja while iterira. Ako je brojač veći od 10, break u liniji 14 završava petlju whi 1 e, a izvršenje programa pada na liniju 16, gde se štampaju rezultati.

Ovaj program radi, ali nije lep. Ovo je dobar primer korišćenja pogrešne alatke za posao. Ista stvar se može postići stavljanjem testa vrednosti promenljive brojač gde on i pripada - u whi l e uslov.

^**UPOZORENJE!** Beskonačne petlje, kao što je while (1), mogu prouzrokovati da Vaš kompjuter visi ako se izlazni uslov nikada ne dostigne. Koristite ih pažljivo i detaljno ih testirajte.

C+ + Vam nudi mnoge različite načine za ostvarenje istog zadatka. Pravo rešenje je izbor pravog alata za odredeni posao.

^**1** pazite Nemojte koristiti iskaz goto.

Koristite petlje whi 1 e za iteraciju dok je uslov istinit.

Obratite pažnju pri korisćenju iskaza continue i break.

Osigurajte da se Vaša petlja konačno završava.

Petlje do..while

Moguće je da se telo petlje while nikada neće izvršiti. Iskaz while proverava svoj uslov pre izvršenja svojih iskaza i, ako se uslov proračuna kao false, celokupno telo petlje whi le se preskače. Listing 7.6 ilustruje ovo.

Listing 7.6: **Preskakanje tela petlje whi 1 e**

```
1:    // Listing 7.6
2:    // Demonstrira preskakanje tela
3:    // petlje while kada je uslov neistinit.
4:
5:    linclude <iostream.h>
6:
7:    int main()
8:    {
9:       int counter;
```

nastavlja se

Listing 7.6: Preskakanje tela petlje whi 1 e

```
cout « "Koliko pozdrava?:
cin » counter;
while (counter > 0)
{
   cout « "Zdravo!\n";
   counter-;;
}
cout « "Brojac na izlazu: " « counter;
 return 0;
```

```
Koliko pozdrava?: 2
Zdravo!
Zdravo!
Brojac na izlazu: 0

Koliko pozdrava?: 0
Brojac na izlazu: 0
```

fflSJjfe* ^ korisnika s e zahteva početna vrednost u liniji 10. Ova početna vrednost se čuva u celobrojnoj promenljivoj brojač. Vrednost promenljive brojač se testira u liniji 12, a umanjuje u telu petlje whi 1 e. U prvom prolazu brojač je bio postavljen na 2, pa se, zbog toga, telo petlje while izvršilo dva puta. U drugom prolazu ipak, korisnik je ukucao 0. Vrednost promenljive brojač je testirana u liniji 12 i uslov je bio neistinit; brojač nije bio veći od 0. Celokupno telo petlje whi 1 e je bilo preskočeno, a Zdravo se nikada nije ni odštampalo.

Sta ako želite da osigurate da se Zdravo uvek štampa bar jednom? Petlja while ne može to ostvariti, zato što se i f uslov testira pre bilo kakvog štampanja. Možete prisiliti problem sa i f iskazom neposredno pre ulaska u whi 1 e:

```
if (counter < 1)   // prisi1 Java minimalnu vrednost
counter = 1;
```

to je ono što programeri nazivaju *kladž* (kludge), ružno i neelegantno rešenje.

do...while

Petlja do.. .while izvršava telo petlje pre testiranja njenog uslova i osigurava da se telo uvek izvrši bar jednom. Listing 7.7 popravlja listing 7.6 korišćenjem petlje do...while.

Listing 7.7: Demonstrira petlju do...whi 1 e

```
1:      // Listing 7.7
2       // Demonstrira do while
3
4       linclude <iostream.h>
5
```


nastavak

```
int  main()
{
   int  counter;
   cout « "Koliko  pozdrava?  ";
   cin  »  counter;
   do
   {
      cout «  "Zdravo\n";
      counter-;;
   }  while (counter >0 );
   cout « "Brojac je:  " « counter « endl;
    return 0;
```

```
     Koliko pozdrava? 2
^    Zdravo
     Zdravo
     Brojac je: 0
```

Od korisnika se traži početna vrednost u liniji 9, koja se čuva u celobrojnoj promenljivoj brojač. U petlji do...whi 1 e ulazi se u telo petlje pre testiranja uslova i zato je zagarantovano da će se telo petlje izvršiti najmanje jednom. U liniji 13 štampa se poruka, u liniji 14 brojač se umanjuje, a u liniji 15 testira se uslov. Ako uslov proračunava TRUE, izvršenje "skače" na vrh petlje u liniji 13; inače, pada na liniju 16.

Iskaz continue i break u petlji do...while rade potpuno isto kao i u petlji while. Jedina razlika izmedu petlje whi 1 e i petlje do...whi 1 e je kada se testira uslov.

Iskaz do. . .while

Sintaksa za iskaz do. . .while je sledeća:

```
do
iskaz
while (uslov)-,
```

Iskaz se izvršava, a onda se proračunava *uslov*. Ako je *uslov* TRUE, petlja se ponavlja; inače, petlja se završava. Iskazi i uslovi su identični petlji while.

Primer 1

```
// broji do 10
int x = 0;
do
cout « "X: " « x++;
while (x < 10)
```

Primer 2

```
// stampa alfabet malim slovima.
char ch = 'a';
do
{
COUt « Ch « ' ';
Ch++;
} while ( ch <= 'z' );
```

*| *PAHП* *W* Koristite do...whi 1 e kada želite da osigurate da se petlja izvrši bar jednom.

Koristite petlju whi 1 e, ako želite da preskočite petlju kada je uslov neistinit.

Testirajte sve petlje da biste bili sigurni da one rade ono što želite.

Petlje for

Prilikom programiranja petlji while, često ćete zateći sebe kako postavljate početni uslov, testirajući da li je uslov istinit i inkrementirajući, ili na drugi način menjajući promenljivu pri svakom prolasku kroz petlju. Listing 7.8 demonstrira ovo.

Listing 7.8: Ponovo ispitana petija whi 1 e

```
1   // Listing 7.8
2   // Upetljavanje sa while
3
4   linclude <iostream.h>
5
6   int main()
7   {
8     int counter = 0;
9
10     while(counter < 5)
11     {
12          counter++;
13          cout « "Upetljavanje!
14
15
16     cout « "\nBrojac: " « counter « ".\n";
17      return 0:
18  }
```

IIIIIJ2> uPetlJavanje! Upetljavanje! Upetljavanje! Upetljavanje! Upetljavanje'
Brojac: 5.

IIIмЛ& uslov se postavlja u liniji 8: brojac se inicijalizuje na 0. U liniji 10 brojac r-io TT • SeteStir3, da bi Se Videlo da lI je manJ od s. Broj ačsc inkrementira u liniji 12. U liniji 16 štampa se jednostavna poruka, ali možete zamisliti da su važnije stvari mogle biti uradene za svako inkrementiranje promenljive brojac.

Petlja for kombinuje tri koraka u jedan iskaz. Ta tri koraka su inicijalizacija, test i inkrementiranje. Iskaz for se sastoji od ključne red for, posle koje sledi par zagrada. Unutar zagrada se nalaze tri iskaza, koji su rastavljeni znakovima tačka-zarez.

Prvi iskaz je inicijalizacija. Ovde se može staviti bilo koji legalan C++ iskaz, ali se ovo, obično, koristi za kreiranje i inicijalizovanje promenljive, koja ima ulogu brojača. Iskaz 2 je test i ovde se može koristiti svaki legalan C++ izraz. To služi istoj ulozi kao i uslov u petlji while. Iskaz 3 je akcija. Obično se vrednost inkrementira, ili dekrementira, iako se ovde može staviti svaki legalan C++ iskaz. Uocite da iskazi 1 i 3 mogu biti bilo koji legalan C++ iskaz, ali iskaz 2 mora biti izraz - C++ iskaz koji vraća vrednost. Listing 7.9 demonstrira petlju loop.

Listing 7.9: Demonstriranje for petlje

```
// Listing 7.9
// Upetljavanje sa for

linclude <iostream.h>

int main()
{
  int counter;
  for (counter = 0; counter < 5; counter++)
     cout « "Upetljavanje! ";

   cout « "\nBrojac: " « counter « "An";
    return 0;
```

Brojac! Brojac! Brojac! Brojac!
Brojac: 5.

JTШJH^ Naredba for u liniji 8 kombinuje inicijalizaciju promenljive brojač, test da je brojač manji od 5 i inkrementiranje promenljive brojač, sve u jednoj liniji. Telo iskaza for je u liniji 9. Naravno, ovde se, takođe, može koristiti blok.

Iskaz for

Sintaksa za iskaz for je sledeća:

```
for (inicijalizacija; test; akcija )
iskaz;
```

Iskaz *inicijalizacija* se koristi za inicijalizovanje stanja promenljive brojač, ili da na drugi način ostvari pripremu za petlju; *test* je bilo koji C++ izraz i proračunava se pri svakom prolasku kroz petlju. Ako *test* ima vrednost TRUE, izvršava se *akcija* u zaglavlju (obično se brojač inkrementira), a onda se izvršava telo petlje for.

Primer 1

```
// stampa Zdravo deset puta
for (int i « 0; i<10; i++)
cout « "Zdravo!
```

Primer 2

```
for (int i = 0; i < 10; i++)
{
    cout « "Zdravo!" « endl;
    cout « "vrednost promenljive i je: " « i « endl;
)
```

Napredne petlje for

Iskazi for su moćni i fleksibilni. Tri nezavisna iskaza *(inicijalizacija,* test i akcija) dozvoljavaju nekoliko varijacija.

Petlja for radi u sledećoj sekvenci:

1. Izvršava operacije u inicijalizaciji.
2. Proračunava uslov.
3. Ako je uslov TRUE, izvršava petlju i akciju.

Posle svakog prolaza, petlja ponavlja korake 2 i 3.

Višestruka inicijalizacija i inkrementiranje

Nije neuobičajeno inicijalizovati vise od jedne promenljive, testirati složen logički izraz i izvrsiti vise od jednog iskaza. Inicijalizacija i akcija se mogu zameniti višestrukim C + + iskazima, od kojih je svaki odvojen zarezom. Listing 7.10 demonstrira inicijalizaciju i inkrementiranje dve promenljive.

Listing 7.10: Demonstriranje višestrukih iskaza u for petljama

```
1:  //listing 7.10
2:  // demonstrira visestruke iskaze u
3:  // for petljama
4:
5:  #include <iostream.h>
6:
7:   int main()
8:  {
9:       for (int i=0, j=0; i<3; i++, j++)
10:           cout « "i: " « i « " j: " « j « endl;
11:      return 0;
12: }
```



```
0 j: 0
1 j: 1
2 j: 2
```

.|-.ir:»M:jpjs. U liniji 9 dve promenljive, i i j , se inicijalizuju sa 0. Proračunava se test (i<3) i, zato što je istinit, izvršava se telo iskaza for i štampaju se vrednosti. Konačno, izvršava se treći deo u iskazu for, tako da se i i j inkrementiraju.

Pošto se završi linija 10, uslov se ponovo proračunava i, ako ostane istinit, akcije se ponavljaju (i i j se ponovo inkrementiraju), a telo petlje se ponovo izvršava. Ovo se nastavlja, sve dok test ne uspe; u torn slučaju, akcija se ne izvršava, a kontrola izlazi iz petlje.

Prazni iskazi u petljama for

Neki, ili svi iskazi u petlji for mogu biti prazni. Da biste ovo postigli, upotrebite znak tačka-zarez, da biste označili mesto gde bi bio iskaz. Da biste kreirali petlju for, koja se ponaša isto kao petlja while, izostavite prvi i treći iskaz. Listing 7.11 ilustruje ovu ideju.

Listing 7.11: Null iskazi u petljama for

```
1:    // Listing 7.11
2:    // For petlje sa null iskazima
3:
4:    linclude <iostream.h>
5:
6:    int mainQ
7:    {
8:        int counter = 0;
9:
10:       for( ; counter < 5; )
11:       (
12:          counter++;
13:          cout « "Upetljavanje!   ";
14:       }
15:
16:       cout « "\nBrojac: " « counter « ".\n";
17:       return 0;
18:   }
```

```
JEStefr Upetljavanje! Upetljavanje! Upetljavanje! Upetljavanje! Upetljavanje!
   ~    Brojac: 5.
```

Ш J Ш J ^ ^vo možete prepoznati kao petlju while, ilustrovanu u listingu 7.8! U liniji 8 inicijalizuje se brojač. Iskaz for u liniji 10 ne inicijalizuje nikakve vrednosti, ali uključuje test za brojač < 5. Nema iskaza za inkrementiranje, tako da se ova petlja ponaša isto kao da je bila napisana u obliku:

```
while (counter < 5)
```

C++ Vam daje nekoliko načina da postignete isti efekat. Ni jedan iskusan C++ programer ne bi koristio petlju for na ovaj način, ali ovo ilustruje fleksibilnost iskaza for. U stvari, moguće je, korišćenjem break i continue, kreirati petlju for, bez i jednog od tri moguća iskaza. Listing 7.12 ilustruje način.

Listing 7.12: Ilustracija praznih iskaza u petlji for

```
1:    //Listing 7.12 ilustrovanje
2:    //prazan for iskaz
3:
4:    linclude <iostream.h>
5:
6:    int main()
7:    {
8:        int counter=0;       //  inicijalizacija
9:        int max;
10:       cout « "Koliko pozdrava?";
11:       cin » max;
12:       for (;;)           // petlja for koja se ne zavrsava
13:       {
14:          if (counter < max)       // test
15:          {
16:            cout « "Zdravo!\n";
17:            counter++;          //  inkrement
18:          }
19:          else
20:              break;
21:       }
22:       return 0;
23:   }
```

```
Koliko pozdrava?3
Zdravo!
Zdravo!
Zdravo!
```

ШтЛр* Petlja *for* je sada dovedena do svoje apsolutne granice. Inicijalizacija, test i akcija su izvadeni iz iskaza for. Inicijalizacija se obavlja u liniji 8, pre početka petlje for. Test se obavlja u odvojenom iskazu if u liniji 14 i, ako test uspe, akcija, inkrement promenljive brojač, se izvršava u liniji 17. Ako test ne uspe, izlazak iz petlje se javlja u liniji 20.

Dok je ovaj program u nekoj meri apsurdan, postoje situacije kada je petlja for(;;) ih petlja while (1) upravo ono što ćete želeti. Videćete primer razumnije upotrebe takvih petlji kada se bude govorilo o iskazima switch, kasnije u ovom Danu.

Prazne petlje for

Toliko mnogo se može uraditi u zaglavlju iskaza for, ali postoje trenuci kada Vam neće biti potrebno da telo bilo šta uradi. U ovom slučaju, stavite prazan iskaz (;) u

telo petlje. Znak tačka-zarez može biti u istoj liniji kao i zaglavlje, ali ovo je lako prevideti. Listing 7.13 ilustruje kako koristiti prazno telo u petlji for.

Listing 7.13: Ilustracija praznog iskaza u petlji for

```
1:    //Listing 7.13
2:    //Demonstrira null iskaz
3:    // kao telo petlje for
4:
5:    linclude <iostream.h>
6:    int main()
7:
8:       for (int i = 0; i<5; cout « "i: " « i++ « endl)
9:
10:      return 0;
11:
```

```
i: 0
i: 1
i: 2
i: 3
i: 4
```

Petlja for u liniji 8 uključuje tri iskaza: inicijalizacioni iskaz kreira brojač i i inicijalizuje ga na 0. Uslovni iskaz testira i < 5, a iskaz akcije štampa vrednost iz i i inkrementira ga.

Ništa nije ostalo da se uradi u telu petlje for, pa se koristi prazan iskaz (;). Uočite da petlja nije dobro dizajnirana: iskaz akcije radi previse. Ovo bi bilo bolje prepraviti u

```
8:        for (int i = 0; i<5; i++)
9:             cout « "i: " « i « endl;
```

Iako oba rade potpuno isto, ovaj primer je lakši za razumevanje.

Ugnježdene petlje

Petlje mogu biti ugnježdene, da je jedna petlja smeštena u drugoj. Unutrašnja petlja će biti izvršena u potpunosti za svako izvršenje spoljašnje petlje. Listing 7.14 ilustruje zapisivanje oznaka u matrici, korišćenjem ugnježdenih petlji for.

Listing 7.14. Ilustruje ugnezdene for petlje.

```
//Listing 7.14
//Ilustruje ugnezdjene for petlje

linclude <iostream.h>

int mainQ
{
    int rows, columns;
```

nastavljase

Listing 7.14. Ilustruje ugneždene for petlje. *nastavak*

```
char theChar;
 cout « "Koliko redova? ";
 cin » rows;
 cout « "Koliko kolona? ";
 cin » columns;
 cout « "Koji karakter? ";
 cin » theChar;
 for (int i = 0; i<rows;
 {
    for (int j = 0; j<columns; j++)
        cout « theChar;
    cout « "\n";
 }
return 0;
```

```
Koliko redova? 4
Koliko kolona? 12
Koji karakter? x
xxxxxxxxxxxx
xxxxxxxxxxxx
xxxxxxxxxxxx
xxxxxxxxxxxx
```

Od korisnika se traži broj za rows i columns karakter za štampanje. Prva petlja for, u liniji 16, inicijalizuje brojač (i) na 0, a onda se izvršava telo spoljašnje petlje for.

U liniji 18, prvoj liniji tela spoljašnje petlje for, osnovana je druga petlja for. Drugi brojač (j) se takođe inicijalizuje na 0 i izvršava se telo unutrašnje petlje for. U liniji 19 štampa se izabrani karakter, a kontrola se vraća na zaglavlje unutrašnje petlje for. Uočite da se unutrašnja petlja for sastoji iz samo jednog iskaza (štampanje karaktera). Uslov se testira (j < columns) i, ako je proračunat kao true, j se inkrementira i štampa se sledeći karakter. Ovo se nastavlja sve dok j ne postane jednako broju kolona.

Pošto u unutrašnjoj petlji for ne uspe test, u ovom slučaju posle štampanja 12x-eva, lzvršenje pada na liniju 20 i štampa se nova linija. Spoljašnja petlja for se sada vraća na svoje zaglavlje, gde se testira njen uslov (i < rows). Ako se on proračuna da je true, i se inkrementira, a telo petlje se izvršava.

U drugoj iteraciji spoljašnje petlje for unutrašnja petlja for se ponovo startuje; j se reinicijalizuje na 0 i cela unutrašnja se ponovo izvršava.

Važna ideja ovde je da korišćenjem ugnježdene petlje, unutrašnja se izvršava za svaku iteraciju spoljašnje. Tako se karakter štampa columns puta za svaki red.

Kao sporedna finjenica, mnogi C++ programeri koriste slova i i j kao promenljive za brojanje. Ova tradicija se vraća do FORTRAN-a, u kome su slova i, j, k, l, m i n bila jedine legalne promenljive za brojanje.

Drugi programeri vise vole da koriste opisna imena promenljivih za brojanje, kao što su Ctrl 1 i Ctrl 2. Ipak, korišćenje i i j za zaglavlja petlje for ne bi trebalo da prouzrokuje mnogo konfuzije.

Opseg u petljama for

Zapamtićete da su promenljive ograničene na blok u kome su kreirane. To znači da je lokalna promenljiva vidljiva samo unutar bloka u kome je kreirana. Važno je da uočite da su promenljive za brojanje, kreirane u zaglavlju petlje for, ograničene spoljašnjim, a ne unutrašnjim blokom. Implikacija ovoga je da, ako imate dve petlje for u istoj funkciji, morate im dati različite brojačke promenljive, ili će se one medusobno mešati.

Sumiranje petlji

U Danu 5, "Funkcije", naučili ste kako da rešite Fibonaccijevu seriju problema, korišćenjem rekurzije. Da ukratko ponovimo: Fibonaccijeva serija počinje sa 1, 1, 2, 3, a svi naredni brojevi su zbir predhodna dva:

```
1,1,2,3,5,8,13,21,34...
```

n-ti Fibonaccijev broj je zbir Fibonaccijevih brojeva n-1 i n-2. Problem rešen u Danu 5 je bio pronalaženje n-tog Fibonaccijevog broja. Ovo je bilo obavljeno rekurzijom. Listing 7.15 nudi rešenje korišćenjem iteracije.

Listing 7.15: Rešavanje n-tog Fibonaccijevog broja korišćenjem iteracije

```
// Listing 7.15
// Demonstrates solving the nth Demonstrira resavanje n-tog
// Fibonaccijevog broja koriscenjem iteracije

#include <iostream.h>

typedef unsigned long int ULONG;

ULONG fib(ULONG position);
 int mainQ
 {
    ULONG answer, position;
    cout « "Koja je pozicija? ";
    cin » position;
    cout « "\n";

    answer = fib(position);
```

nastavlja se

Listing 7.15: Rešavanje n-tog Fibonaccijevog broja koriscenjem iteracije *nastavak*

```
18        cout « answer « " je
19        cout « position « ". Fibonaccijev broj.\n";
20       return 0;
21   }
22
23   ULONG fib(ULONG n)
24   {
25        ULONG minusTwo=1, minusOne=1, answer=2;
26
27        if (n < 3)
28             return 1;
29
30        for (n -= 3; n; n--))
31        {
32             minusTwo = minusOne;
33             minusOne = answer;
34             answer = minusOne + minusTwo;
35        }
36
37        return answer;
38
```

```
Koja je pozicija? 4

3 je 4. Fibonaccijev broj.

Koja je pozicija? 5

5 je 5. Fibonaccijev broj.

Koja je pozicija? 20

6765 je 20. Fibonaccijev broj.

Koja je pozicija? 100

3314859971 je 100. Fibonaccijev broj.
```

Listing 7.15 rešava Fibonaccijevu seriju, korišćenjem iteracije, umesto rekurzije. Ovaj pristup je brži i koristi manje memorije, nego rekurzivno rešenje.

U linji 13 od korisnika se traži pozicija za proveru. Poziva se funkcja `fib()`, koja proračunava poziciju. Ako je pozicija manja od 3, funkcija vraća vrednost 1. Počevši od pozicije 3, funkcija iterira, korišćenjem sledećeg algoritma:

1. Kreirati početnu poziciju: Popuniti promenljivu `answer` sa 2, `minusTwo` sa 1 i `minusOne` sa 1. Dekrementirati poziciju za 3, zato što se sa prva dva broja rukuje početnom pozicijom.
2. Za svaki broj, izbrojati Fibonaccijevu seriju. Ovo se radi sa
   a. stavljanjem vrednosti koja je trenutno u mi nusOne u mi nusTwo
   b. stavljanjem vrednosti koja je trenutno u answer u mi nusOne
   c. sabiranjem mi nusOne i mi nusTwo i stavljanjem zbira u answer
   d. dekrementiranjem n.
3. Kada n dostigne 0, vraća odgovor.

Ovo je način na koji biste rešili ovaj problem olovkom i papirom. Da je od Vas tražen peti Fibonaccijev broj, pisali biste:

```
1, 1, 2,
```

i razmislili, "Još dve stvari da se urade". Onda biste sabrali **2+1,** i napisali 3 i razmislili "još jedan da se pronađe". Konačno napisali biste **3+2** i odgovor bi bio **5.** U stvari, Vi pomerate Vašu pažnju udesno za jedan broj, pri svakom prolazu, i dekrementirate broj onih koje treba pronaci.

Uočite uslov koji je testiran u liniji 30 (n). Ovo je C++ idiom i potpuno je jednak n != 0. Ova petlja for se oslanja na činjenicu da će n kada dostigne 0, biti proračunat kao fal se, jer je 0 u C++ jednako neistini. Zaglavlje petlje for je moglo biti napisano:

```
for (n-=3; n>0; n++)
```

što bi možda bilo jasnije. Ipak, ovaj idiom je tako uobičajen u C++-U da ima malo smisla boriti se protiv njega.

Kompajlirajte, povežite i izvršite program, zajedno sa rekurzivnim rešenjem, koje je ponuđeno u Danu 5. Pokušajte da pronadete poziciju 25 i uporedite vreme koje je potrebno svakom programu. Rekurzija je elegantna, ali, zato što poziv funkcije donosi dodatne troškove prilikom izvođenja i zato što se poziva mnogo puta, njeno izvođenje je uočljivo sporije od iteracije. Mikrokompjuteri teže da budu optimizovani za aritmetičke operacije, tako da bi iteraciono rešenje trebalo da bude mnogo brže.

Pazite na to koliko veliki broj unosite; f i b raste brzo i velike celobrojne vrednosti će izazvati prekoračenje posle izvesnog vremena.

Iskazi switch

U Danu 4, videli ste kako da napišete i f i i f/el se iskaze. Ovo može postati prilično zbunjujuće kod veoma dubokog ugnježdavanja, pa C++ nudi alternativu. Za razliku od if, koji proračunava jednu vrednost, iskazi switch Vam dozvoljavaju grananje na bilo koju od nekoliko različitih vrednosti. Opšti oblik iskaza switch je:

```
switch (izraz)
{
case valueOne: iskaz;
```

```
                    break;
case valueTwo:  iskaz;
                    break;

case valueN:    iskaz;
                    break;
default:        iskaz;
}
```

izraz je bilo koji legalan C + + izraz, a *iskazi* su bilo koji legalni C + + iskazi, ili blokovi iskaza; `switch` proračunava izraz i poredi rezultat sa svakom `case` vrednošću. Uočite, ipak, da je proračun samo za jednakost; ni ovde ne mogu `biti` korišćeni rela-**rroni.** ni Boolean operatori.

Alio se jedna od `case` vrednosti podudara sa izrazom, izvršenje "skače" na `te` iskaze i nastavlja se do kraja `switch` bloka, dok se ne susretne iskaz break. Ako se ništa ne podudara, izvršenje se grana na opcioni default iskaz. Ako nema podrazumevane vrednosti i vrednosti koja se podudara, izvršenje prolazi kroz iskaz `switch` i iskaz se završava.

||^**АИШМ**||p Skoro uvek je dobra ideja imati default (engl. default = podrazumevani) slučaj u iskazima `switch`. Ako nemate drugu potrebu za default, upotrebite ga za testiranje prerpostavljeno nemogućeg slučaja i štampanje poruke o grešci; ovo može biti izvanredna pomoć u dibagiranju.

Važno je da uočite da će, ako nema iskaza break na kraju iskaza `case`, izvršenje pasti na sleded iskaz `case`. Ovo je, ponekad, neophodno, ali je, obično, greška. Ako odlučite da dozvolite da izvršenje padne, osigurajte komentar, pokazujući da niste zaboravili break.

Listing `7.16`. ilustruje upotrebe iskaza `switch`.

Listing 7.16: Demonstriranje switch iskaza.

```
//Listing 7.16
// Demonstrira switch iskaz

linclude <iostream.h>

int main()
{
  unsigned short int number;
  cout « "Unesite broj izmedju 1 i 5: ";
   cin » number;
   switch (number)
   {
      case 0:    cout « "Jako mali, zao mi je!";
                 break;
      case 5:   cout « "Dobar posao!\n";  // propada
      case 4:   cout « "Dobar pik!\n"; // propada
      case 3:   cout « "Izvanredno!\n"; // propada
      case 2:   cout « "Majstorski!\n"; // propada
      case 1:   cout « "Neverovatno!\n";
                break;
      default:  cout « "Prevelik!\n";
                break;
  }
   cout « "\n\n";
   return 0;
}
```

```
Unesite broj izmedju 1 i 5: 3
Izvanredno!
Majstorski!
Neverovatno!

Unesite broj izmedju 1 i 5: 8
Prevelik!
```

Od korisnika se traži broj, koji se daje iskazu `switch`. Ako je broj `0`, iskaz `case` u liniji `13` se podudara, štampa se poruka `Previše mali, žao mi je!`, a iskaz `break` završava prekidač (engl. switch = prekidač). Ako je vrednost `5`, izvršenje se prebacuje na liniju `15`, gde se štampa poruka, a, onda, " propada" do linije `16`, štampa se druga poruka i tako dalje, do udaranja u `break` u liniji `20`.

Efekat mreže ovih iskaza je za broj izmedu `1` i `5` takav da se štampaju mnoge poruke. Ako vrednost broja nije između `0` i `5`, smatra se prevelikom i poziva se default iskaz iz linije `21`.

Iskaz switch

Sintaksa za iskaz switch je sledeća:

```
switch (izraz)
{
case valueOne:  iskaz;
case valueTwo:  iskaz;

case valueN:  iskaz
default:  iskaz;
}
```

Iskaz `switch` dozvoljava grananje na višestruke vrednosti *izraza*. Izraz se proračunava i, ako se podudara sa nekom od `case` vrednosti, izvršenje "skače" na tu liniju. Izvršenje se nastavlja ili do kraja iskaza `switch`, ili do susretanja iskaza `break`.

Ako se *izraz* ne podudara ni sa jednim `case` iskazom i ako postoji iskaz default, izvršenje se prebacuje na iskaz default; inače se iskaz `switch` završava.

Primer 1

```
switch (choice)
{
case 0:
       cout « "Nula!" « endl;
       break
case 1:
       cout « "Jedan!" « endl;
      break;
case 2:
      cout « "Dva!" « endl;
default:
      cout « "Podrazumevano!" « endl;
}
```

Primer 2

```
switch (choice)
{
choice 0:
choice 1:
choice 2:
      cout « "Manje od 3!";
      break;
choice 3:
      cout « "Jednako 3!";
      break;
default:
      cout « "Vece od 3!";
}
```

Korišćenje iskaza switch sa menijem

Listing 7.17 se vraća na petlju for (;;) koja je ranije opisana. Ove petlje se, takođe, nazivaju večne, zato što će one večno "petljati" ako se ne susretne `break`. Večna petlja se koristi za postavljanje menija, zahtevanje izbora od korisnika, delovanje po izboru, a onda za povratak u meni. Ovo će se nastaviti, sve dok korisnik ne izabere izlazak.

^**MAPOMENA**^ Neki programeri vole da pišu

```
Idefine EVER ;;
for (EVER)
{
    // iskazi...
}
```

Korišćenje #define je "pokriveno" u Danu 17, "Pretprocesor".

Beskonačna petlja je petlja koja nema izlazni uslov. Da bi se izašlo iz petlje, mora se koristiti iskaz `break`.

Listing 7.17: Demonstriranje vecne petlje.

```
//Listing 7.17
//Using a forever loop to manage Koriscenje vecne petlje za upravljanje
//korisnickom interakcijom
linclude <iostream.h>

// tipovi i definisanja
enum BOOL { FALSE, TRUE };
typedef unsigned short int USHORT;

// prototipovi
USHORT menu();
void DoTask0ne();
void DoTaskMany(USHORT);

int main()


    BOOL exit = FALSE;
    for (;;)
    {
        USHORT choice = menu();
        switch(choice)
        {
            case (1):
                DoTask0ne();
                break;
            case (2):
                DoTaskMany(2);
                break;
            case (3):
                DoTaskMany(3);
                break;
            case (4):
                continue;  // suvisno!
                break;
            case (5):
                exit=TRUE;
                break;
            default:
                cout « "Molim Vas, izaberite povovo!\n";
                break;
        }          // kraj switch-a

        if (exit)
            break;
                // kraj vecne petlje
return 0;
                // kraj mainQ
```

nastavlja se

Listing 7.17: Demonstriranje večne petlje.

```
49
50   USHORT menu()
51   {
52       USHORT choice;
53
54       cout « " **** Meni ****\n\n";
55       cout « "(1) Izbor jedan.\n";
56       cout « "(2) Izbor dva.\n";
57       cout « "(3) Izbor tri.\n";
58       cout « "(4) Ponovo prikazi meni.\n"
59       cout « "(5) Izlaz.\n\n";
60       cout « ": ";
61       cin » choice;
62       return choice;
63
64
65   void DoTask0ne()
66   {
67       cout « "Zadatak jedan!\n";
68   }
69
70   void DoTaskMany(USHORT which)
71   {
72       if (which == 2)
73           cout « "Zadatak dva!\n";
74       else
75           cout « "Zdatak Tri!\n";
76
```

```
**** Meni ****

(1) Izbor jedan.
(2) Izbor dva.
(3) Izbor tri.
(4) Ponovo prikazi meni.
(5) Izlaz.

: 1
Zadatak jedan!
 **** Meni ****
(1) Izbor jedan.
(2) Izbor dva.
(3) Izbor tri.
(4) Ponovo prikazi meni.
(5) Izlaz.

: 3
Zadatak tri!
 **** Meni ****
```


nastavak

```
(1) Izbor jedan.
(2) Izbor dva.
(3) Izbor tri.
(4) Ponovo prikazi meni.
(5) Izlaz.

: 5
```

Ovaj program objedinjava nekoliko koncepata iz ovog i prethodnih Dana. On, takode, pokazuje uobičajenu upotrebu iskaza switch. U liniji 7 enumeracija, BOOL, se kreira sa dve moguće vrednosti: FALSE, što je jednako 0, kao što i treba, i TRUE, što je jednako 1. U liniji 8 koristi se typedef za kreiranje alijasa, USHORT, za unsigned short int.

Beskonačna petlja počinje u liniji 19. Poziva se funkcija menu(), koja štampa meni na ekranu i vraća korisnikov izbor. Iskaz switch, koji počinje u liniji 22, a završava se u liniji 42, prelazi na korisnikov izbor.

Ako korisnik unese 1, izvršenje "skače" na iskaz case 1: u liniji 24. Linija 25 prebacuje izvršenje na funkciju DoTaskOneO, koja štampa poruku, a zatim se vraća. Na svom povratku, izvršenje nastavlja na liniji 26, gde break završava iskaz switch, a izvršenje propada do linije 43. U liniji 44 proračunava se promenljiva exit. Ako proračun daje true, prekid u liniji 45 će se izvršiti, a petlja for(;;) će se završiti, ali ako proračun daje false, izvršenje se nastavlja na vrhu petlje u liniji 19.

Uočite da je iskaz continue u liniji 34 suvišan i kada bi ona bila izostavljena i kada bi se naišlo na iskaz break, switch bi se završila, exit bi bilo proračunato kao FALSE, petlja bi reiterirala, a meni bi bio ponovo odštampan; continue, ipak, preskače testiranje exit.

<j| MXKE ^ Upotrebite iskaze swi tch da biste izbegli duboko ugnježdene iskaze.

Nemojte zaboraviti break na kraju svakog case, osim ako želite propadanje.

Napravite pazljivu dokumentaciju svih namernih "propadajuah" case.

Stavite podrazumevani case u iskaze switch, bar samo da detektujete prividno nemoguće situacije.

Rezime

Postoje različiti načini da se prouzrokuje upetljavanje C++ programa. Petlje while proveravaju uslov i, ako je istinit, izvršavaju iskaze u telu petlje. Petlje do.. .while izvršavaju telo petlje, a, onda, testiraju uslov. Petlje for inicijalizuju vrednost, pa testiraju izraz. Ako je izraz istinit, izvršava se poslednji iskaz u zaglavlju petlje for, kao i telo petlje. U svakom narednom prolazu izraz se ponovo testira.

Iskaz goto se, generalno, izbegava, jer prouzrokuje bezuslovan skok na prividno neodredenu lokaciju u kodu i time čini izvorni kod težak za razumevanje i održava-

nje; **continue** prouzrokuje da petlje **whi** le, do.. **.whi** le i for ponovo počnu, a break da **se** iskazi **whi 1 e,** do.. **.whi 1** e, for **i swi tch** završe.

Pitanja i odgovori

P **Kako izabrati izmedu if/else i switch?**

O Ako postoji vise od jednog, **ili** dva else dela, a svi testiraju istu vrednost, razmotrite korišćenje iskaza **switch.**

P **Kako izabrati između while i do.. .while?**

O Ako bi trebalo da se telo petlje uvek izvrši najmanje jednom, razmotrite petlju do.. **.whi** le; inače, pokušajte da upotrebite petlju **whi** le.

P **Kako izabrati između while i for?**

O Ako inicijalizujete promenljivu za brojanje, testirate tu promenljivu i inkrementirajte je pri svakom prolazu kroz petlju i razmotrite petlju for. Ako je Vaša promenljiva već inicijalizovana i ne inkrementira se u svakoj petlji, petlja **whi** le može biti bolji izbor.

P **Kako izabrati između** rekurzije **i** iteracije?

O Pri rešavanju nekih problema je veoma potrebna rekurzija, ali većina problema će "popustiti" pred iteracijom. Stavite rekurziju u zadnji džep; može jednom biti korisna.

P **Da li je bolje** koristiti **while (1), ili for(;;)?**

O Nema značajne razlike.

Radionica

Nudimo Vifei test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Kako da inicijalizujem vise od jedne promenljive u petlji for?
2. Zašto se izbegava goto?
3. Da li je moguće napisati petlju for sa telom koje se nikada ne izvršava?
4. Da li je moguće ugnjezditi petlje while unutar petlji for?
5. Da li je moguće kreirati petlju koja se nikada ne završava? Dajte primer.
6. Šta se dešava ako kreirate petlju koja se nikada ne završava?

Vežbe

1. Koja je vrednost x po završetku petlje for?

```
for (int x = 0; x < 100; x++)
```

2. Napišite ugneždenu petlju for koja štampa šablon 10x10, ispunjen znakom 0.
3. Napišite iskaz for koji će brojati od 100 do 200 sa korakom 2.
4. Napišite petlju **whi 1** e koja će brojati od 100 do 200 sa korakom 2.
5. Napišite petlju do**...whi 1** e koja će brojati od 100 do 200 sa korakom 2.
6. ISTERIVAČ BAGOVA: Šta nije u redu sa ovim kodom?

```
int counter = 0
while (counter < 10)
{
    cout « "brojac: " « counter;
}
```

7. ISTERIVAČ BAGOVA: Šta nije u redu sa ovim kodom?

```
for (int counter = 0; counter < 10; counter++);
     cout « counter « " ";
```

8. ISTERIVAČ BAGOVA: Šta nije u redu sa ovim kodom?

```
int counter = 100;
while (counter < 10)
{
    cout « "brojac sada: " « counter;
    counter-;;
}
```

9. ISTERIVAČ BAGOVA: Šta nije u redu sa ovim kodom?

```
cout « "Unesite broj izmedju 0 i 5: ";
cin » theNumber;
switch (theNumber)
{
   case 0:
         doZero();
   case 1:              // propada
   case 2               // propada
   case 3               // propada
   case 4               // propada
   case 5
         doOneToFive();
```

```
            break;
        default:
            doDefault();
            break;
    }
```

Pregled sadržaja

Listing PS1.1: Sedmica 1. U pregledu sadriaja

```
#include <iostream.h>

typedef unsigned short int USHORT;
typedef unsigned long int ULONG;
enum BOOL { FALSE, TRUE};
enum CHOICE { DrawRect = 1, GetArea,
   GetPerim, ChangeDimensions, Quit};
// Deklaracija klase pravougaonika
class Rectangle
{
   public:
     //  konstruktori
     Rectangle(USH0RT width, USHORT height);
     -Rectangle();

     // metode pristupa

     USHORT GetHeightO const { return itsHeight; }
     USHORT GetWidth() const { return itsWidth; }
     ULONG GetAreaO const { return itsHeight * itsWidth; }
     ULONG GetPerimO const { return 2*itsHeight + 2*itsWidth;
     void SetSize(USHORT newWidth, USHORT newHeight);

     // Razlicite metode
     void DrawShapeQ const;
```

nastavljase

Listing PS1.1: Sedmica 1. U pregledu sadriaja

nastavak

```
26      private:
27        USHORT itsWidth;
28        USHORT itsHeight;
29
30
31  // Implementacije metoda klase
32  void Rectangle::SetSize(USHORT newWidth, USHORT newHeight)
33  {
34     itsWidth = newWidth;
35     itsHeight = newHeight;
36
37
38
39  Rectangle::Rectangle(USH0RT width. USHORT height)
40  {
41     itsWidth = width;
42     itsHeight = height;
43
44
45  Rectangle::-Rectangle() {}
46
47  USHORT DoMenu();
48  void DoDrawRect(Rectangle);
49  void DoGetArea(Rectangle);
50  void DoGetPerim(Rectangle);
51
52  void main ()
53  {
54     // inicijalizuje pavougaonik na 30,5
55     Rectangle theRect(30,5);
56
57     USHORT choice = DrawRect;
58     USHORT fQuit = FALSE;
59;
60     while (!fQuit)
61     {
62       choice = OoMenu();
63       if (choice < DrawRect || choice >  Quit)
64       {
65         cout « "\n Nevažeći Izbor, molim vas pokušajte ponovo.\n\n"
66         conti nue;
67       }
68       switch (choice)
69       {
70       case  DrawRect:
71         DoDrawRect(theRect);
72         break;
73       case GetArea:
74         DoGetArea(theRect);
75         break;
76       case GetPerim:
77         DoGetPerim(theRect);
78         break;
79       case ChangeDimensions:
80         USHORT newLength, newWidth;
81         cout « "\n Nova širina ";
82         cin » newWidth;
83         cout « "Nova visina: ";
84         cin » newLength;
85         theRect.SetSize(newWidth, newLength);
86         DoDrawRect(theRect);
87         break;
88       case Quit:
89         fQuit = TRUE;
90         cout «  "\nIzlazak...\n\n";
91         break;
92       default:
93         cout « "Greska u izboru!\n";
94         fQuit = TRUE;
95         break;
96       }    // kraj switch
97     }      // kraj while
98  }       // kraj main
99:
100:
101: USHORT DoMenu()
102: {
103:   USHORT choice;
104:    cout « "\n\n    *** Meni *** \n";
105:    cout « "(1) Nacrtaj pravougaonik\n";
106:    cout « "(2) Povrsinu\n";
107:    cout « "(3) 0bim\n";
108:    cout « "(4) Pomeni velicinu\n";
109:    cout « "(5) Izlaz\n";
110:
111:  cin » choice;
112:  return choice;
113: }
114:
115: void DoDrawRect(Rectangle theRect)
116: {
117:   USHORT height = theRect.GetHeightQ;
118:   USHORT width = theRect.GetWidth();
119:
120:   for (USHORT i = 0; i<height; i++)
121:   {
122:     for (USHORT j = 0; j< width; j++)
123:       cout «  "*";
124:     cout « "\n";
```

nastavljase

Listing PS1.1: Sedmica 1. U pregiedu sadriaja

nastavak

```
125     }
126
127
128
129  void  DoGetArea(Rectangle  theRect)
130  {
131    cout « "Povrsina:  " «    theRect.GetAreaO  « endl;
132  }
133
134  void  DoGetPerim(Rectangle  theRect)
135  {
136    cout « "Obim:  " «    theRect.GetPerim()  « endl;
137  }
```

```
*** Meni  ***
(1) Draw  Rectangle
(2)  Povrsinu
(3) Obim
(4) Promeni  velicinu
(5) Izlaz
1
******************************
******************************
******************************
******************************
******************************

    ***  Meni  ***
(1) Nacrtaj  pravougaonik
(2)  Povrsinu
(3) Obim
(4) Promeni  velicinu
(5) Izlaz
2
Povrsina:  150

    ***  Meni  ***
(1) Nacrtaj  pravougaonik
(2)  Povrsinu
(3) Obim
(4) Promeni velicin
(5) Izlaz
3
Obim:  70

    *** Meni  ***
(1) Nacrtaj  pravougaonik
(2)  Povrsinu
(3) Obim
(4) Promeni  velicinu
(5) Izlaz
4

Nova sirina:  10
Nova visina:  8
**********
**********
**********
**********
**•*••*•*•
**********
**********
**********

***  Meni  ***
(1) Nacrtaj  pravougaonik
(2)  Povrsinu
(3) Obim
(4) Promeni  velicinu
(5) Izlaz

Povrsina:  80

    ***  Meni  ***
(1) Nacrtaj  pravougaonik
(2)  Povrsinu
(3) Obim
(4) Promeni  velicinu
(5) Izlaz
3
Obim:  36

    ***  Meni  ***
(1) Nacrtaj  pravougaonik
(2)  Povrsinu
(3) Obim
(4) Promeni  velicinu
(5) Izlaz
5

Izlazak..
```

Ovaj program koristi većinu veština koje ste naučili u ovoj nedelji. Ne samo da bi trebalo da budete sposobni da unesete, kompajlirate, povežete i izvršite program, nego i da razumete šta i kako radi, bazirajući se na radu koji ste ostvarili u ovoj nedelji.

Prvih šest linija kreiraju nove tipove i definicije koji će biti korišćeni kroz ceo program.

Linije 9-29 deklarišu klasu Rectangle. Postoje javni metodi pristupa za dobijanje i postavljanje širine i visine pravougaonika, kao i za proračunavanje površine i obima. Linije 32-43 sadrže definicije funkcija klase, koje nisu deklarisane sa i nl i ne.

Prototipovi funkcija, za funkcije nečianice klase, su u linijama 47-50, a program počinje u linji 52. Suština ovog programa je generisanje pravougaonika, a onda štampanje menija koji nudi pet opcija: crtanje pravougaonika, odredivanje njegove površine, odredivanje njegovog obima, promena velidne pravougaonika, i izlaz.

Zastavica se postavlja u liniji 58, a kada nije postavljena na TRUE petlja, menija se nastavlja. Zastavica se postavlja na TRUE, ako korisnik izabere Izlaz (Quit) iz menija.

Svaki drugi izbor, sa izuzetkom ChangeDimensions, poziva funkciju. Ovo čini iskaz switch čistijim. ChangeDimensions ne može pozvati funkciju, zato što on mora promeniti dimenzije pravougaonika. Da je pravougaonik predat (po vrednosti) funkciji, kao što je DoChangeDi mens ions (), dimenzije bi bile promenjene za lokalnu kopiju pravougaonika u DoChangeDimensionsQ a ne za pravougaonik u main(). U Danu 8., "Pokazivači", i Danu 10., "Napredne funkcije", naučićete kako da nadvladate ovu restrikciju, ali za sada je napravljena promena u funkciji mai n().

Uočite kako upotreba enumeracije čini iskaz switch jasnijim i lakšim za razumevanje. Da je switch zavisila od numeričkih izbora (1-5) korisnika, Vi biste morali da se konstantno upućujete na opis menija da biste videli koji je koji izbor.

U liniji 63 ispituje se korisnikov izbor, da bi se proverilo da je u opsegu. Ako nije, štampa se poruka i ponovo se štampa meni. Uočite da iskaz switch uključuje "nemoguće" podrazumevano stanje. Ovo je pomoć u dibagiranju. Ako program radi, do tog iskaza se nikada ne može doći.

Pregled sadriaja

Cestitamo! Završili ste prvu nedelju! Sada možete kreirati i razumeti sofisticirane C + + programe. Naravno, ima još mnogo vise da se uradi i sledeća nedelja počinje jednim od najtežih koncepata u C+ + -u: pokazivačima. Nemojte sada odustati, Vi ste u prilici da prodrete duboko u značenje i korišćenje objektno-orijentisanog programiranja, virtualnih funkcija i mnogih naprednih karakteristika ovog moćnog jezika.

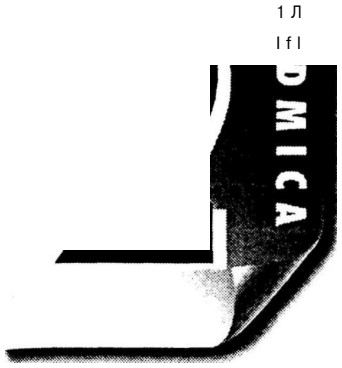
Odmorite se, sunčajte se u slavi Vašeg dostignuća, a onda okrenite stranicu, da biste počeli Nedelju 2.

Na prvi pogled

Završili ste prvu nedelju učenja programiranja u C++-U. Do sada bi trebalo da se osećate udobno, unoseći programe, koristeći Vaš kompajler i razmišljajući o objektima, klasama, i programskom toku.

Gde idete

Sedmica 2 počinje pokazivačima. Pokazivači su tradicionalno težka tema za nove C+ + programere, ali videćete da su oni opisani potpuno i jasno, i ne bi trebalo da budu blok za spoticanje. Dan 9, "Reference", uči o referencama, koje su bliski rodaci pokazivačima. Dan 10, "Napredne funkcije", videćete kako da preklopite funkcije, a u Danu 11, "Nizovi", naučićete kako da radite sa nizovima i kolekcijama. Dan 12, "Nasledivanje", uvodi nasleđivanje, fundamentalni koncept u objektno-orijentisanom programiranju. Dan 13, "Polimorfizam", proširuje lekcije iz Dana 12. da bi objasnio polimorfizam, i Dan 14, "Specijalne klase i funkcije", završava nedelju diskusijom o statičkim funkcijama i prijateljima.



Dan 8

Pokazivad

Jedan od najmoćnijih alata koji su na raspolaganju C+ + programeru je pokazivač, kojim se manipuliše memorija kompjutera direktno. Danas ćete naučiti

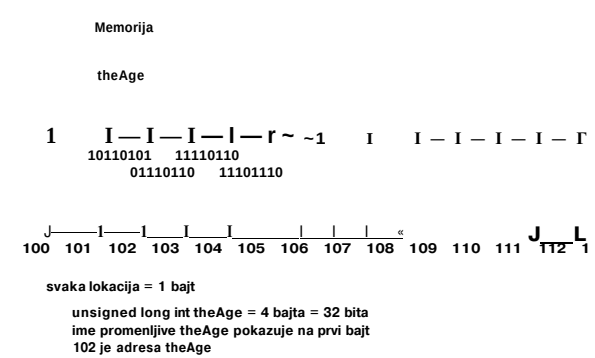
- šta su pokazivači
- kako deklarisati i koristiti pokazivače
- šta je slobodno skladište i kako manipulisati memorijom.

Pokazivači predstavljaju dva specijalna izazova pri učenju C++-a. Oni mogu biti, u izvesnoj meri, zbunjujući, i nije odmah očigledno zašto su potrebni. Uovom poglavlju biće objašnjeno kako pokazivači rade, korak, po korak. Potpuno ćete razumeti potrebu za pokazivačima u narednim poglavljima.

Šta je pokazivač?

Pokazivač je promenljiva koja čuva memorijsku adresu.

Da biste razumeli pokazivače, morate znati ponešto o kompjuterskoj memoriji. Kompjuterska memorija je podeljena na sekvencijalno označene memorijske lokacije. Svaka promenljiva se locira na jedinstvenoj lokaciji u memoriji, poznatoj kao njena adresa. (ovo je opisano u sekciji "Dodatno objašnjenje", koja sledi posle Dana 5, "Funkcije"). Slika 8.1 prikazuje šematsku reprezentaciju skladištenja `unsigned long` celobrojne promenljive `theAge`.



Slika 8.1. Šematska reprezentacija promenljive theAge.

Različiti kompjuteri označavaju ovu memoriju, koristeći različite, kompleksne seme. Obično nije potrebno da programeri znaju odredenu adresu neke date promenljive, zato što kompajler rukuje detaljima. Ipak, ako želite ovu informaciju, možete upotrebiti operator adresa od (&), što je ilustrovano u listingu 8.1.

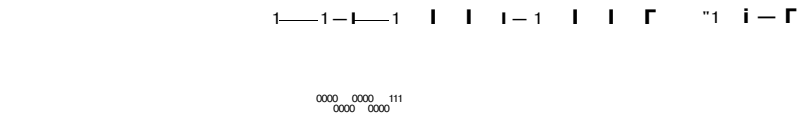
Listing 8.1: Demonstriranje adresa od promenljivih.

```
1:  // Listing 8.1 Demonstrira operator adresa od
2:  // i adrese lokalnih promenljivih
3:
4:  linclude <iostream.h>
5:
6:  int main()
7:  {
8:     unsigned short shortVar=5;
9:     unsigned long  longVar=65535;
10:    long sVar = -65535;
11:
12:    cout « "shortVar:\t" « shortVar;
13:    cout « " Adresa shortVar:\t";
14:    cout «  &shortVar  « "\n";
15:
16:    cout « "longVar:\t"  « longVar;
17:     cout « " Adresa longVar:\t" ;
18:     cout «  &longVar  « "\n";
19:
20:    cout « "sVar:\t"      « sVar;
21:      cout     « " Adresa sVar:\t" ;
22:      cout     «  &sVar     « "\n";
23:
24: return 0;
25: }
```

```
shortVar: 5       Adresa shortVar:  0x8fc9:fff4
longVar:  65535   Adresa longVar:   0x8fc9:fff2
sVar:    -65535   Adresa  sVar:     0x8fc9:ffee
```


Tri promenljive se deklarišu i inicijalizuju: short u liniji 8, unsigned **1** ong u liniji 9 i long u liniji 10. Njihove vrednosti i adrese se štampaju u linijama 12-16, korišćenjem operatora adresa od (&).

Vrednost promenljive shortVar je 5, kao što je i očekivano, a njena adresa je 0x8fc9:fff4 kada se izvršava na mom 80386-baziranom kompjuteru. Ova komplikovana adresa je specifična za kompjuter i može se malo promeniti svaki put kada se pokrene program. Vaši rezultati će biti drugačiji. Ono što se, ipak, ne menja je to da je razlika izmedu prve dve adrese dva bajta, ako Vaš kompjuter koristi dvobajtne short celobrojne vrednosti. Razlika između drugog i trećeg je četiri bajta, ako vaš kompjuter koristi četvorobajtne long celobrojne vrednosti. Slika 8.2 ilustruje kako bi promenljive u ovom programu bile čuvane u memoriji.



Slika 8.2. Ilustracija čuvanja promenljivih

Nema razloga da znate stvarnu numeričku vrednost adrese svake promenljive. Ono o čemu Vi brinete je da svaka ima adresu i da je odgovarajuća količina memorije rezervisana. Vi, deklarisanjem tipa promenljive, saopštavate kompajleru koliko memorije da za nju rezerviše; kompajler joj, automatski, dodeljuje adresu. Na primer, `long` celobrojna vrednost je obično četiri bajta, što znači da promenljiva ima adresu za četiri bajta memorije.

Cuvanje adrese u pokazivaču

Svaka promenljiva ima adresu. Čak i bez znanja specifične adrese date promenljive, Vi možete čuvati tu adresu u pokazivaču.

Na primer, pretpostavimo da je `howOld` celobrojna promenljiva. Da biste deklarisali pokazivač, nazvan `pAge`, za čuvanje njene adrese, napisali biste

```
int *pAge = 0
```

Ovo deklariše `pAge` kao pokazivač na `int` što znači da se `pAge` deklariše za čuvanje adrese od `int`.

Uočite da je `pAge` promenljiva, kao i bilo koja druga. Kada deklarišete celobrojnu promenljivu (tipa `int`), ona se kreira za čuvanje celobrojne vrednosti. Kada deklarišete pokazivačku promenljivu, kao što je `pAge`, ona se kreira za cuvanje adrese; `pAge` je samo drugi tip promenljive.

U ovom primeru, pAge se inicijalizuje na nula. Pokazivač, čija je vrednost nula, naziva se null pokazivač. Svi pokazivači, kada se kreiraju, trebalo bi da budu inicijalizovani na nešto. Ako ne znate šta želite da dodelite pokazivaču, dodelite 0. Pokazivač koji nije inicijalizovan se naziva *divlji pokazivač.* Divlji pokazivači su veoma opasni.

yHAPOЛИМА^ Praktikujte sigurno pisanje programa: intcijalizujte Vaše pokazivače!

Ako inicijalizujete pokazivač na 0, morate specifično dodeliti adresu od howOld pokazivaču pAge. Evo primera koji pokazuje kako to da uradite:

```
unsigned short int howOld =50;        // kreiraj promenljivu
unsigned short int * pAge = 0;        // kreiraj pokazivač
pAge = &how01d;                      // stavi adresu od howOld u pAge
```

Prva linija kreira promenljivu - howOld, čiji je tip unsigned short int, i inicijalizuje je na vrednost 50. Druga linija deklariše pAge kao pokazivač na tip unsigned short int i inicijalizuje ga na nula. Znate da je pAge pokazivač zbog asteriska (*), posle tipa promenljive, a pre imena promenljive.

Treća i poslednja linija dodeljuju adresu promenljive howOl d pokazivaču pAge. Možete red da se adresa promenljive howOl d dodeljuje zbog operatora adresa od (&). Da operator adresa od nije bio korišćen, bila bi dodeljena vrednost promenljive howOld. To bi, možda, bila važeća adresa.

U ovom trenutku, pAge ima kao svoju vrednost adresu promenljive howOld, a howOld, dalje, ima vrednost 50. Ovo ste mogli postići jednim korakom manje, kao u

```
unsigned short int howOld = 50;        //  kreiraj  promenljivu
unsigned short int * pAge = &how01d;   // kreiraj pokazivač na howOld
```

pAge je pokazivač koji sada sadrži adresu promenljive howOld. Korišćenjem pAge, možete odrediti vrednost promenljive howOld, koja je, u ovom slučaju, 50. Pristupanje promenljivoj howOld, korišćenjem pokazivača pAge naziva se indirekcija, jer Vi indirektno pristupate promenljivoj howOld preko pAge. Kasnije, u ovom Danu, videćete kako da koristite indirekciju za pristup vrednosti promenljive.

> *Indirekcija* znači pristupanje vrednosti, preko adrese, koja se čuva u pokazivaču. Pokazivač obezbeđuje indirektan način za dobijanje vrednosti, koja se čuva na toj adresi.

Imena pokazivača

Pokazivači mogu imati bilo koje ime legalno za druge promenljive. Ova knjiga prati konvenciju imenovanja svih pokazivača sa inicijalnim 44, kao u pAge ili pNumber.

Operator indirekcije

Operator indirekcije (*) se, takode, naziva operator dereferenciranja. Kada se pokazivač dereferencira, vraća se vrednost na adresu koja se čuva u pokazivaču.

Normalne promenljive obezbeduju direktan pristup svojiim vrednostima. Ako kreirate novu promenljivu tipa unsigned short int, nazvanu yourAge, i želite da dodelite vrednost iz howOld toj novoj promenljivoj,

```
unsigned short int yourAge;
yourAge = howOld;
```

Pokazivač obezbeđuje indirektan pristup vrednosti promenljive čiju adresu čuva. Da biste dodelili vrednost iz howOl d novoj promenljivoj yourAge, korišćenjem pokazivača pAge, pišete

```
unsigned short int yourAge;
yourAge = *pAge;
```

Operator indirekcije (*) ispred promenljive pAge znači "vrednost koja se čuva na". Ova dodela govori: "Uzmi vrednost koja se čuva na adresu u pAge i dodeli je promenljivoj yourAge."

^NAFOMEHA^ Operator indirekcije (*) se koristi na dva važna načino sa pokazivačima: za deklaraciju i za dereferenciranje. Kada se pokazivač deklariše, zvezdica pokazuje da je to pokazivač, a ne obična promenljiva. Na primer,

```
unsigned short * pAge = 0; // kreiraj pokazivač na neoznačenu kratku
```

Kada se pokazivač dereferencira, operator indirekcije pokazuje da će se pristupiti vrednosti na memorijskoj lokaciji, koja se čuva u pokazivaču, a ne na samoj adresi.

```
*pAge = 5; // dodeljuje 5 vrednosti na pAge
```

Takode uočite da se ovaj isti karakter (*) koristi kao operator množenja. Kompajler zna koji operator da pozove, bazirajud se na kontekstu.

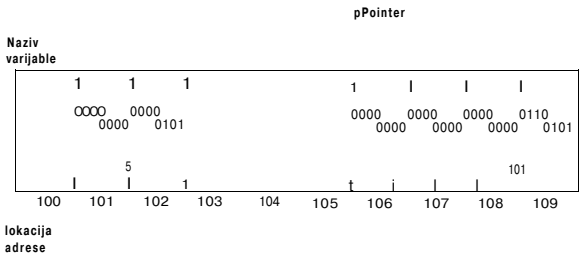
Pokazivaci, adrese i promenljive

Važno je praviti razliku između pokazivača, adrese koju pokazivač čuva i vrednosti na adresi koju čuva pokazivač. Ovo je izvor mnogih konfuzija o pokazivačima.

Razmotrite sledeći fragment koda:

```
int theVariable = 5;
int * pPointer = &theVariable ;
```

theVariable je deklarisana kao celobrojna promenljiva, inicijalizovana sa vrednošću 5. pPointer je deklarisan kao pokazivač na celobrojnu vrednost - inicijalizovan je adresom promenljive theVariable; pPointer je pokazivač Adresa koju pPointer čuva je adresa promenljive theVari abl e. Vrednost na adresi koju pPoi nter čuva je 5. Slika 8.3 prikazuje šematsku reprezentaciju promenljive theVariable i pPointer.



Slika 8.3. Šematska reprezentacija memorije

Manipulisanje podacima, korišćenjem pokazivača

Pošto se pokazivaču dodeli adresa promenljive, Vi možete koristiti taj pokazivač za pristup podatku u toj promenljivoj. Listing 8.2 demonstrira kako se dodeljuje adresa lokalne promenljive pokazivaču i kako pokazivač manipuliše vrednostima u toj promenljivoj.

Listing 8.2: Manipulisanje podacima korišćenjem pokazivača.

```
1   // Listing 8.2 Korišćenje pokazivača
2
3   linclude  <iostream.h>
4
5   typedef unsigned short int USHORT;
6   int main()
7   {
8      USHORT myAge;          //   promenljiva
9      USHORT * pAge = 0;     // pokazivač
10     myAge = 5;
11     cout « "myAge    « myAge « "\n";
12
13     pAge = &myAge;      // dodeli adresu od myAge pokazivaču pAge
14
15     cout « "*pAge    « *pAge « "\n\n";
16
17     cout « "*pAge = 7\n";
18
19     *pAge =7;          // postavlja myAge na 7
20
21     cout « "*pAge:   « *pAge « "\n";
22     cout « "myAge:   « myAge « "\n\n"
23
24
25     cout « "myAge = 9\n";
26
27     myAge = 9;
28
```



```
   cout « "myAge:  " « myAge « "\n"
   cout « "*pAge:  " « *pAge « "\n"

return 0;
}
```

```
myAge:
*pAge:

*pAge = 7
*pAge: 7
myAge: 7

myAge = 9
myAge: 9
*pAge: 9
```

ШЗ1НЈ£> Ovaj program deklariše dve promenljive: unsigned short - myAge i pokazivač na unsigned short - pAge. Promenljivoj myAge se dodeljuje vrednost 5 u liniji 10; ovo se verifikuje izlazom u liniji 11.

U liniji 13 pokazivaču pAge se dodeljuje adresa promenljive myAge. U liniji 15 pAge se dereferencira i štampa, što pokazuje da je vrednost na adresi koju čuva pAage jednaka 5, što se čuva u myAge. U liniji 19 vrednost 7 se dodeljuje promenljivoj na adresi, koja se čuva u pAge. Ovo postavlja myAge na 7, a izlazi u linijama 21-22 potvrduju.

U liniji 27 vrednost 9 se dodeljuje promenljivoj myAge. Ova vrednost se dobija direktno u liniji 29, a indirektno (dereferenciranjem pokazivača pAge) u liniji 30.

Ispitivanje adrese

Pokazivaci Vam omogućavaju da manipulišete adresama, bez poznavanja njihove prave vrednosti. Posle ovog Dana, znaćete da kada dodelite adresu promenljive pokazivaču, on stvarno ima adresu te promenljive kao svoju vrednost. Ali samo ovaj put, zašto ne proveriti da bismo bili sigurni? Listing 8.3 ilustruje ovu ideju.

Listing 8.3: Otkrivanje šta se čuva u pokazivaclma.

```
1:     // Listing 8.3 Šta se čuva u pokazivaču.
2:
3:     linclude  <iostream.h>
4:
5:     typedef unsigned short int USHORT;
6:     int main()
7:     {
8:        unsigned short int myAge = 5, yourAge = 10;
9:        unsigned short int * pAge = &myAge;  // pokazivač
10:
11:       cout « "myAge:\t" « myAge «   "\tyourAge:\t" « yourAge « "\n";
```

nastavlja se

Listing 8.3: Otkrivanje šta se čuvo u pokaiivočimo. *nastavak*

```
    cout « "&myAge:\t" « &myAge «   "\t&yourAge:\t" « iyourAge «"\n";

    cout « "pAge:\t" « pAge « "\n";
    cout « "*pAge:\t" « *pAge « "\n";

    pAge = &yourAge;       // ponovo dodeljuje vrednost pokazivaču

    cout « "myAge:\t" « myAge «   "\tyourAge:\t" « yourAge « "\n";
    cout « "&myAge:\t" « imyAge «   "\t&yourAge:\t" « &yourAge «"\n";

    cout « "pAge:\t" « pAge « "\n";
    cout « "*pAge:\t" « *pAge « "\n";

    cout « "&pAge:\t" « &pAge « "\n";
return 0;
```

```
myAge:      5           yourAge:  10
&myAge:     0x355C      &yourAge: 0x355E
pAge:       0x355C
*pAge:      5
myAge:      5           yourAge:  10
&myAge:     0x355C      &yourAge: 0x355E
pAge:       0x355E
*pAge:      10
&pAge:      0x355A
```

U liniji 8 myAge i yourAge se deklarišu kao promenljive tipa unsigned short int. U liniji 9 pAge se deklariše kao pokazivač na unsigned short int i inicijalizuje se adresom promenljive myAge.

Linije 11 i 12 štampaju vrednosti i adrese promenljivih myAge i yourAge. Linija 14 štampa sadržaj promenljive pAge, koji je adresa promenljive myAge. Linija 15 štampa rezultat dereferenciranja pokazivača pAge, što štampa vrednost na pAge vrednost u myAge, ili 5.

Ovo je suština pokazivača. Linija 14 prikazuje da pAge čuva adresu od myAge, a linija 15 pokazuje kako dobiti vrednost, koja se Čuva u myAge, dereferenciranjem pokazivača pAge. Proverite da li ste ovo potpuno razumeli pre nego što nastavite. Proučite kod i pogledajte izlaz.

U liniji 17 ponovo se obavlja dodela pokazivaču pAge, da bi pokazivao na adresu od yourAge. Ponovo se štampaju vrednosti i adrese. Izlaz pokazuje da pAge sada ima adresu promenljive yourAge i da dereferenciranje dobija vrednost u yourAge.

Linija 25 štampa adresu od pAge. Kao i svaka promenljiva, on ima adresu, koja se može čuvati u pokazivaču (dodeljivanje adrese pokazivača drugom pokazivaču će biti opisano ukratko).

^ PAzm Koristite operator indirekcije (*) za pristup podacima koji se cuvaju na adresi u pokazivaču.

Inicijalizujte sve pokazivače ili na važeću adresu, ili na null (o).

Zapamtite razliku između adrese u pokazivaču i vrednosti na toj adresi.

Pokazivaci

Da biste deklarisali pokazivač, napišite tip promenljive, ili objekta, čija će adresa biti čuvana u pokazivaču, praćen operatorom pokazivača (*) i imenom pokazivača. Na primer,

```
unsigned short int * pPointer = 0;
```

Da biste dodelili vrednost, ili inicijalizovali pokazivač, pre imena promenljive, čija adresa se dodeljuje, navedite operator adresa od (&). Na primer

```
unsigned short int theVariable = 5;
unsigned short int * pPointer = & theVariable;
```

Da bi ste dereferencirali pokazivač, pre imena pokazivača navedite operator dereferenciranja (*). Na primer,

```
unsigned short int theVariable = *pPointer;
```

Zašto biste koristili pokazivače?

Do sada ste videli korak-po-korak detalje dodeljivanja adrese promenljive pokazivaču. U praksi, pak, ovo nikada nećete raditi. Posle svega, zašto dosadivati sa pokazivačem, kada već imate promenljivu sa pristupom do te vrednosti? Jedini razlog za ovu vrstu manipulacije pokazivačima automatske promenljive je da bi se demonstriralo kako pokazivaci rade. Sada, kada ste upoznali sintaksu pokazivača, možete ih staviti u dobru upotrebu. Pokazivaci se, najčešće, koriste za tri zadatka:

- upravljanje podacima na slobodnom skladištu
- pristupanje podacima clanovima i funkcijama članicama klase
- predavanje promenljivih funkcijama po referenci

Ostatak ovog dana fokusira upravljanje podacima na slobodnom skladištu i pristupanje podacima clanovima i funkcijama članicama klase. Sutra ćete učiti o predavanju promenljivih po referenci.

Stek i slobodno skladište

U sekciji "Dodatno objašnjenje", koja sledi posle diskusije funkcija u Danu 5, pomenuto je pet područja memorije:

- prostor globalnih imena

- slobodno skladište
- registxi
- prostor za kod
- stek

Lokalne promenljive su na steku, zajedno sa parametrima funkcije. Kod se nalazi u prostoru za kod, naravno, a globalne promenljive u prostoru globalnih imena. Registri se koriste za interne domaćinske funkcije, kao što je pamćenje vrha steka i pokazivača instrukcija. Skoro sva preostala memorija se dodeljuje slobodnom skladištu, na koje se ponekad upućuje sa gomila.

Problem sa lokalnim promenljivama je taj što one nisu trajne: Po povratku iz funkcije, one se odbacuju. Globalne promenljive rešavaju taj problem po cenu neograničenog pristupa širom programa, što vodi do kreiranja koda koji je težak za razumevanje i održavanje. Stavljanje podataka u slobodno skladište rešava oba ova problema.

0 slobodnom skladištu možete razmišljati ko o masivnoj sekciji memorije, u kojoj hiljade sekvencijalno označenih kockica leže, čekajući Vaše podatke. Ipak, Vi ne možete označiti ove kockice kao što možete stek. Morate tražiti adresu kockice koju rezervišete, a onda tu adresu smestiti u pokazivač.

Jedan od načina razmišljanja je da se poslužite analogijom. Prijatelj Vam da broj 800 za Vrhunsku Poštansku Pošiljku. Vi odete kući i isprogramirate Vaš telefon sa tim brojem, a onda bacite parče papira na kojem je bio zapisan broj. Ako pritisnete dugme, telefon će negde zazvoniti i Vrhunska Poštanska Pošiljka odgovara. Vi se ne sedate broja i ne znate gde je lociran drugi telefon, ali Vam dugme daje pristup Vrhunskoj Poštanskoj Pošiljci, koja je Vaš podatak na slobodnom skladištu. Ne znate gde je, ali znate kako da dodete do nje. Pristupate joj korišćenjem njene adrese - u ovom slučaju, to je broj telefona. Vi ne morate znati taj broj; samo treba da ga stavite u pokazivač (dugme), koji Vam daje pristup podacima, ne opterećujući Vas detaljima.

Stek se automatski čisti po povratku iz funkcije. Sve lokalne promenljive izlaze iz opsega i uklanjaju se sa steka. Slobodno skladište se ne čisti sve dok se vaš program ne završi i Vaš je zadatak da oslobodite svaki memorijski prostor koji ste rezervisali.

Prednost slobodnog skladišta je što memorija koju rezervišete ostaje raspoloživa, dok je eksplicitno ne oslobodite. Ako rezervišete memoriju na slobodnom skladištu u funkciji, memorija je još uvek raspoloživa po povratku iz funkcije.

Prednost pristupanja memoriji na ovaj način, umesto korišćenja globalnih promenljivih, je to da samo funkcije sa pristupom pokazivaču imaju pristup podacima. Ovo obezbeduje usko kontrolisan interfejs ka tim podacima i eliminiše opasnost da neka funkcija promeni podatke na neočekivane i nepripremljene načine.

- Da bi ovo funkcionisalo, morate biti sposobni da kreirate pokazivač na područje u slobodnom skladištu i predajete taj pokazivač izmedu funkcija. Sledeće sekcije
i opisuju kako to uraditi.

new

Memoriju na slobodnom skladištu u C + + alocirate korišćenjem ključne reči new. Posle nje sledi tip objekta koji želite da alocirate tako da kompajler zna koliko se memorije zahteva. New unsigned short int alocira dva bajta na slobodnom skladištu, a new **1** ong alocira četiri.

Povratna vrednost iz new je memorijska adresa. Ona se mora dodeliti pokazivaču. Da biste kreirali unsigned short na slobodnom skladištu, napišite

```
unsigned short int * pPointer;
pPointer = new unsigned short int;
```

Možete, naravno, inicijalizovati pokazivač pri njegovom kreiranju sa

```
unsigned short int * pPointer = new unsigned short int;
```

U svakom slučaju, pPointer sada pokazuje na unsigned short int na slobodnom skladištu. Njega možete koristiti kao i svaki drugi pokazivač na promenljivu i dodeliti vrednost području memorije pisanjem

Ovo znači: "Stavite 72 na vrednost u pPointer", ili: "Dodeli te vrednost 72 području na slobodnom skladištu na koje pokazuje pPointer".

Ako new ne može kreirati memoriju na slobodnom skladištu (memorija je, ipak, ograničen resurs) on vraca null pokazivač. Vi morate proveriti da li je Vaš pokazivač null svaki put kada zahtevate novu memoriju.

|UPOzORfHJi| Svaki put kada alocirate memoriju, korišćenjem ključne reči new, morate proveriti da biste bili sigurni da pokazivač nije null.

delete

Kada završite sa Vašim područjem memorije, morate pozvati delete za pokazivač - on vraća memoriju slobodnom skladištu. Zapamtite da je sam pokazivač, §to je suprotno memoriji na koju on pokazuje, lokalna promenljiva. Po povratku iz funkcije u kom je deklarisan, pokazivač izlazi iz opsega i postaje izgubljen. Memorija alocirana sa new se ne oslobada automatski. Ona postaje neraspoloživa - situacija nazvana memorijska pukotina, jer se ta memorija ne može povratiti, dok se program ne završi. To je kao da je memorija "iscurela" iz Vašeg kompjutera.

Da biste vratili memoriju u slobodno skladište, koristite ključnu reč delete. Na primer

```
delete pPointer;
```

Kada brišete pokazivac, ono što stvarno radite je oslobadanje memorije čija se adresa čuva u pokazivaču. Vi govorite: "Vrati slobodnom skladištu memoriju, na koju ovaj pokazivač pokazuje." Pokazivač je još uvek pokazivač i njemu se ponovo može dodeliti vrednost. Listing 8.4 demonstrira alociranje promenljive na gomili, korišćenje te promenljive i njeno brisanje.

^UPOZORENJp **Kada pozivate delete za pokazivač, oslobada se memorija na koju on pokazuje. Pozivanje delete ponovo za taj pokazivac će oboriti Vas program! Kada obrisete pokazivac, postavite ga na nula (null). Pozivanje delete za nul l pokazivac je garantovano sigurno. Na primer:**

```
Animal *pDog = new Animal;
delete pDog; //oslobada memoriju
   pDog = 0; //postavlja pokazivac na null
   //•••
delete pDog; //bezopasno
```

Listing 8.4: Alociranje, korišćenje, i brisanje pokazivača.

```
// Listing 8.4
// Alociranje i brisanje pokazivača

#include <iostream.h>
int main()
{
   int localVariable = 5;
   int * pLocal= &localVariable;
   int * pHeap = new int;
   if (pHeap == NULL)
    {
        cout « "Greska! Nema memorije za pHeap!
        return 0;

   *pHeap = 7;
   cout « "localVariable: " « localVariable « "\n"
   cout « "*pLocal: " « *pLocal « "\n";
   cout « "*pHeap: " « *pHeap « "\n";
   delete pHeap;
   pHeap = new int;
   if (pHeap == NULL)
   {
        cout « "Greska! Nema memorije za pHeap!";
        return 0;
   }
   *pHeap = 9;
   cout « "*pHeap: " « *pHeap « "\n";
   delete pHeap;
   return 0;


   localVariable: 5
   *pLocal: 5
   *pHeap: 7
   *pHeap: 9
```


Linija 7 deklariše i inicijalizuje lokalnu promenljivu. Linija 8 deklariše i inicijalizuje pokazivac adresom lokalne promenljive. Linija 9 deklariše drugi pokazivac, ali ga inicijalizuje rezultatom, koji je dobijen pozivanjem new int. Ovo alocira prostor na slobodnom skladištu za i nt. Linija 10 verifikuje da je memorija alocirana i da je pokazivac važeći (ne nul l). Ako memorija ne može biti alocirana, pokazivac je nul l i štampa se poruka o grešci.

Da biste se držali jednostavnosti, ova provera greške često neće biti reprodukovana u budućim programima, ali Vi morate uključiti neku vrstu provere greške u svojim programima.

Linija 15 dodeljuje vrednost 7 novoalociranoj memoriji. Linija 16 štampa vrednost lokalne promenljive, a linija 17 štampa vrednost na koju pokazuje pLocal. Kao što se očekuje, one su jednake. Linija 18 štampa vrednost na koju pokazuje pHeap. Ona pokazuje da je vrednost koja je dodeljena u liniji 15, u stvari, pristupačna.

U liniji 19 memorija alocirana u liniji 9 se vraća slobodnom skladištu, pozivom delete. Ovo oslobada memoriju rastavlja pokazivac od memorije. Sada je pHeap Slobodan da pokazuje na drugu memoriju. Njemu se ponovo dodeljuje vrednost u linijama 20 i 26, a linija 27 štampa rezultat. Linija 28 vraća tu memoriju slobodnom skladištu.

Iako je linija 28 suvišna (kraj programa bi vratio tu memoriju), dobra je ideja osloboditi ovu memoriju eksplicitno. Ako se program promeni, ili proširi, biće korisno što ste se već pobrinuli o ovom koraku.

Memorijske pukotine

Drugi način na koji biste, možda, kreirali memorijsku pukotinu je ponovno dodeljivanje vrednosti Vašem pokazivaču, pre brisanja memorije, na koju on pokazuje. Razmotrite ovaj fragment koda:

```
1:   unsigned short int * pPointer = new unsigned short int;
2:   *pPointer = 72;
3:   pPointer = new unsigned short int;
4:   *pPointer = 84;
```

Linija 1 kreira pPointer i dodeljuje mu adresu područja na slobodnom skladištu. Linija 2 stavlja vrednost 72 u to područje memorije. Linija 3 dodeljuje pokazivac pPointer drugom području memorije. Linija 4 stavlja vrednost 84 u to područje. Originalno područje, u kojem se sada čuva vrednost 72, neraspoloživo je, jer je obavljena ponovna dodela pokazivaču na to područje memorije. Nema načina da se pristupi torn originalnom području memorije, niti ima bilo kakvog načina da se ono slobodi pre završetka programa.

Kod bi trebalo da bude napisan ovako:

```
1: unsigned short int * pPointer = new unsigned short int;
2: *pPointer = 72;
3: delete pPointer;
4: pPointer = new unsigned short int;
5: *pPointer = 84;
```

Sada se briše memorija, na koju se originalno pokazuje sa pPointer, i time, oslobada u liniji **3.**

i m ***^ФАШЛШЦ*** **Uvek kada pozovete new, trebalo bi da postoji poziv i za delete. Yazno je da pamtite koji pokazivac poseduje podrucje memorije i da osigurate da se memorija vraća slobodnom skladistu kada ste zavrsili sa njom.**

Kreiranje objekata na slobodnom skladištu

Kao što možete kreirati pokazivac na celobrojnu vrednost, možete kreirati pokazivac na bilo koji objekat. Ako ste deklarisali objekat tipa Cat, možete deklarisati pokazivac na tu klasu i instancirati objekat Cat na slobodnom skladištu, kao što to možete uraditi na steku. Sintaksa je ista kao i za celobrojne vrednosti:

Ovo poziva podrazumevani konstruktor - konstruktor koji ne prihvata parametre. Konstruktor se poziva uvek kada se kreira objekat (na steku, ili na slobodnom skladištu).

Brisanje objekata

Kada pozovete delete za pokazivac na objekat na slobodnom skladištu, poziva se destruktor objekta pre oslobadanja memorije. Ovo daje Vašoj klasi šansu da obavi čišćenje, kao što to ona radi za objekte koji se uništavaju na steku. Listing 8.5 ilustruje kreiranje i brisanje objekata na slobodnom skladištu.

Listing 8.5: Kreiranje i brisanje objekata na slobodnom skladištu.

```
1    // Listing 8.5
2    // Kreiranje objekata na slobodnom skladištu
3
4     linclude <iostream.h>
5
6     class SimpleCat
7     {
8     public:
9           SimpleCat();
10:           -SimpleCatO;
11    private:
12          int itsAge;
13       };
14
15     SimpleCat::SimpleCat()
16    {
17          cout « "Pozvan konstruktor.\n";
18          itsAge = 1;
19    }
20
21    SimpleCat: :-SimpleCat()
22    {
23          cout « "Pozvan dekonstruktor.\n";
24    }
25
26    int mainQ
27    {
28          cout « "SimpleCat Frisky...\n";
29          SimpleCat Frisky;
30          cout « "SimpleCat *pRags = new SimpleCat...\n";
31          SimpleCat * pRags = new SimpleCat;
32          cout « "brise pRags...\n";
33          delete pRags;
34          cout « "Izlaz, evo ide Frisky ...\n";
35    return 0;
36    }
```

```
SimpleCat Frisky...
Pozvan konstruktor.
SimpleCat *pRags = new SimpleCat..
Pozvan konstruktor.
brise pRags...
Pozvan dekonstruktor.
Izlaz, evo ide Frisky ...
Pozvan dekonstruktor.
```

Linije 6-13 deklarišu svučenu klasu SimpleCat. Linija 9 deklariše konstruktor klase SimpleCat, a linije 15-19 sadrže njenu definiciju. Linija 10 deklariše destruktor klase SimpleCat, a linije 21-24 sadrže njenu definiciju.

U liniji 29 Fri sky se kreira na steku, što prouzrokuje zvanje konstruktora. U liniji 31 SimpleCat, na koji pokazuje pRags, kreira se na gomili; konstruktor se ponovo poziva. U liniji 33 delete se poziva za pRags i poziva se destruktor. Kada se funkcija završi, Frisky izlazi iz opsega, i poziva se destruktor.

Pristupanje podacima clanovima

Podacima clanovima i funkcijama ste pristupali, korišćenjem operatora tačka (.) za Cat objekte koji su lokalno kreirani. Da biste pristupili Cat objektu na slobodnom skladištu, morate dereferencirati pokazivac i pozvati operator tačka za objekat na koji pokazuje pokazivac. Da biste pristupili funkciji članici GetAge, pišete.

Zagrade se koriste da bi se osiguralo da se pRags dereferencira, pre nego što se pristupi GetAgeO.

Zato što je ovo nezgodno, C++ obezbeduje stenografski operator za indirektan pristup: operator pokazuje na (->), koji se kreira ukucavanjem crtice (-), posle koje odmah sledi simbol veće od (>). C++ ovo tretira kao jedan simbol. Listing 8.6 demonstrira pristupanje promenljivim članicama i funkcijama objekata kreiranih na slobodnom skladištu.

Listing 8.6: Pristup podacima clanovima objekata koji se nalaze na slobodnom skladištu.

```
// Listing 8.6
// Pristup podacima clanovima objekata koji se nalaze na gomili

#include <iostream.h>

class SimpleCat
{
public:
      SimpleCat() {itsAge = 2; }
       -SimpleCatQ  {}
       int GetAgeO const { return itsAge; }
       void SetAge(int age) { itsAge = age; }
private:
       int itsAge;


  int main()
  {
       SimpleCat * Frisky = new SimpleCat;
       cout « "Frisky ima " « Frisky->GetAge() « " godina\n"
       Frisky->SetAge(5);
       cout « "Frisky ima " « Frisky->GetAge() « " godina\n"
       delete Frisky;
  return 0;
  }
```

```
Frisky ima 2 godina
Frisky ima 5 godina
```

U liniji 19 `SimpleCat` objekat se instancira na slobodnom skladištu. Podrazumevani konstruktor postavlja svoju starost na 2 i poziva se metod `GetAgeO` u liniji 20. Zato što je ovo pokazivac, operator indirekcije (->) se koristi za pristup podacima clanovima i funkcijama članicama. U liniji 21 poziva se metoda `SetAge()`, a funkciji `GetAgeO` se ponovo pristupa u liniji 22.

Podaci clanovi na slobodnom skladištu

Jedan, ili vise podataka članova klase mogu biti pokazivaci na objekat na slobodnom skladištu. Memorija se može alocirati u konstruktoru klase, ili u jednom od njegovih metoda, a može se izbrisati u njenom destruktoru, kao što ilustruje listing 8.7.

§ **Listing 8.7: Pokazivaci kao podaci clanovi.**

```
// Listing 8.7

// Pokazivaci kao podaci clanovi

  #include <iostream.h>

  class SimpleCat
  {
  public:
         SimpleCat();
          -SimpleCat();
          int GetAgeO const { return *itsAge; }
          void SetAge(int age) { *itsAge = age; }
          int GetWeightQ const { return *itsWeight; }
          void setWeight (int weight) { *itsWeight = weight; }

   private:
          int * itsAge;
          int * itsWeight;


     SimpleCat::SimpleCat()
     {
          itsAge = new int(2);
          itsWeight = new int(5);


     SimpleCat::-SimpleCat()
     {
          delete itsAge;
          delete itsWeight;


     int main()
     {
           SimpleCat *Frisky = new SimpleCat;
           cout « "Frisky ima " « Frisky->GetAge() « " godina\n"
           Frisky->SetAge(5);
           cout « "Frisky is " « Frisky->GetAge() « " godina\n";
           delete Frisky;
     return 0;
```

```
Frisky ima 2 godina
Frisky ima 5 godina
```

IECTEfcfr K*[asa] SimpleCat je deklarisana sa dve promenljive članice - obadve su pokazivaci na celobrojne vrednosti, u linijama 18 i 19. Konstruktor (linije 22-26) inicijalizuje pokazivače na memoriju sa slobodnog skladišta i na podrazumevane vrednosti.

Destruktor (linije 28-32) čisti alociranu memoriju. Zato što je ovo destruktor, nema svrhe dodeljivati null ovim pokazivačima, jer oni vise neće biti pristupačni. Ovo je jedno od sigurnih mesta gde se može prekršiti pravilo da izbrisanim pokazivačima treba dodeliti null, iako pridržavanje pravila neće smetati.

Pozivajuća funkcija (u ovom slučaju, main()) nesvesna je da su itsAge i itsWeight pokazivaci na memoriju sa slobodnog skladišta. Potom main() nastavlja sa pozivanjem GetAgeO i SetAge(), a detalji upravljanja memorijom su sakriveni u implementaciji klase, kao što i treba.

Kada se Frisky izbriše u liniji 40, poziva se njegov destruktor, koji briše svaki njegov pokazivac clan. Ako oni, dalje, pokazuju na objekte drugih korisnički definisanih klasa, njihovi destruktori se takode pozivaju.

Pokazivac this

Svaka funkcija članica klase ima skriveni parametar: pokazivac this, koji pokazuje na individualni objekat. U svakom pozivu funkcija GetAgeO, ili Set Age (), pokazivac this za objekat je uključen kao skriven i parametar.

Moguće je koristiti pokazivac this eksplicitno, kao što ilustruje listing 8.8.

Listing 8.8: Korišćenje pokazivača this.

```
1:    // Listing 8.8
2:    // Koriščenje pokazivača this
3:
4:    linclude <iostream.h>
5:
6:    class Rectangle
7:    {
8:    public:
9:         Rectangle();
10:         -RectangleQ;
11:         void SetLength(int length) { this->itsLength = length; }
12:         int GetLength() const { return this->itsLength; }
13:
14:         void SetWidth(int width) { itsWidth = width; }
15:         int GetWidth() const { return itsWidth; }
16:
17:    private:
```



```
18:         int itsLength;
19:         int itsWidth;
20:
21:
22:   Rectangle::Rectangle()
23:
24:       itsWidth = 5;
25:       itsLength = 10;
26:
27:   Rectangle::~Rectangle()
28:    }
29:
30:    nt main()
31:
32:         Rectangle theRect;
33:         cout « "theRect je " < theRect.GetLength()    " stopa dug.\n";
34:         cout « "theRect je " < theRect.GetWidth() <    stopa sirokAn";
35:         theRect.SetLength(20);
36:         theRect.SetWidth(IO);
37:         cout « "theRect je " < theRect.GetLength()« "stopa dug.\n";
38:         cout « "theRect je M < theRect.GetWidth()« "stopa sirokAn"
39:   return 0;
40:   }
```

```
theRect je 10 stpa dug.
theRect je 5 stopa sirok.
theRect je 20 stopa dug.
theRect je 10 stopa sirok.
```

Funkcije pristupa SetLenghtO i GetLenghtO eksplicitno koriste pokazivac this za pristup promenljivama članicama Rectangle objekta. A SetWidth i GetWidth to ne rade. Nema razlike u njihovom ponašanju, iako je sintaksa lakša za razumevanje.

Kada bi to bilo sve o pokazivaču this, bilo bi malo svrhe dosadivatj Vam s njim. Ipak, pokazivac this je pokazivac; on čuva memorijsku adresu objekta, pa, zato, može biti moćna alatka.

Videćete praktičnu upotrebu za pokazivac this u Danu 10, "Napredne funkcije", kada se bude govorilo o operatoru preklapanja. Za sada, Vaš cilj je da znate za pokazivac this i da razumete šta je on: pokazivac na sam objekat.

Ne treba da brinete o kreiranju i brisanju pokazivac this. O tome brine kompajler.

Izgubljeni, ili klimavi pokazivaci

Jedan izvor bagova koji su nezgodni i teški za pronalaženje su izgubljeni pokazivaci. Izgubljeni pokazivaci se kreiraju kada pozovete delete za pokazivac, čime oslobadate memoriju na koju on pokazuje, a kasnije pokušate da ponovo upotrebite taj pokazivac bez ponovnog dodeljivanja.

To je kao da se kompanija Vrhinska Poštanska Pošiljka odselila, a vi još uvek pritiskate programirano dugme na Vašem telefonu. Moguće je da se ne dogodi ništa strašno - telefon zvoni u napušenom skladištu. Možda je broj telefona dodeljen fabrici oružja, a Vaš poziv detonira eksploziv i digne u vazduh ceo grad!

Ukratko, pazite da ne koristite pokazivac pošto ste pozvali delete za njega. Pokazivac još uvek pokazuje na staro podrčje memorije, ali kompajler ima slobodu da tamo stavi druge podatke; korišćenje pokazivača može prouzrokovati pad Vašeg programa. Još gore, Vaš program bi mogao nemarno nastaviti i pasti nekoliko minuta kasnije. Ovo se zove tempirana bomba, a to nije zabavno. Da bi ste bili sigurni, pošto brišete pokazivac, postavite ga na null (0). Ovo razoružava pokazivac.

yHAPOMIMAy. Izgubljeni pokazivaci se često nazivaju divlji pokazivaci ili klimavi pokazivaci.

Listing 8.9 ilustruje kreiranje izgubljenog pokazivača.

liiPOZOREMJIl Ovaj program namerno kreira izgubljeni pokazivac NEMOJTE pokrenuti ovaj program - on će pasti, ako ste srećni.

Listing 8.9: Kreiranje zalutaiog pokazivača.

```
1:    // Listing 8.9
2:    // Demonstrira zalutali pokazivac
3:    typedef unsigned short int USHORT;
4:    linclude <iostream.h>
5:
6:    int main()
7:    {
8:       USHORT * pint = new USHORT;
9:       *pInt = 10;
10:      cout « "*pInt: " « *pInt « endl;
11:      delete pint;
12:      pint = 0;
13:      long * pLong = new long;
14:      *pLong = 90000;
15:      cout « "*pLong: " « *pLong « endl;
16:
17:      *pInt = 20;      // uh oh, ovaj je izbrisan!
18:
19:      cout « "*pInt: " « *pInt  « endl;
20:      cout « "*pLong: " « *pLong  « endl;
21:      delete pLong;
22:   return 0;
23:   }
```

```
J Ш a Ш ^  *pInt:    10
           *pLong:   90000
           *pInt:    20
           *pLong:   65556
           Null pointer assignment
```


'IIIIIIjjJ^ Linija 8 deklariše pint kao pokazivac na USHORT, a pint pokazuje na novu alociranu memoriju. Linija 9 stavlja vrednost 10 u tu memoriju, a linija 10 štampa njenu vrednost. Posle štampanja vrednosti, delete se poziva za pokazivac. Sada je pint izgubljen, ili klimavi pokazivac.

Linija 13 deklariše novi pokazivac, pLong, koji pokazuje na memoriju alociranu sa new. Linija 14 dodeljuje vrednost 9.000 pokazivaču pLong, a linija 15 štampa njegovu vrednost.

Linija 17 dodeljuje vrednost 20 memoriji na koju pokazuje pint ali on ne pokazuje na važeći prostor. Memorija na koju pint pokazuje je oslobodena pozivom del ete, pa je dodeljivanje vrednosti toj memoriji sigurno dovodi do katastrofe.

Linija 19 štampa vrednost na pint. Stvarno, to je 20. Linija 20 štampa 20, vrednost na pLong; ona je iznenada promenjena na 65.556. Javljaju se dva pitanja:

1. Kako se vrednost pokazivača pLong mogla promeniti, imajući u vidu to da pLong nije diran?

2. Gde je 20 otišlo kada je upotrebljen pint u liniji 17?

Kao što, možda, pogadate, ova pitanja su povezana. Kada je vrednost stavljena na pint u liniji 17, kompajler je srećno stavio vrednost 20 na memorijsku lokaciju, na koju je pint prethodno pokazivao. Ipak, zato što je ta memorija oslobođena u liniji 11, kompajler je bio Slobodan daje ponovo dodeli. Kada je pLong bio kreiran u liniji 13, njemu je bila data stara memorijska lokacija pokazivača pint (na nekim kompjuterima ovo se, možda, neće dogoditi, što zavisi od toga gde se u memoriji čuvaju ove promenljive). Kada je vrednost 20 bila dodeljena lokaciji, na koju je pint prethodno pokazivao, to je prepisalo vrednost na koju je pokazivao pLong. Ovo se naziva "gaženje pokazivača". To je često nesrećni rezultat korišćenja izgubljenog pokazivača.

Ovo je posebno nezgodan bag, jer vrednost koja je promenjena nije bila povezana sa izgubljenim pokazivačem. Promena vrednosti na pLong je bio sporedan efekat zloupotrebe pokazivača pint. U velikom programu ovo bi bilo veoma teško za pronalaženje.

Samo iz zabave, evo detalja kako je broj 65.556 dospeo na memorijsku adresu:

1. pint je pokazivao na odredenu memorijsku lokaciju i dodeljena je vrednost 10.

2. delete je pozvan za pint, što je reklo kompajleru da može da stavi nešto drugo na tu lokaciju. Pokazivaču pLong je dodeljena ista memorijska lokacija.

3. Vrednost 9.000 je dodeljena *pLong. Kompjuter, korišćen u ovom primeru, smestioje četvorobajtnu vrednost 90.000 (00 01 5F 90) u zamenjenom redosledu bajtova. Tako, smešteno je 5F 90 00 01 .

4. Pokazivaču pint je dodeljena vrednost 20, ili 00 14 u heksadecimalnoj notaciji. Zato što je pint još uvek pokazivao na istu adresu, prva dva bajta pokazivača pLong su bila prepisana, što je ostavilo 00 14 00 01.
5. Vrednost na pLong je odštampana, vraćajući bajtove u njihov ispravan redosled 00 01 00 14, koja je prevedena u DOS vrednost 65.556.

<^f PAzm Upotrebite new za kreiranje objekata na slobodnom skladištu.

Upotrebite delete za uništavanje objekata na slobodnom skladištu i za vraćanje njihove memorije.

Nemojte zaboraviti da izbalansirate sve iskaze new iskazima del ete.

Nemojte zaboraviti da dodelite nul 1 (0) svim pokazivačima za koje pozivate delete.

Ispitajte vrednost vraćenu sa new.

const pokazivaci

Možete koristiti ključnu reč const za pokazivače pre i posle tipa, ili na oba mesta. Na primer, sve što sledi su legalne deklaracije:

```
const int * pOne;
int * const pTwo;
const int * const pThree;
```

pOne je pokazivac na konstantan celobrojan broj. Vrednost na koju se pokazuje se ne može promeniti.

pTwo je konstantan pokazivac na celobrojnu vrednost. Celobrojna vrednost se može promeniti, ali pTwo ne može pokazivati na nešto drugo.

pThree je konstantan pokazivac na konstantnu celobrojnu vrednost. Vrednost na koju se pokazuje se ne može promeniti, a pThree se ne može promeniti da pokazuje na nešto drugo.

Trik za razumevanje ovoga je da pogledate na desnu stranu ključne reči const, da biste otkrili šta se deklariše kao konstantno. Ako je tip sa desne strane ključne reči, tada je vrednost konstantna. Ako je promenljiva sa desne strane ključne reči const, tada je sama pokazivačka promenljiva konstantna.

```
const int * pi;   // int na koji se pokazuje je konstanta
int * const p2;   // p2 konstanta, on ne može pokazivati ni na šta drugo
```


const pokazivaci i const funkcije članice

U Danu 6, "Osnovne klase", naučili ste da možete primeniti ključnu reč const na funkciju članicu. Kada se funkcija deklariše sa const, kompajler označava kao grešku svaki pokušaj da se promene podaci u objektu unutar te funkcije.

Ako deklarišete pokazivac na const objekat, jedini metodi koje možete pozvati tim pokazivačem su const metodi. Listing 8.10 ilustruje ovo.

Listing 8.10: Korišćenje pokazivača na const objekte.

```
// Listing 8.10

// Korišćenje pokazivača sa const metodama

linclude <iostream.h>

class Rectangle
{
public:
     Rectangle();
      -RectangleQ;
      void SetLength(int length) { itsLength - length; }
      int GetLength() const { return itsLength; }

      void SetWidth(int width) { itsWidth = width; }
      int GetWidth() const { return itsWidth; }

 private:
      int itsLength;
      int itsWidth;
 };

 Rectangle::Rectangle():
 itsWidth(5),
 itsLength(IO)
 {}

 Rectangle::-Rectangle!)
 {}

 int mainQ
 {
      Rectangle* pRect =  new Rectangle;
      const Rectangle * pConstRect = new Rectangle;
      Rectangle * const pConstPtr = new Rectangle;

      cout « "pRect width: " « pRect->GetWidth() « " feet\n";
      cout « "pConstRect width: " « pConstRect->GetWidth() « " feet\n"
      cout « "pConstPtr width: " « pConstPtr->GetWidth() « " feet\n";
```

nastavlja se

Listing 8.10: Korišćenje pokazivača na const objekte. *nastavak*

```
        pRect->SetWidth(10);
        // pConstRect->SetWidth(10);
        pConstPtr->SetWidth(10);

        cout « "pRect width: " « pRect->GetWidth() « " feet\n";
        cout « "pConstRect width: " « pConstRect->GetWidth() « " feet\n"
        cout « "pConstPtr width: " « pConstPtr->GetWidth() « " feet\n";
return 0;
```

```
pRect width: 5 feet
pConstRect width: 5 feet
pConstPtr width: 5 feet
pRect width: 10 feet
pConstRect width: 5 feet
pConstPtr width: 10 feet
```

ANALIZA linije 6-20 deklarišu `Rectangle`. Linija 15 deklariše metodu članicu `GetWidth()` kao `const`. Linija 32 deklariše pokazivac na `Rectangle`. Linija 33 deklariše `pConstRect`, koji je pokazivac na konstantan `Rectangle`. Linija 34 deklariše `pConstPtr`, koji je konstantan pokazivac na `Rectangle`.

Linije 36-38 štampaju njihove vrednosti.

U liniji 40 `pRect` se koristi za postavljanje širine pravougaonika na 10. U liniji 41 `pConstRect` bi bio korišćen, ali on je deklarisan da pokazuje na konstantan `Rectangle`. Zato, on ne može legalno pozvati ne-const funkciju članicu; isključen'je komentarom. U liniji 42 `pConstPtr` poziva `SetWidth()`, `pConstPtr` je deklarisan kao konstantan pokazivac na pravougaonik. Drugim rečima, pokazivac je konstantan i ne može pokazivati na nešto drugo, ali pravougaonik nije konstantan.

const this pokazivaci

Kada deklarišete objekat sa `const`, Vi, u stvari, deklarišete da je pokazivac `this` pokazivac na `const` objekat. Pokazivac `const this` se može koristiti samo sa `const` funkcijama članicama.

Konstantni objekti i konstantni pokazivaci će biti objašnjeni ponovo sutra, kada se bude diskutovalo o referencama na konstantne objekte.

PAZITI Zaštitite objekte predate po referenci sa `const`, ako ne treba da budu promenjeni.

Predajte po referenci, kada objekat može biti promenjen.

Predajte po vrednosti, kada mali objekti ne treba da budu promenjeni.

Pokazivaci obezbeduju moćan način za pristup podacima, preko indirekcije. Svaka promenljiva ima adresu, koja se može dobiti korišćenjem operatora adresa od (&). Adresa se može čuvati u pokazivaču.

Pokazivaci se deklarišu navođenjem tipa objekta na koji oni pokazuju, posle koga slede operator indirekcije (*) i ime pokazivača. Pokazivaci bi trebalo da budu inicijalizovani da pokazuju na objekat, ili na nul 1 (0).

Vi pristupate vrednosti na adresi, Čuvanoj u pokazivaču, korišćenjem operatora indirekcije (*). Možete deklarisati const pokazivače, kojima se ne može ponovo dodeliti vrednost da bi pokazivali na druge objekte, i pokazivače na const objekte, koji se ne mogu koristiti za menjanje objekata na koje pokazuju.

Da biste kreirali nove objekte na slobodnom skladištu, koristite ključnu reč new i pokazivaču dodeljujete adresu koja se vraća. Tu memoriju oslobadate pozivanjem ključne reči delete za pokazivac. Ona oslobada memoriju, ali ne uništava pokazivac. Zato, morate obaviti ponovno dodeljivanje pokazivaču, posle oslobadanja njegove memorije.

Pitanja i odgovori

P Zašto su pokazivaci tako važni?

O Danas ste videli kako se pokazivaci koriste za cuvanje adrese objekata na slobodnom skladištu i kako se oni koriste za predavanje argumenata po referenci. Pored toga, u Danu 13, "Polimorfizam", videćete kako se pokazivaci koriste u polimorfizmu klasa.

P Zašto bih dosadivao sa deklarisanjem nečega na slobodnom skladištu?

O Objekti na slobodnom skladištu postoje posle povratka iz funkcije. Pored toga, sposobnost da se čuvaju objekti na slobodnom skladištu omogućava Vam da odlučite, u vreme izvršenja, koliko objekata Vam je potrebno, umesto da ih morate deklarisati u napred. Ovo će sutra biti detaljnije ispitano.

P Zašto bi trebalo da deklarišem objekat sa const, ako to ograničava ono što ja mogu uraditi sa njim?

O Kao programer, Vi tražite od kompajlera pomoć u pronalaženju bagova. Jedan ozbiljan bag, čije je pronalaženje teško, je funkcija koja menja objekat na načine koji nisu očigledni za pozivajuću funkciju. Deklarisanje objekta sa const sprečava takve promene.

Radionica

Nudimo Vara test, koji treba da Vam pomogne da utvrdite svoje razumevanje predenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

1. Koji operator se koristi za odredivanje adrese promenljive?
2. Koji operator se koristi za pronalaženje vrednosti na adresi u pokazivaču?
3. Šta je pokazivac?
4. Kakva je razlika izmedu adrese koja se čuva u pokazivaču i vrednosti na toj adresi?
5. Kakva je razlika između operatora indirekcije i operatora adresa od?
6. Kakva je razlika između const int * ptrOne i int * const ptrTwo?

Vežbe

1. Šta ove deklaracije rade?

   a. `int * pOne;`

   b. `int vTwo;`

   c. `int * pThree = &vTwo;`

2. Ako imate unsigned short promenljivu, nazvanu yourAge, kako biste deklarisali pokazivac za manipulisanje promenljivom yourAge?
3. Dodelite vrednost 50 promenljivoj yourAge, korišćenjem pokazivača, koji je deklarisan u Vežbi 2.
4. Napišite mali program koji deklariše celobrojnu vrednost i pokazivac na celobrojnu vrednost. Dodelite pokazivaču adresu celobrojne vrednosti. Upotrebite pokazivac za postavljanje vrednosti u celobrojnoj promenljivoj.

5. ISTERIVAČI BAGOVA. Šta nije u redu sa ovim kodom?

```
#include <iostream.h>
int main()
{
     int *pInt;
     *pInt = 9;
     cout « "The value at pint: " « *pInt;
     return 0;
}
```

6. ISTERIVAČI BAGOVA: Šta nije u redu sa ovim kodom?

```
int main()

    int SomeVariable = 5;
    cout « "SomeVariable: " « SomeVariable « "\n";
    int *pVar = & SomeVariable;
    pVar = 9;
    cout « "SomeVariable: " « *pVar « "\n";
return 0;
```

Dan 9

Reference

Juče ste naučili kako da koristite pokazivače za manipulisanje objektima na slobodnom skladištu i kako da indirektno uputite na te objekte. Reference, tema današnje glave, pokazaće Vam skoro svu snagu pokazivača, ali sa mnogo lakšom sintaksom. Danas ucite sledeće

- šta su reference
- po čemu se reference razlikuju od pokazivača
- kako kreirati i koristiti reference
- koja su ograničenja referenci
- kako predati vrednosti i objekte u funkcijei iz funkcija po referenci.

v

Sta je referenca?

Referenca je alijas; kada kreirate referencu, Vi je inicijalizujete imenom drugog objekta, metom. Od tog momenta, referenca se ponaša kao alternativno ime za metu i sve što radite referenci se stvarno radi meti.

Referencu kreirate navodenjem tipa ciljnog objekta, praćenog referencnim operatorom (&), posle kojeg sledi ime reference. Reference mogu koristiti svako legalno ime promenljive, ali za ovu knjigu mi ćemo dodati prefiks imenima svih referenci sa V\ Ako imate, na primer, celobrojnu promenljivu nazvanu somelnt, možete napraviti referencu na tu promenljivu pisanjem na sledeći način:

```
int &rSomeRef = someInt;
```

Ovo se čita kao: "rSomeRef je referenca na celobrojnu vrednost koja je inicijalizovana da upućuje na someInt." Listing 9.1 prikazuje kako se reference kreiraju i koriste.

Uočite da je referencni operator (&) isti simbol kao onaj korišćert za adresu operatora od. Ipak, ovo nisu isti operatori, iako su blisko povezani.

Listing 9.1. Kreiranje i korišćenje referenci.

```
//Listing 9.1
// Demonstriranje upotrebe referenci

#include <iostream.h>

int main()
{
     int    intOne;
     int &rSomeRef = intOne;

     intOne = 5;
     cout « "intOne:  " « intOne « endl;
     cout « "rSomeRef:  " « rSomeRef « endl;

     rSomeRef = 7;
     cout « "intOne:  " « intOne « endl;
     cout « "rSomeRef: " « rSomeRef « endl;
return 0;
```

```
intOne: 5
rSomeRef: 5
intOne: 7
rSomeRef: 7
```

U liniji 8 deklarisana je lokalna promenljiva `int, intOne`. U liniji 9 deklarisana je referenca na `int, rSomeRef`, i inicijalizovana da upućuje na `intOne`. Ako deklarišete referencu, ali je ne inicijalizujete, dobićete grešku u vreme kompajliranja. Reference moraju biti inicijalizovane.

U liniji 11 promenljivoj `intOne` se dodeljuje vrednost 5. U linijama 12 i 13 štampaju se vrednosti u `intOne` i `rSomeRef` i one su, naravno, iste.

U liniji 15 referenci `rSomeRef` se dodeljuje 7. Pošto je ovo referenca, ona je alijas za `intOne,` i tako se 7, zapravo, dodeljuje promenljivoj `intOne,` kao što je prikazano izlazima u linijama 16 i 17.

Korišćenje operatora adresa od (&) sa referencama

Ako od reference tražite adresu, ona vraća adresu svoje mete. To je priroda referenci. One su alijasi za metu. Listing 9.2 ovo demonstrira.

Listing 9.2. Uzimanje adrese reference.

```
1    //Listing 9.2
2
3    // Demonstriranje upotrebe referenci
4
5    linclude <iostream.h>
6
7    int main()
8    {
9        int    intOne;
         int irSomeRef = intOne;
10
11       intOne = 5;
12       cout « "intOne:  " « intOne « endl;
13       cout « "rSomeRef:  " « rSomeRef « endl;
14
15       cout « "&int0ne:  "   « &int0ne « endl;
16       cout « "&rSomeRef:  " « &rSomeRef « endl;
17
18   return 0;
19   }
```

```
intOne: 5
rSomeRef: 5
&int0ne:  0x3500
&rSomeRef: 0x3500
```

У^**МАРОМЕНА**^ Vaš izlaz se može razlikovati u zadnje dve linije.

ОЗПЈЛ^ Da ponovimo: `rSomeRef` se inicijalizuje kao referenca na `intOne`. Ovog puta se štampaju adrese dve promenljive i one su identične. C++ Vam ne daje način da pristupite adresi same reference, jer to nije značajno, kao što bi bilo da ste koristili pokazivac, ili drugu promenljivu. Reference se inicijalizuju kada se kreiraju i uvek se ponašaju kao sinonim za njihovu metu, čak i kada se primeni operator adresa od.

Na primer, ako imate klasu nazvanu `President,` možda biste deklarisali primerak te klase na sledeći način:

```
President  Wi11iam_Jefferson_Clinton;
```

Možda biste onda deklarisali referencu na `President` i inicijalizovali je ovim objektom:

```
President &Bi11_C1inton = Wi11iam_Jefferson_Clinton;
```

Postoji samo jedan `President;` oba identifikatora upućuju na isti objekat iste klase. Svaka akcija koju preduzmete nad `Bill_Cl inton` biće, takođe, izvedena nad `William Jefferson Clinton.`

Budite pažljivi i pravite razliku između simbola & u liniji 9 iz listinga 9.2, koja deklariše referencu na int, nazvanu rSomeRef, i simbola & u linijama 15 i 16, koji vraćaju adrese celobrojne promenljive intOne i reference rSomeRef.

Normalno, kada koristite referencu, ne koristite operator adresa od. Jednostavno, koristite referencu, kao što biste koristili ciljnu promenljivu. Ovo je prikazano u liniji 13.

Čak i iskusni C+4- programeri, koji znaju pravilo da se nad referencama ne može ostvariti ponovna dodela i da su one uvek alijasi za njihovu metu, mogu biti zbunjeni time šta se dešava kada pokušate da obavite ponovno dodeljivanje referenci. Ono što se čini da je ponovna dodela ispada da je dodela nove vrednosti meti. Listing 9.3 ilustruje ovu činjenicu.

Listing 9.3. Dodeljivanje referenci.

```
1  //Listing 9.3
2   //Ponovno dodeljivanje referenci
3
4   linclude <iostream.h>
5
6   int main()
7   I
8        int   intOne;
9        int &rSomeRef = intOne;
10
11        intOne = 5;
12        cout « "int0ne:\tN « intOne « endl;
13        cout « "rSomeRef:\t" « rSomeRef « endl;
14        cout « "&int0ne:\t"   « &int0ne « endl;
15        cout « "&rSomeRef:\t" « &rSomeRef « endl;
16
17        int intTwo = 8;
18        rSomeRef = intTwo;  // nije ono što mislite!
19        cout « "\nint0ne:\t" « intOne « endl;
20        cout « "intTwo:\t" « intTwo « endl;
21        cout « "rSomeRef:\t" « rSomeRef « endl;
22        cout « "&int0ne:\t"   « &int0ne « endl;
23        cout « "&intTwo:\t"   « &intTwo « endl;
24        cout « "&rSomeRef:\t" « SrSomeRef « endl;
25  return 0;
26

      intOne:
      rSomeRef:
      &int0ne:              0x213e
      SrSomeRef:      0x213e

      intOne:
      intTwo:
      rSomeRef:       8
      iintOne:            0x213e
      &intTwo:            0x2130
      ArSomeRef:    0x213e
```

p-I'MHHfc- Celobrojna promenljiva i referenca na celobrojnu vrednost su deklarisane u linijama 8 i 9. Celobrojnoj promenljivoj je dodeljena vrednost 5 u liniji 11, a vrednosti i njihove adrese se štampaju u linijama 12-15.

U liniji 17 nova promenljiva, intTwo, kreira se i inicijalizuje vrednošću 8. U liniji 18 programer pokušava da ostvari ponovnu dodelu referenci rSomeRef, da sada bude alijas za promenljivu intTwo. Medutim, rSomeRef nastavlja da se ponaša kao alijas za intOne, pa je zato ova dodela potpuno ekvivalentna sledećem:

```
intOne = intTwo;
```

Kada se štampaju vrednosti promenljivih intOne i rSomeRef (linije 19-21) one su iste kao i intTwo. U stvari, kada se štampaju adrese u linijama 22-24, vidite da rSomeRef nastavlja da upućuje na intOne, a ne na intTwo.

PAZm Upotrebite reference za kreiranje alijasa za objekat.

Inicijalizujte sve reference.

Nemojte pokušati da ostvarite ponovnu dodelu reterenci.

Nemojte pomesati operator adresa od sa referencnim operatorom.

Na šta se može upućivati?

Na svaki objekat se može upućivati, uključujući korisnički definisane objekte. Uočite da kreirate referencu za objekat, a ne za klasu. Ne pišete ovo:

```
int & rIntRef = int;     // pogresno
```

Morate rIntRef inicijalizovati na odredenu celobrojnu promenljivu, kao:

```
int howBig = 200;
int & rIntRef · howBig;
```

Na isti način, ne inicijalizujete referencu na CAT:

```
CAT & rCatRef = CAT;    // pogresno
```

Morate rCatRef inicijalizovati na odredeni CAT objekat:

```
CAT frisky;
CAT & rCatRef = frisky;
```

Reference na objekte se koriste baš kao i sami objekti. Podacima clanovima i metodama Članicama se pristupa korišćenjem normalnog operatora za pristup članu klase (.) i, kao i sa ugradenim tipovima, referenca se ponaša kao alijas za objekat. Listing 9.4 ilustruje ovo.

Listing 9.4. Reference no objekte.

```
// Listing 9.4
// Reference na objekte klase

linclude <iostream.h>

class SimpleCat
{
   public:
      SimpleCat (int age, int weight);
      -SimpleCat() {}
      int GetAgeO ( return itsAge; }
      int GetWeight() { return itsWeight; }
   private:
      int itsAge;
      int itsWeight;
};

SimpleCat::SimpleCat(int age, int weight)
{
     itsAge = age;
     itsWeight = weight;


int mainQ
{
     SimpleCat Frisky(5,8);
     SimpleCat & rCat = Frisky;

     cout «  "Frisky je: ";
     cout «  Frisky.GetAgeO  «  " godina star. \n"
     cout «  "I Frisky je tezak:
     cout «  rCat.GetWeight()  « ' funti. \n";
return 0;


       Frisky je: 5 godina star.
       I Frisky je tezak 8 funti.
```

U liniji 26 Frisky je deklarisan kao SimpleCat objekat. U liniji 27 SimpleCat referenca, rCat, deklarisana je i inicijalizovana da upućuje na Frisky. U linijama 30 i 32 pristupa se SimpleCat metodima prvo korišćenjem SimpleCat objekta a, onda, SimpleCat reference. Uočite da je pristup identičan. Ponovimo: referenca je alijas za stvarni objekat.

Reference

Referencu deklarišete navođenjem tipa, zatim referencnog operatora (&), pa imena reference. Reference se moraju inicijalizovati u vreme kreiranja.

Primer 1

```
int hisAge;
int &rAge = hisAge;
```

Primer 2

```
CAT boots;
CAT &rCatRef = boots;
```

Null pokazivaci i null reference

Kada pokazivaci nisu inicijalizovani, ili kada se izbrišu, ispravno je dodeliti im nul 1 (0). Ovo ne važi za reference. U stvari, reference ne mogu biti nul 1 i program sa referencom na null objekat se smatra nevažećim. Kada je program nevažeći, skoro ništa se ne može dogoditi. Može izgledati da on radi, ili on može izbrisati sve datoteke na Vašem disku. I prvo i drugo su mogući rezultati nevažećeg programa.

Većina kompajlera će podržati nul 1 objekat, bez mnogo prigovaranja, ali će pasti ako samo pokusate da upotrebite objekat na neki način. Ipak, uzimanje ove prednosti još uvek nije dobra ideja. Kada premestite Vaš program na drugu mašinu, ili kompajler, ako imate nul 1 objekte, mogu se razviti mistični bagovi.

Predavanje funkcijskih argumenata po referenci

U Danu 5, "Funcije", naučili ste da funkcije imaju dva ograničenja: argumenti se predaju po vrednosti, a povratni iskaz može vratiti samo jednu vrednost.

Predavanje vrednosti funkciji po referenci može prevazići oba ova ograničenja. U C+ +-u predavanje po referenci se ostvaruje na dva načina: korišćenjem pokazivača i korišćenjem referenci. Sintaksa se razlikuje, ali neto efekat je isti. Umesto kreiranja kopije, unutar opsega funkcija, funkciji se predaje originalan objekat.

•yMAPOMEMAy Ako ste pročitali sekciju sa dodatnim objašnjenjem posle Dana 5, naučili ste da se funkcijama predaju njihovi parametri na steku. Kada se funkciji predaje vrednost po referenci (ili korišćenjem pokazivača, ili referenci), na stek se stavlja adresa objekta, a ne ceo objekat.

U stvari, u nekim kompjuterima adresa se drži u registru, a na stek se ništa ne stavlja. U svakom slucaju, kompajler sada zna kako da dode do originalnog objekta i promene se prave nad njim, a ne nad kopijom.

Predavanje objekta po referenci dozvoljava funkciji da promeni objekat na koji se upućuje.

Setite se da listing 5.5 iz Dana 5 demonstrira da poziv funkcije swap() ne utiče na vrednosti u pozivajućoj funkciji. Listing 5.5 je ovde reprodukovan kao listing 9.5, radi lakšeg praćenja.

Listing 9.5. Demonstriranje predavanja po vrednosti.

```
//Listing 9.5 Demonstrira predavanje po vrednosti

#include <iostream.h>

void swap(int x, int y);

int main()
{
  int x = 5, y = 10;

   cout « "Main. Pre swap, x: " « x « " y: " « y « "\n";
   swap(x.y);
   cout « "Main. Posle swap, x: " « x « " y: " « y « "\n";
return 0;
}

void swap (int x, int y)
{
   int temp;

   cout « "Swap. Pre swap, x: " « x « " y: " « y « "\n";

   temp = x;
   x = y;
   y = temp;

   cout « "Swap. Posle swap, x: " « x « " y: " « y « "\n";
```

```
Main. Pre swap, x: 5 y: 10
Swap. Pre swap, x: 5 y: 10
Swap. Posle swap, x: 10 y: 5
Main. Posle swap, x: 5 y: 10
```

Ovaj program inicijalizuje dve promenljive u mainQ, a onda ih predaje funkciji swapO, koja izgleda kao da ih zamenjuje. Kada se ponovo ispitaju u mainQ , one su nepromenjene.

Problem je taj što s e x i y predaju funkciji swapQ po vrednosti. To znači da su u funkciji napravljene lokalne kopije. Ono što Vi želite je da x i y predate po referenci.

U C + + - U postoje dva načina da se reši ovaj problem: možete kao parametre funkcije swapQ upotrebiti pokazivače na originalne vrednosti, ili možete predati reference na originalne vrednosti.

Kako da swap() radi sa pokazivačima

Kada predajete pokazivac, predajete adresu objekta i tako funkcija može manipulisati vrednošću na toj adresi. Da bi swapQ promenila stvarne vrednosti, korišćenjem pokazivača, funkcija, swapQ, trebalo bi da bude deklarisana da prihvata dva int pokazivača. Zatim, dereferenciranjem pokazivača, vrednosti x i y će, u stvari, biti promenjene. Listing 9.6 demonstrira ovu ideju.

Listing 9.6. Predavanje po referenci korišćenjem pokazivača.

```
1   //Listing 9.6 Demonstrira predavanje po referenci
2
3    linclude  <iostream.h>
4
5    void swap(int *x, int *y);
6
7    int main()
8    {
9      int x = 5, y = 10;
10
11      cout « "Main. Pre swap, x: " « x « " y: " « y « "\n";
12      swap(&x,&y);
13      cout « "Main. Posle swap, x: " « x « " y: " « y « "\n";
14   return 0;
15
16
17    void swap (int *px, int *py)
18    {
19      int temp;
20
21      cout « "Swap. Pre swap, *px: " « *px « " *py: " « *py « "\n";
22
23      temp = *px;
24      *px = *py;
25      *py = temp;
26
27      cout « "Swap. Posle swap, *px: " « *px « " *py: " « *py « "\n"
28
29
```

```
Main. Pre swap, x: 5 y: 10
Swap. Pre swap, *px: 5 *py: 10
Swap. Posle swap, *px: 10 *py: 5
Main. Posle swap, x: 10 y: 5
```

Uspeh! U liniji 5 prototip funkcije swap () je promenjen da pokazuje da će njegova dva parametra biti pokazivaci na int, a ne na int promenljive. Kada se pozove swap() u liniji 12, kao argumenti se predaju adrese od x i y.

U liniji 19 lokalna promenljiva, temp, deklarisana je u funkciji swap() - nije potrebno da temp bude pokazivac; ona će čuvati samo vrednost od *px (to jest, vrednost promenljive x u pozivajućoj funkciji), tokom života funkcije. Posle povratka iz funkcije, temp vise neće biti potrebna.

U liniji 23 promenljivoj temp se dodeljuje vrednost na px. U liniji 24 vrednost na px se dodeljuje vrednosti na py. U liniji 25 vrednost čuvana u temp (to jest, originalna vrednost na px) stavlja se u py.

Neto efekat ovoga je da su vrednosti u pozivajućoj funkciji, čija adresa je bila predata funkciji swap(), u stvari, zamenjene.

Implementiranje swap() sa referencama

Prethodni program radi, ali sintaksa funkcije swap() je problematična zbog dve stvari. Prvo, ponavljajuća potreba za dereferenciranjem pokazivača unutar funkcije swapQ čini ga podložnim greškama i teškim za čitanje. Drugo, potreba da se preda adresa promenljivih u pozivajućoj funkciji čini unutrašnji rad funkcije swap() vise nego očiglednim za korisnike.

Cilj C + +-a je da spreči korisnika funkcije da se brine o tome kako ona radi. Predavanje preko pokazivača stavlja teret na pozivajuću funkciju, umesto tamo gde on pripada - na pozvanu funkciju. Listing 9.7 prepisuje funkciju swap(), korišćenjem referenci.

Listing 9.7. Prepisana funkcija swapQ sa referencama.

```
//Listing 9.7 Demonstrira predavanje po referenci
 // korišćenjem referenci!

   linclude <iostream.h>

   void swap(int &x, int &y);

   int main()
   {
        int x = 5, y = 10;

        cout « "Main. Pre swap, x: " « x « " y: " « y << "\n";
         swap(x.y);
         cout « "Main. Posle swap, x: " « x « " y: " « y << "\n";
return 0;
       }
```



```
void swap (int &rx, int &ry)
{
  int temp;

     cout « "Swap. Pre swap, rx: " « rx « " ry: " « ry « "\n";

     temp = rx;
     rx = ry;
     ry = temp;

     cout « "Swap. Posle swap, rx: " « rx « " ry: " « ry « "\n"

 }
```

```
Main. Pre swap, x:5 y: 10
Swap. Pre swap, rx:5 ry:10
Swap. Posle swap, rx:10 ry:5
Main. Posle swap, x:10, y:5
```

*.....mmmmmCj/^* Kao i u primeru sa pokazivačima, deklarisane su dve promenljive u liniji 10, a njihove vrednosti se štampaju u liniji 12. U liniji 13 poziva se funkcija swap(), ali uočite da se predaju x i y, a ne njihove adrese. Pozivajuća funkcija, jednostavno, predaje promenljive.

Kada se pozove swap(), izvršenje programa "skače" na liniju 18, gde se promenljive identifikuju kao reference. Njihove vrednosti se štampaju u liniji 22, ali uočite da se ne zahtevaju posebni operatori. Oni su alijasi za originalne vrednosti i mogu se samostalno koristiti.

U linijama 24-26 vrednosti se zamenjuju, a onda se štampaju u liniji 28. Izvršenje programa "skače" u pozivajuću funkcuju i u liniji 14 vrednosti se štampaju u mai n( ). Zato što su parametri za swap() deklarisani kao reference, vrednosti iz mainQ se predaju po referenci i time se, takode, menjaju i u mainQ .

Reference obezbeđuju pogodnost i lakoću upotrebe normalnih promenljivih, sa snagom i sposobnošću predavanja po referenci, koju imaju pokazivaci.

Razumevanje funkcijskih zaglavlja i prototipova

Listing 9.6 prikazuje swapQ koja koristi pokazivače, a listing 9.7 prikazuje swapQ koja koristi reference. Korišćenje funkcije koja prihvata reference je lakše i kod je lakši za čitanje, ali kako pozivajuća funkcija zna da li se vrednosti predaju po referenci, ili po vrednosti? Kao klijent funkcije swapQ, programer mora osigurati da će swapQ, u stvari, promeniti parametre.

Ovo je druga upotreba za prototip funkcije. Ispitivanjem parametara, deklarisanih u prototipu, koji se, obično, nalazi u zaglavlju, zajedno sa svim drugim prototipovima, programer zna da se vrednosti predate funkciji swapQ predaju po referenci i da će time biti zamenjene ispravno.

Da je swap() bila funkcija članica klase, deklaracija klase, takođe raspoloživa u zaglavlju, obezbedila bi ovu informaciju.

U C + + - U klijenti klase i funkcija se oslanjaju na zaglavlje za saopštavanje svega što je potrebno; ono se ponaša kao interfejs za klasu, ili funkciju. Stvarna implementacija je skrivena od klijenta. Ovo omogućava programeru da blisko fokusira problem i da koristi klasu, ili funkciju, bez brige o tome kako ona radi.

Kada je Colonel John Roebling dizajnirao Bruklinski most, brinuo je o detaljima: izmedu ostalog, kako je beton izliven i kako je proizvedena žica za most. On je bio uključen u matematičke i hemijske procese, potrebne za kreiranje materijala za gradnju mosta. Danas ipak, inženjeri efikasnije troše svoje vreme, korišćenjem dobro poznatih gradivnih materijala, ne brinući kako ih je proizvođač napravio.

Cilj C+ +-a je da dozvoli programerima da se oslone na dobro poznate klase i funkcije, bez pogleda na njihov interni rad. Ovi "komponentni delovi" se mogu povezati, da bi proizveli program, što je veoma slično načinu na koji su žice, cevi, stege i drugi delovi povezani, da bi proizveli zgrade i mostove.

Na veoma sličan način na koji inženjer ispituje specifikacioni list za cev, da bi odredio njen kapacitet izdržljivosti, volumen, veličinu smeštanja i tako dalje, C + + programer čita interfejs funkcije, ili klase, da bi odredio koje usluge ona obezbeduje, koje parametre prihvata i koje vrednosti vraća.

Vraćanje višestrukih vrednosti

Kao što je rečeno, funkcije mogu vratiti samo jednu vrednost. Šta učiniti ako je potrebno da vratite dve vrednosti iz funkcije? Jedan način da rešite ovaj problem je da funkciji predate dva objekta po referenci. Funkcija onda može popuniti objekte ispravnim vrednostima. Pošto predavanje po referenci dozvoljava funkciji da promeni originalne objekte, ovo efektno dozvoljava funkciji da vrati dve informacije. Ovaj pristup zaobilazi povratnu vrednost funkcije, koja se može rezervisati za izveštavanje o greškama.

Ovo se može ostvariti referencama, ili pokazivačima. Listing 9.8 demonstrira funkciju koja vraća tri vrednosti: dve kao pokazivačke parametre i jednu kao povratnu vrednost funkcije.

Listing 9.8. Vroćanje vrednosti sa pokazivačima.

```
//Listing 9.8

// Vraćanje vise vrednosti iz funkcije

linclude <iostream.h>

typedef unsigned short USHORT;

short Factor(USH0RT, USHORT*, USHORT*);
```



```
9
10  int main()
11  {
12     USHORT number, squared, cubed;
13     short error;
14
15     cout « "Unesite broj (0-20): ";
16     cin » number;
17
18     error = Factor(number, isquared, &cubed);
19
20     if (!error)
21     {
22         cout « "broj: " « number « "\n";
23         cout « "kvadrat: " « squared « "\n";
24         cout « "kub: " « cubed « "\n";
25     }
26     else
27     cout « "Doslo je do greske!!\n";
28   return 0;
29
30
31  short Factor(USH0RT n, USHORT *pSquared, USHORT *pCubed)
32  {
33  short Value = 0;
34     if (n > 20)
35        Value = 1;
36     else
37     {
38         *pSquared = n*n;
39         *pCubed = n*n*n;
40         Value = 0;
41     }
42     return Value;
43
```

```
Unesite broj (0-20): 3
broj: 3
kvadrat: 9
kub: 27
```

U liniji 12 number, squared i cubed su definisani kao USHORT. Promenljivoj number se dodeljuje broj baziran na korisničkom ulazu. Ovaj broj i adresa od squered i cubed se predaju funkciji Factor().

Factor () ispituje prvi parametar, koji se predaje po vrednosti. Ako je on veći od 20 (maksimalan broj kojim ova funkcija može rukovati), ona postavlja return Value (engl. return value = povratna vrednost) na jednostavnu vrednost greške. Uočite da je povratna vrednost iz funkcije Factor() rezervisana ili za ovu povratnu vrednost,

ili za vrednost 0, koja pokazuje da je sve prošlo dobro, i uočite da funkcija vraća ovu vrednost u liniji 42.

Stvarne potrebne vrednosti, kvadrat i kub promenljive number, vracaju se ne korišćenjem povratnog mehanizma, nego menjanjem pokazivača koji su predati funkciji.

U linijama 38 i 39 pokazivačima se dodeljuju njihove povratne vrednosti. U liniji 40 return Value dobija uspešnu vrednost. U liniji 41 return Value se vraća.

Jedno poboljšanje u ovom programu bi moglo biti deklarisanje sledećeg:

```
enum ERRORJALUE { SUCCESS, FAILURE};
```

Zatim, umesto vraćanja 0, ili 1, program bi mogao da vrati SUCCESS, ili FAILURE.

Vraćanje vrednosti po referenci

Iako listing 9.8 radi, on se može učiniti lakšim za čitanje i održavanje, korišćenjem referenci, umesto pokazivača. Listing 9.9 prikazuje isti program, prepisan radi korišćenja referenci i uključivanja ERROR enumeracije.

Listing 9.9. Listing 9.8 prepisan korišćenjem referenci.

```
1:    //Listing 9.9
2:     // Vraćanje vise vrednosti iz funkcije
3:     // korišćenje referenci
4:
5:     linclude <iostream.h>
6:
7:     typedef unsigned short USHORT;
8:     enum ERR_C0DE { SUCCESS, ERROR };
9:
10:     ERR_C0DE Factor(USHORT, USH0RT&, USH0RT&);
11:
12:     int main()
13:     {
14:          USHORT number, squared, cubed;
15:          ERR_C0DE result;
16:
17:          cout « "Unesite broj (0 - 20): ";
18:          cin » number;
19:
20:          result = Factor(number, squared, cubed);
21:
22:          if (result == SUCCESS)
23:          {
24:               cout « "broj: " « number « "\n";
25:               cout « "kvadrat: " « squared « "\n";
26:               cout « "kub: "  « cubed   « "\n";
27:          }
```



```
28:          else
29:          cout « "Doslo je do greske!!\n";
30:     return 0;
31:     }
32:
33:     ERR_C0DE Factor(USH0RT n, USHORT &rSquared, UShGRT -.rCuaed)
34:     {
35:          if (n > 20)
36:               return ERROR;    // jednostavan kod za grešku
37:          else
38:          {
39:               rSquared = n*n;
40:               rCubed = n*n*n;
41:               return SUCCESS;
42:          }
43:     }
```

```
ja H Г У ^ Unesite broj (0 -20): 3
           broj: 3
           kvadrat: 9
           kub: 27
```

ˇm>m,^ik Listing 9.9 je identičan listingu 9.8, uz dva izuzetka. Enumeracija ERR_C0DE čini izveštavanje o grešci vise eksplicitnim u linijama 36 i 41, kao i rukovanje greškama u liniji 22.

Ipak, veća promena je ta da je Factor() sada deklarisana da prihvata reference na squared i cubed, a ne na pokazivace, što čini manipulaciju ovim paramctrima daleko jednostavnijom i lakšom za razumevanje.

Predavanje po referenci, zbog efikasnosti

Svaki put kada predate objekat funkciji po vrednosti, pravi se njegova kopija. Svaki put kada vratite objekat iz funkcije po vrednosti, pravi se druga kopija.

U sekciji "Dodatno objašnjenje", na kraju Dana 5, naučili ste da se ovi objekti kopiraju na stek. Zato ovo uzima vreme i memoriju. Za male objekte, kao što su ugradene celobrojne vrednosti, to je trivijalna cena.

Ipak, za veće korisnički kreirane objekte, cena je veća. Veličina korismčki kreiranih objekata na steku je suma svih njegovih promenljivih članica. Svaka od njih, dalje, može biti korisnički kreiran objekat i predavanje tako masivne strukture njenim kopiranjem na stek može biti veoma skupo zbog performansi i korišćenja memorije.

Postoji, takođe, i druga cena. Sa klasama koje kreirate, svaka ova privremena kopija se kreira kada kompajler pozove specijalan konstruktor: *konstruktor kopije*. Sutra ćete naučiti kako konstruktori kopije rade i kako možete napraviti sopstveni, ali, za sada, dovoljno je da znate da se konstruktor kopije poziva svaki put kada se privremena kopija objekta stavi na stek.

Kada se privremeni objekat uništava, što se dešava po povratku iz funkcije, poziva se destruktor objekta. Ako se neki objekat vraća iz funkcije po vrednosti, takode se mora napraviti i uništiti kopija tog objekta.

Sa velikim objektima, ovi pozivi konstruktora i destruktora mogu biti skupi zbog smanjenja brzine i povećanja korišćenja memorije. Radi ilustracije ove ideje, u listingu 9.9 kreiran je pojednostavljen korisnički kreiran objekat: SimpleCat. Pravi objekat bi bio veći i skuplji, ali ovo je dovoljno da bi se pokazalo koliko često se pozivaju konstruktor kopije i destruktor.

Listing 9.10 kreira SimpleCat objekat, a, onda, poziva dve funkcije. Prva funkcija prima Cat po vrednosti, a potom ga vraca po vrednosti. Druga prima pokazivac na objekat, a ne sam objekat, i vraća pokazivac na objekat.

Listing 9.10. Predavanje objekata po referenci.

```
//Listing 9.10
// Predavanje pokazivača na objekte

linclude <iostream.h>

class SimpleCat
{
public:
        SimpleCat ();                   //  konstruktor
        SimpleCat(SimpleCat&);          // konstruktor kopije
        -SimpleCat();                   //  destruktor


 iimpleCat: :SimpleCat()

        cout « "Jednostavan Cat konstruktor...\n";


 limpleCat: :SimpleCat(SimpleCat&)

        cout « "Jednostavan Cat Copy konstruktor...\n"


 impleCat: :~SimpleCat()

        cout « "Jednostava Cat destructor...\n";


 impleCat FunctionOne (SimpleCat theCat);
 impleCat* FunctionTwo (SimpleCat *theCat);

 nt main()

        cout « "Kreiranje cat...\n";
```



```
35:         SimpleCat Frisky;
36:         cout « "Poziv FunctionOne.. An"
37:         FunctionOne(Frisky);
38:         cout « "Poziv FunctionTwo..An"
39:         FunctionTwo(&Frisky);
40:     return 0;
41:
42:
43:  // FunctionOne, predaje po vrednosti
44:  SimpleCat FunctionOne(SimpleCat theCat)
45:  {
46:                 cout « "Function One. Povratak...\n"
47:                 return theCat;
48:  }
49:
50:  // functionTwo, predaje po referenci
51:  SimpleCat* FunctionTwo (SimpleCat *theCat)
52:  {
53:                 cout « "Function Two. Povratak..An"
54:                 return theCat;
55:
```

```
Krei ranje cat...
Jednostavan cat konstruktor...
Poziv FunctionOne...
Jednostavan Cat Copy konstructor...
Function One. Povratak...
Simple Cat Copy Constructor...
Simple Cat Destructor...
Simple Cat Destructor...
9   Calling FunctionTwo...
10: Function Two. Returning...
11: Simple Cat Destructor...
```

yNAPOMENAy Brojevi linija se neće odštampati. Oni su dodati, radi pomoći u analizi.

Veoma pojednostavljena klasa `SimpleCat` je deklarisana u linijama 6-12. Konstruktor, konstruktor kopije i -destruktor štampaju informativnu poruku, tako da možete znati kada su oni pozvani.

U liniji 34 `main()` štampa poruku, a ona se vidi u izlazu 1. U liniji 35 instancira se objekat `SimpleCat`. Ovo prouzrokuje da se pozove konstruktor, a izlaz iz konstruktora se vidi na izlaznoj liniji 2.

U liniji 36 `main()` izveštava da poziva funkciju `FunctionOne`, što kreira izlaznu liniju 3. Zato što se `FunctionOnef)` poziva predavanjem `SimpleCat` objekta po vrednosti, kopija `SimpleCat` objekta se pravi na steku, kao lokalni objekat za pozvanu funkciju. Ovo prouzrokuje da se pozove konstruktor kopije, što kreira izlaznu liniju 4.

Izvršenje programa skače na liniju 46 u pozvanoj funkciji, koja štampa informativnu poruku, izlaznu liniju 5. Zatim se ostvaruje povratak iz funkcije i ona vraća SimpleCat objekat po vrednosti. Ovo kreira drugu kopiju objekta, pozivajući konstruktor kopije i proizvodeći liniju 6.

Povratna vrednost iz FunctionOneO se ne dodeljuje ni jednom objektu i tako privremeno kreirani za povratak se baca, pozivajući destruktor, što proizvodi izlaznu liniju 7. Pošto je FunctionOneO završena, njena lokalna kopija izlazi iz opsega i uništava se, poziva se destruktor i proizvodi linija 8.

Izvrsenje programa se vraća u main(), a poziva se FunctionTwo(), ali se parametar predaje po referenci. Ne proizvodi se kopija, pa, zato, nema izlaza. FunctionTwoQ štampa poruku, koja se pojavljuje kao izlazna linija 10, a, onda, vraća SimpleCat objekat, ponovo po referenci, i tako ponovo ne proizvodi pozive konstruktora i destruktora.

Konačno, program se završava i Frisky izlazi iz opsega, prouzrokujući finalni poziv destruktora i štampajući izlaznu liniju 11.

Neto efekat je sledeći: daje poziv funkcije FunctionOneO , jer je ona predala mačku po vrednosti, proizveo je dva poziva konstruktora kopije i dva destruktora, dok poziv funkcije FunctionTwo() nije proizveo ni jedan.

Predavanje const pokazivača

Iako je predavanje pokazivača funkciji FunctionTwo0 efikasnije, ono je opasno. Ne planira se da funkciji FunctionTwoO bude dozvoljeno da promeni SimpleCat objekat, koji joj se predaje, ali joj se još uvek daje adresa od SimpleCat. Ovo ozbiljno izlaže objekat promeni i narušava zaštitu koja je ponuđena u predavanju po vrednosti.

Predavanje po vrednosti je kao davanje muzeju fotografije Vašeg remek dela, umesto konkretne stvari. Ako ga vandali oštete, original nije oštećen! Predavanje po referenci je kao slanje Vaše kućne adrese muzeju i pozivanje gostiju da navrate i pogledaju pravu stvar.

Rešenje je da se preda const pokazivac na SimpleCat. Ovo sprečava pozivanje svake ne - const metode za SimpleCat i time se štiti objekat od promene. Listing 9.11 demonstrira ovu ideju.

Listing 9.11. Predavanje const pokazivača.

```
    //Listing 9.11
2       // Predavanje pokazivača na objekte
3
4         #include <iostream.h>
5
6         class SimpleCat
7         {
8         public:
```



```
9:              SimpleCat();
10:             SimpleCat(SimpleCat&);
11:             -SimpleCat ();
12:
13:             int GetAgeO const { return itsAge; }
14:             void SetAge(int age) { itsAge = age; }
15:
16:       private:
17:             int itsAge;
18:
19:
20:         impleCat: :SimpleCat()
21:
22:               cout « "Simple Cat Constructor...\n"
23:               itsAge = 1;
24:
25:
26:         impleCat: :SimpleCat(SimpleCat&)
27:
28:               cout « "Simple Cat Copy Constructor..An";
29:
30:
31:         impleCat::-SimpleCat()
32:
33:               cout « "Simple Cat Destructor..An";
34:
35:
36: const SimpleCat * const FunctionTwo (const SimpleCat * const theCat);
37:
38:         int main()
39:
40:               cout « "Making a cat..An";
41:               SimpleCat Frisky;
42:               cout « "Frisky is " ;
43                cout « Frisky.GetAgeO;
44:               cout « " years old\n";
45:               int age = 5;
46:               Frisky.SetAge(age);
47:               cout « "Frisky is " ;
48                cout « Frisky.GetAgeO;
49:               cout « " years old\n";
50:               cout « "Calling FunctionTwo..An"
51:               FunctionTwo(&Frisky);
52:               cout « "Frisky is " ;
53                cout « Frisky.GetAgeO;
54:               cout « " years old\n";
55:   return 0;
56:
57:
```

nastavlja se

nastavak

шшшшмт **Listing 9.11. Predavanje const pokazivača.**

```
58:    // functionTwo, predaje const pokazivac
59:    const SimpleCat * const FunctionTwo (const SimpleCat * const theCat)
60:    {
61:            cout « "Function Two. Returning...\n";
62:            cout « "Frisky is now " « theCat->GetAge();
63:            cout « " years old \n";
64:            // theCat->SetAge(8);    konstantan!
65:            return theCat;
66:          }
```

```
Making a cat...
Simple Cat constructor...
Frisky is 1 years old
Frisky is 5 years old
Calling FunctionTwo...
FunctionTwo. Returning...
Frisky is now 5 years old
Frisky is 5 years old
Simple Cat Destructor...
```

SimpleCat je dodala dve funkcije pristupa, GetAgeO u liniji 13, koja je const funkcija, i SetAge() u liniji 14, koja nije const funkcija. Takođe je dodala promenljivu članicu itsAge u liniji 17.

Konstruktor, konstruktor kopije i destruktor su još uvek definisani da štampaju svoje poruke. Ipak, konstruktor kopije se nikada ne poziva, zato što se objekat predaje po referenci, pa se ne prave kopije. U liniji 41 kreira se objekat i štampa se njegova podrazumevana starost, počevši od linije 42.

U liniji 46 itsAge se postavlja korišćenjem metoda pristupa SetAge, a rezultat se štampa u liniji 47. FunctionOne se ne koristi u ovom programu, ali FunctionTwof) se poziva. FunctionTwoO je promenjena malo; parametar i povratna vrednost su sada deklarisani, u liniji 36, da uzimaju konstantni pokazivac na konstantni objekat i da vrate konstantan pokazivac na konstantan objekat.

Zato što se parametar i povratna vrednost još uvek predaju po referenci, ne prave se kopije i konstruktor kopije se ne poziva. Ipak, pokazivac u FunctionTwoO je sada konstantan i zato ne može pozvati ne-const metod, SetAge0. Da poziv funkcije SetAge() u liniji 64 nije isključen komentarom, program se ne bi kompajlirao.

Uočite da objekat kreiran u main() nije konstantan, a Frisky može pozvati SetAgeO . Adresa ovog nekonstantnog objekta se predaje funkciji FunctionTwoO, ali zato što deklaracija funkcije FunctionTwoO deklariše pokazivac kao konstantan, objekat se tretira kao konstantan!

Reference kao alternative

Listing 9.11 rešava problem kreiranja dodatnih kopija i time štedi pozive konstruktora kopije i destruktora. On koristi konstantne pokazivače na konstantne objekte i tako rešava problem funkcije koja menja objekat. Ipak, on je još uvek u izvesnoj meri nezgodan, zato što su objekti predati funkciji pokazivaci.

Pošto znate da objekat neće nikada biti null, bilo bi lakše raditi sa njim u funkciji kada bi referenca bila predata, a ne pokazivac. Listing 9.12 ilustruje ovo.

Listing 9.12. Predavanje referenci na objekte.

```
//Listing 9.12
// Predavanje referenci na objekte

linclude <iostream.h>

class SimpleCat
(
public:
        SimpleCat();
         SimpleCat(SimpleCat*);
         -SimpleCat ();

         int GetAgeO const { return itsAge; }
         void SetAge(int age) { itsAge = age; }

 private:
         int itsAge;


    SimpleCat::SimpleCat()
    {
          cout « "Simple Cat Constructor...\n"
          itsAge = 1;


    SimpleCat::SimpleCat(SimpleCat&)
    {
          cout « "Simple Cat Copy Constructor..An";
    }

    SimpleCat::-SimpleCat()
    {
          cout « "Simple Cat Destructor..An";

    }

    const     SimpleCat & FunctionTwo (const SimpleCat & theCat);
```

nastavljase

Listing 9.12. Predavanje referenci na objekte. *nastavak*

```
38      int  main()
39      {
40            cout « "Making a cat.. An";
41            SimpleCat  Frisky;
42            cout « "Frisky is  «  Frisky.GetAgeO      years old\n";
43            int age = 5;
44            Frisky.SetAge(age);
45            cout « "Frisky is " « Frisky.GetAgeO      years old\n"
46            cout « "Calling  FunctionTwo..An";
47            FunctionTwo(Frisky);
48            cout « "Frisky is " « Frisky.GetAgeO      years old\n"
49      return 0;
50      }
51
52      // functionTwo,  predaje  referencu  na  const  objekat
53      const SimpleCat & FunctionTwo (const SimpleCat & theCat)
54      {
55                  cout « "Function  Two.  Returning..An";
56                  cout « "Frisky  is  now  " « theCat.GetAge();
57                  cout « " years old \n";
58                  // theCat.SetAge(8);    konstantan!
59                  return theCat;
60
```

```
Making a cat...
Simple  Cat  constructor...
Frisky is 1 years old
Frisky is 5 years old
Cal1ing  FunctionTwo...
FunctionTwo.  Returning...
Frisky is now 5 years old
Frisky is 5 years old
Simple  Cat  Destructor...
```

Izlaz je identičan onom koji je proizveden u listingu 9.11. Jedina značajna promena je da `FunctionTwo()` sada prihvata i vraća referencu na konstantan objekat. Rad sa referencama je jednostavniji od rada sa pokazivačima, a postižu se iste uštede i efikasnost, kao i sigurnost, korišćenjem const.

const reference

C++ programed, obično, ne razlikuju "konstantnu referencu na `SimpleCat` objekat" od "reference na konstantan `SimpleCat` objekat". Samim referencama se nikada ne može ostvariti ponovna dodela da bi one upućivale na drugi objekat, pa su, zato, one uvek konstantne. Ako se reč const primeni na referencu, to će objekat na koji se upućuje učiniti konstantnim.

Kada koristiti reference, a kada pokazivače?

C++ programeri izrazito vise vole reference. One su čistije i lakše za korišćenje i obavljaju bolje posao skrivanja informacija, kao što smo videli u prethodnom primeru.

Nad referencama se ne može ostvariti ponovna dodela. Ako je potrebno da prvo pokazujete na jedan objekat, a, onda, na drugi, morate koristiti pokazivac. Reference ne mogu biti `nul 1`, pa, ako postoji šansa da objekat na koji upućuju može biti `nul 1`, ne smete koristiti referencu. Morate koristiti pokazivac.

Na kraju, recimo nešto i o operatoru `new`. Ako new ne može alocirati memoriju na slobodnom skladištu, on vraća `nul 1` pokazivac. Pošto referenca ne može biti `nul 1`, Vi ne smete inicijalizovati referencu na ovu memoriju, dok ne ispitate da li je ona `null`. Sledeći primer prikazuje kako da rukujete ovim:

```
int *plnt = new int;
if (pint != NULL)
int SrInt = *plnt
```

U ovom primeru pokazivac na int, pint, je deklarisan i inicijalizovan sa memorijom koja je vraćena operatorom `new`. Adresa u pint se testira i, ako nije `null`, pint se dereferencira. Rezultat dereferenciranja int promenljive je `int` objekat, a `rInt` se inicijalizuje da upućuje na `taj` objekat. Tako, rlnt postaje alijas na `int`, koji je vraćen operatorom `new`.

<g| **PAZITE** |£ Predajte parametre po referenci, uvek kada je to moguće.

Vratite po referenci uvek, kada je to moguće.

Nemojte koristiti pokazivače, ako će reference raditi.

Upotrebite const da biste zaštitili reference i pokazivače, uvek kada je to moguće.

Nemojte vratiti referencu na lokalni objekat.

Mešanje referenci i pokazivača

Savršeno je valjano da deklarišete i pokazivače i reference u listi parametara iste funkcije, zajedno sa objektima koji se predaju po vrednosti. Evo primera:

```
CAT * SomeFunction (Person &the0wner, House *theHouse, int age);
```

Ova deklaracija govori da `SomeFunction` prihvata tri parametra. Prvi je referenca na `Person` objekat, drugi je pokazivac na `House` objekat, a treći je celobrojna promenljiva. `SomeFunction` vraća pokazivac na `CAT` objekat.

^**MAPOMENA**^ Pitanje gde staviti referencni operator (&), ili operator indirekcije (*) pri deklarisanju ovih promenljivih je velika kontraverza. Legalno možete napisati bilo sta od sledećeg:

```
1: CAT& rFrisky;
```

```
2: CAT & rFrisky;
```

```
3: CAT &rFrisky;
```

Razmak se potpuno ignorile, pa zato na svakom mestu gde ga vidite možete staviti koliko god želite razmaka, tabova i novih linija.

Ostavljajud po strani slobodu izražavanja u izrazu, koji je najbolji? Evo argumenata za sva tri:

Argument za slučaj **1** je da je promenljiva čije ime je rFrisky i o čijem tipu se moze razmišljati kao o "referenci na CAT objekat". Ovaj argument kaže da bi & trebalo da bude sa tipom.

Kontraargument je da je CAT tip. & je deo "deklaratora", koji uključuje ime promenljive i ampersand. Još je vainije da kada je & blizu CAT to može dovesti do sledećeg baga:

Površno ispitivanje ove linije bi dovelo do pretpostavke da su i rFrisky i rBoots reference na CAT objekte, ali to bi bilo pogresno. Ovo stvarno govori da je rFrisky referenca na CAT, a rBoots (uprkos njenom imenu) nije referenca, nego jednostavna stara CAT promenljiva. Ovo bi trebalo biti prepisano kao:

Odgovor na ovu primedbu je da deklaracije referenci i promenljivih nikada ne bi trebalo da budu kombinovane na ovaj nacln. Evo pravog odgovora:

Konačno, mnogi programeri ne poštuju ni jedan argument i slede sredinu, stavljanje & u sredinu, kao što je ilustrovano u slučaju 2.

Naravno, sve sto je do sada rečeno o referencnom operatoru (&) odnosi se isto tako na operator indirekcije (*). Izaberite stil koji je dobar za Vas i budite dosledni u svakom programu; jasnoća jeste, i ostaje, cilj.

Ova knjiga će usvojiti dve konvencije za deklarisanje referenci i pokazivača:

1. stavljanje ampersanda i asteriska u sredinu, sa razmakom na obe strane
2. nikada ne deklarisati reference, pokazivače i promenljive u istoj liniji.

Nemojte vraćati referencu na objekat koji nije u opsegu!

Pošto C + + programeri nauče da predaju po referenci, teže da rade šta im "padne na pamet". Ipak, moguće je, preterati. Zapamtite da je referenca uvek alijas na neki drugi objekat. Ako predate referencu funkciji, ili iz funkcije, pitajte sebe: "Šta je objekat za koji koristim alijas i da li će on još uvek postojati svaki put kada bude korišćen?"

Listing 9.13. Vraćanje reference na nepostojeći objekat.

```
// Listing 9.13
 // Vraćanje reference na objekat
 // koji vise ne postoji
```



```
#include <iostream.h>

class SimpleCat
{
public:
       SimpleCat (int age, int weight);
       -SimpleCat() {}
       int GetAgeO { return itsAge; }
       int GetWeight() { return itsWeight; }
 private:
      int itsAge;
      int itsWeight;
};

 SimpleCat::SimpleCat(int age, int weight):
 itsAge(age), itsWeight(weight) {}

 SimpleCat &TheFunction();

 int main()
 {
      SimpleCat &rCat = TheFunction();
      int age = rCat.GetAgeO;
      cout « "rCat is " « age « " years old!\n"
return 0;


 SimpleCat &TheFunction()
 {
      SimpleCat Frisky(5,9);
      return Frisky;

     Compile error: Attempting to return a reference to a local object!
```

vUPOZORENjr> Ovaj program se neće kompajlirati na Borlandovom kompajleru. Kompajliraće se na Microsoftovim kompajlerima, ipak, trebalo bi uočiti da je to loša praksa kodiranja.

U linijama 7-17 deklarisana je SimpleCat. U liniji 26 inicijalizuje se referenca na SimpleCat sa rezultatima pozivanja funkcije TheFunction(), koja je deklarisana u liniji 22 da vrati referencu na SimpleCat.

Telo funkcije `TheFunction()` deklariše lokalni objekat tipa `SimpleCat` i inicijalizuje njegovu starost i težinu. Onda vraća lokalni objekat po referenci. Neki kompajleri su dovoljno pametni da uhvate ovu grešku i neće Vam dozvoliti da izvršite program. Drugi će Vam dozvoliti da izvršite program, što može izazvati nepredvidive rezultate.

Po povratku iz funkcije TheFunctionO, lokalni objekat, Frisky, biće uništen (bezbolno, uveravam Vas). Referenca vraćena iz ove funkcije će biti alijas na nepostojeći objekat, a to je lose.

Vracanje reference na objekat sa gomile

Možda ćete biti iskušani da rešite problem u listingu 9.13, time što će TheFunctionO kreirati Friskyja na gomili (engl. heap = gomila). Na taj način, po povratku iz TheFunctionO, Frisky će još uvek postojati.

Pri ovakvom pristupu javlja se problem: šta radite sa memorijom koja je alocirana za Frisky kada završite sa njim? Listing 9.14 ilustruje ovaj problem.

Listing 9.14. Memorijske pukotine.

```
1:    // Listing 9.14
2:    // Rešenje memorijskih pukotina
3:    linclude <iostream.h>
4:
5:    class SimpleCat
6:    {
7:    public:
8:          SimpleCat (int age, int weight);
9:          -SimpleCat() {}
10:         int GetAgeO { return itsAge; }
11:         int GetweightO { return itsWeight; }
12:
13    private:
14:         int itsAge;
15:         int itsWeight;
16:   };
17:
18:   SimpleCat::SimpleCat(int age, int weight):
19:   itsAge(age), itsWeight(weight) {}
20:
21:   SimpleCat & TheFunctionO;
22:
23:   int main()
24:   {
25:         SimpleCat & rCat = TheFunctionO;
26:         int age = rCat.GetAgeO ;
27:         cout « "rCat is " « age « " years old!\n";
28:         cout « "&rCat: " « &rCat « endl;
29:         // Kako da se oslobodite te memorije?
30:         SimpleCat * pCat = &rCat;
31:         delete pCat;
32:         // Uh oh, rCat sada upućuje na ??
33:   return 0;
34:   }
```



```
35:
36:   SimpleCat &TheFunction()
37:   {
38:         SimpleCat * pFrisky = new SimpleCat(5,9);
39:         cout « "pFrisky: " « pFrisky « endl;
40:         return *pFrisky;
41:   }
```

III EEfcfr
```
pFrisky:  0x2bf4
rCat is 5 years old!
&rCat:  0x2bf4
```

^KPOZORENJEjr Ovo se kompajlira, povezuje i radi. Ali to je tempirana bomba, spremna da bude aktivirana.

%±ШjA& TheFunctionO je promenjena tako da vise ne vraća referencu na lokalnu promenljivu. Memorija se alocira na slobodnom skladištu i dodeljuje pokazivaču u liniji 38. Štampa se adresa koju čuva pokazivac, a onda se pokazivac dereferencira i SimpleCat objekat se vraća po referenci.

U liniji 25 vraćanje funkcije TheFunctionO se dodeljuje referenci na SimpleCat i taj objekat se koristi za dobijanje mačkine starosti, koja se štampa u liniji 27.

Da bi se dokazalo da referenca deklarisana u main() upućuje na objekat koji je stavljen na slobodno skladište u TheFunctionO, primenjuje se operator adresa od na referencu rCat. Prikazuje se adresa objekta na koga ona upućuje i ona odgovara adresi objekta na slobodnom skladištu.

Do sada, sve je dobro. Ali kako će ta memorija biti oslobođena? Ne možete pozvati delete za referencu. Jedno pametno rešenje je da se kreira drugi pokazivac i da se inicijalizuje adresom dobijenom iz rCat. Ovo briše memoriju i popunjava memorijsku pukotinu. Ipak, postoji jedan mali problem: na šta rCat upućuje posle linije 31? Kao što je naglašeno ranije, referenca uvek mora biti alijas stvarnog objekta; ako ona upućuje na nul 1 objekat (kao ova sada), program je nevažeći.

^HAMMIMA^ Nikad nisu suvišne napomene da se program sa referencom na null objekat može kompajlirati, ali on je nevažeći i njegovo izvođenje je nepredvidivo.

Postoje tri rešenja za ovaj problem. Prvo je da se deklariše SimpleCat objekat u liniji 25 i da se vrati mačka iz TheFunction po vrednosti. Drugo je da se nastavi i deklariše SimpleCat na slobodnom skladištu u funkciji TheFunctionO, ali da TheFunctionO vrati pokazivac na tu memoriju. Onda pozivajuća funkcija može izbrisati pokazivac, kada je sa njim zavrseno.

Treće upotrebljivo rešenje, i pravo, je da se deklariše objekat u pozivajućoj funkciji, a onda da se preda funkciji TheFunctionO po referenci.

Pokazivač, pokazivač, ko ima pokazivač?

Kada Vaš program alocira memoriju na slobodnom skladištu, vraća se pokazivač Imperativ je da sačuvate pokazivač na tu memoriju, jer ako se on izgubi, ona se ne može izbrisati i postaje memorijska pukotina. Čim predajete ovaj blok memorije izmedu funkcija, neko će "posedovati" pokazivač. Obično će vrednost u bloku biti predata korišćenjem referenci, a funkcija koja je kreirala memoriju je ona koja je briše. Ovo je opšte pravilo, a ne izuzetak.

Ipak, opasno je da jedna funkcija kreira memoriju a druga da je oslobađa. Dvosmislenost o tome ko poseduje pokazivač može dovesti do jednog od sledeća dva problema: zaboravljanja da se obriše pokazivač, ili brisanja pokazivača dva puta. Oba mogu prouzrokovati ozbiljne probleme u Vašem programu. Sigurnije je da izgradite Vaše funkcije tako da one obrišu memoriju koju kreiraju.

Ako pišete funkciju koja treba da kreira memoriju, a onda da je vrati pozivajućoj funkciji, razmotrite menjanje Vašeg interfejsa. Neka pozivajuća funkcija alocira memoriju, a, onda, neka je preda Vašoj funkciji po referenci. Ovo premešta upravljanje memorijom iz Vašeg programa u funkciju koja je pripremljena da je izbriše.

^ **PAZin** Predajte parametre po vrednosti, kada morale.

Obavite vraćanje po vrednosti, kada morate.

Nemojte obavljati predavanje po referenci, ako clan na koji se upućuje može izaći izvan opsega.

Nemojte koristiti reference na null objekte.

Rezime

Danas ste naučili šta su to reference i kako se one porede sa pokazivačima. Videli ste da reference moraju biti inicijalizovane da upućuju na postojeći objekat i ne može se ostvariti ponovna dodela nad njima da bi upućivale na nešto drugo. Svaka akcija preduzeta nad referencom se u, stvari, preduzima nad ciljnim objektom reference. Dokaz ovoga je da uzimanje adrese reference vraća adresu mete.

Videli ste da predavanje objekata po referenci može biti efikasnije od predavanja po vrednosti. Predavanje po referenci takode dozvoljava pozvanoj funkciji da promeni vrednost u argumentima iz pozivajuće funkcije.

Videli ste da se argumenti za funkcije i vrednosti vraćene iz funkcija mogu predati po referenci i da se ovo može implementirati sa pokazivačima, ili referencama.

Videli ste kako da koristite const pokazivače i const reference za sigurno predavanje vrednosti između funkcija, dok istovremeno postižete i efikasnost predavanja po referenci.

fpitanja i odgovori

P Zašto imati reference ako pokazivači mogu uraditi sve što i reference?

O Reference su lakše za korišćenje i razumevanje. Indirekcija je sakrivena, i nema potrebe iznova dereferencirati promenljivu.

P Zašto imati pokazivače ako su reference lakše?

O Reference ne mogu biti null, i nad njima se ne može ostvariti ponovna dodela. Pokazivači nude veću fleksibilnost, ali su malo teži za korišćenje.

P Zašto uopšte ostvarivati vraćanje iz funkcije po vrednosti?

O Ako je objekat koji se vraća lokalan, morate vratiti po vrednosti, ili ćete vratiti referencu na nepostojeći objekat.

P Zbog opasnosti u vraćanju po referenci, zašto uvek ne ostvariti vraćanje po vrednosti?

O Postoji daleko veća efikasnost u vraćanju po referenci. Memorija se štedi a program se izvršava brže.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Kakva je razlika između reference i pokazivača?
2. Kada morate koristiti pokazivač, umesto reference?
3. Šta new vraća, ako nema dovoljno memorije za kreiranje Vašeg novog objekta?
4. Šta je konstantna referenca?
5. Kakva je razlika između predavanja po referenci i predavanja reference?

Vežbe

1. Napišite program koji deklariše int, referencu na int i pokazivač na int. Upotrebite pokazivač i referencu za manipulisanje vrednošću u int.

Napišite program koji deklariše konstantan pokazivač na konstantnu celobrojnu vrednost. Inicijalizujte pokazivač na celobrojnu promenljivu, varOne. Dodelite 6 promenljivoj varOne. Upotrebite pokazivač za dodeljivanje 7 promenljivoj varOne. Kreirajte drugu celobrojnu promenljivu, varTwo. Preusmerite pokazivač na varTwo.

3. Kompajlirajte program u Vežbi 2. Šta proizvodi greške? Šta proizvodi upozorenja?
4. Napišite program koji proizvodi izgubljeni pokazivač.
5. Popravite program iz Vežbe 4.
6. Napišite program koji proizvodi memorijsku pukotinu.
7. Popravite program iz Vežbe 6.

ISTERIVAC BAGOVA: Sta nije u redu sa ovim programom?

```
1    #inc1ude <iostream.h>
2
3    class CAT
4    {
5       public:
6          CAT(int age) { itsAge   age;
7          ~CAT(){}
8          int GetAge() const { return itsAge;}
9       private:
10         int itsAge;
11
12
13   CAT & MakeCat(int age);
14   int main()
15   {
16      int age = 7;
17      CAT Boots = MakeCat(age);
18      cout « "Boots is " « Boots.GetAqe()      years o1d\n";
19
20
21   CAT & MakeCat(int age)
22   {
23      CAT * pCat = new CAT(age);
24      return *pCat;
25
```

9. Popravite program iz Vežbe 8.

Dan 10

Napredne funkcije

U Danu 5, "Funkcije", naučili ste osnove rada sa funkcijama. Sada, kada znate kako rade pokazivači i reference, možete vise uraditi sa funkcijama. Danas učite

- kako da preklopite funkcije članice
- kako da preklopite operatore
- kako da napišete funkcije, da biste podržali klase sa dinamički alociranim promenljivama.

Preklopljene funkcije clanice

U Danu 5, naučili ste kako da implementirate funkcijski polimorfizam, ili preklapanje funkcija, pisanjem dve, ili vise funkcija, sa istim imenom, ali sa različitim parametrima. Funkcije članice klase se takode mogu preklopiti, na veoma sličan način.

Klasa Rectangle, demonstrirana u listingu 10.1, ima dve funkcije DrawShape(). Jedna, koja ne uzima parametre, crta Rectangle (engl. rectangle = pravougaonik), bazirajući se na tekućim vrednostima klase. Druga prihvata dve vrednosti, width i length, i crta pravougaonik, bazirajući se na tim vrednostima, ignorišući tekuće vrednosti klase.

Listing 10.1: Preklapanje funkcija članica.

```
//Listing 10.1 Preklapanje funkcija članica klasa
#include <iostream.h>

typedef unsigned short int USHORT;
```

nastavlja se

nastavak

Listing 10.1; Preklapanje funkcija ganka.

```
5:    enum BOOL { FALSE, TRUE};
6:
7:    // Deklaracija za klasu Rectangle
8:    class Rectangle
9:    {
10:   public:
11:      // konstruktori
12:      Rectangle(USHORT width, USHORT height);
13:      -RectangleOO
14:
15:      // preklopljena klasna funkcija DrawShape
16:      void DrawShapeO const;
17:      void DrawShape(USHORT aWidth, USHORT aHeight) const;
18:
19:   private:
20:      USHORT itsWidth;
21:      USHORT itsHeight;
22:   };
23:
24:   //Implementacija konstruktora
25:  Rectangle::Rectangle(USHORT width, USHORT height)
26:   {
27:      itsWidth = width;
28:      itsHeight = height;
29:   }
30:
31:
32:   // Preklopljena DrawShape - ne prihvata vrednosti
33:   // Crta bazirajući se na tekućim vrednostima članova klase
34:  void Rectangle::DrawShapeO const
35:   {
36:       DrawShape( itsWidth, itsHeight);
37:   }
38:
39:
40:   // preklopljena DrawShape - prihvata dve vrednosti
41:   // crta oblik bazirajuči se na parametrima
42:  void Rectangle::DrawShape(USHORT width, USHORT height) const
43:   {
44:      for (USHORT i = 0; Kheight; i++)
45:      {
46:         for (USHORT j = 0; j< width; j++)
47:         {
48:            cout « "*";
49:         )
50:         cout « "\n";
51:      }
52:   }
```


```
// Demonstracioni program koji demonstrira preklopljene funkcije
int mainQ
{
   // inicijalizuje pravougaonik na 30,5
   RectangletheRect(30,5);
   cout « "DrawShapeO: \n";
   theRect.DrawShapeO;
   cout « "\nDrawShape(40,2): \n";
   theRect.DrawShape(40,2);
 return 0;
```

v^MAPOMEItty Ovaj listing predaje vrednosti width i height nekim funkcijama. Trebalo bi da uodte da u nekim slučajevima prvo se predaje wi dth, a u drugim slučajevima hei ght.

```
DrawShape(0,0,TRUE)...
******************************
******************************
******************************
******************************
******************************

DrawShape(40,2)...
**********************************  r**********************
**********************************  r**********************
```

ђшмЈјп^ Listing 10.1 predstavlja pojednostavljenu verziju projekta iz pregleda sadržaja iz Nedelje 1. Test nelegalnih vrednosti je izvaden, da bi se uštedeo prostor, kao i neke funkcije pristupa. Glavni program je pojednostavljen i sada je jednostavni kontinuelni, a ne program sa menijem.

Ipak, važan kod je u linijama 16 i 17, gde je preklopljena funkcija DrawShapeO. Implementacija za ove preklopljene klasne metode je u linijama 32-52. Uočite da verzija funkcije DrawShapeO, koja ne prihvata parametre, jednostavno, poziva verziju koja prihvata dva parametra, predajući tekuće promenljive članice. Uložite napor da izbegnete dupliranje koda u dve funkcije. Inače, održavanje njihove sinhronizacije kada se naprave promene u jednoj, ili drugoj će biti teško i podložno greškama.

Kontinuelni program, u linijama 54-64, kreira objekat pravougaonika, a, onda, poziva DrawShapeO, ne predajući parametre, i predajući dve unsigned short celobrojne vrednosti.

Kompajler odlučuje koji metod da pozove, bazirajući se na broju i tipu navedenih parametara. Neko može zamisliti treću preklopljenu funkciju, nazvanu DrawShapeO, koja prihvata jednu dimenziju i enumeraciju za to da li je u pitanju širina ili visina, u zavisnosti od korisnikovog izbora.

Korišćenje podrazumevanih vrednosti

Kao što neklasne funkcije mogu imati jednu, ili vise podrazumevanih vrednosti može i svaka funkcija članica klase. Ista pravila se odnose na deklarisanje podrazumevanih vrednosti, kao što je ilustrovano u listingu 10.2.

Listing 10.2: Korišćenje podrazumevanih vrednosti

```
1   //Listing 10.2 Podrazumevane vrednosti u funkcijama članicama
2   #include <iostream.h>
3
4   typedef unsigned short int USHORT;
5   enum BOOL { FALSE, TRUE};
6
7   // Deklaracija klase Rectangle
8   class Rectangle
9   {
10  public:
11     // konstruktori
12     Rectangle(USH0RT width, USHORT height);
13     -RectangleOO
14     void DrawShape(USHORT aWidth, USHORT aHeight, B00L UseCurrentVals = FALSE)
       ^·const;
15:
16  private:
17     USHORT itsWidth;
18     USHORT itsHeight;
19  };
20
21  //Implementacija konstruktora
22  Rectangle: .-Rectangle(USHORT width, USHORT height):
23  itsWidth(width),        // inicijalizacije
24  i tsHeight(height)
25  (}                      // prazno telo
26
27
28  // podrazumevane vrednosti koje se koriste za treći parametar
29  void Rectangle::DrawShape(
30     USHORT width,
31     USHORT height,
32     B00L UseCurrentValue
33     ) const
34  (
35     int printWidth;
36     int  printHeight;
37
38     if (UseCurrentValue == TRUE)
39
40        printWidth = itsWidth;        // koriste se tekuće vrednosti klase
41        printHeight = itsHeight;
```



```
   else
   {
      printWidth = width;          // koriste se vrednosti parametara
      printHeight = height;
   }


   for (int i = 0; i<printHeight; i++)
   {
      for (int j = 0; j< printWidth; j++)
      {
         COUt « "*";
      }
      cout « "\n";



// Kontinualni program koji demonstrira preklopljene funkcije
int main()
{
   // inicijalizuje pravougaonik na 10,20
   Rectangle theRect(30,5);
   cout « "DrawShape(O.O.TRUE)...\n";
   theRect.DrawShape(0,0,TRUE);
   cout «"DrawShape(40,2).. An";
   theRect.DrawShape(40,2);
 return 0;
```

```
jBEEfffcfe  DrawShape(0,0,TRUE)...
            **************************
            ******************************
            ******************************
            ******************************
            ******************************
            DrawShape(40,2)...
            ************************************************************
            ************************************************************
```

ШШJj5> Listing 10.2 zamenjuje preklopljenu funkciju `DrawShapeO` jednom funkcijom sa podrazumevanim parametrima. Funkcija je deklarisana u liniji 14 da prihvati tri parametra. Prva dva, `aWidth` i `aHeight`, su tipa `USHORT`, a treći, `UseCurrentVals`, je tipa `B00L` (istinito, ili neistinito), čija je podrazumevana vrednost `FALSE`.

NAPOMENA^ Boolean vrednosti su one koje se svode na TRUE, ili FALSE. C+ + smatra da je 0 neistina, a sve druge vrednosti da su istina.

Implementacija **za** ovu, u izvesnoj men, nezgodnu funkciju počinje u liniji 29. Treći parametar, **UseCurrentVal ue,** se proračunava. Ako je on **TRUE,** promenljive članice **itsWidth** i **itsHeight** se koriste **za** postavljanje lokalnih promenljivih **printWidth f printHeight.**

Ako je **UseCurrentVal ue** jednako **FALSE,** bilo zato što ima podrazumevanu vrednost **FALSE,** ili **zato što** ju je na tu vrednost postavio korisnik, prva **dva** parametra se koriste **za** postavljanje **printWidth** i **printHeight.**

Uočite da, **ako** je **UseCurrentVal ue** jednako **TRUE,** vrednosti druga **dva** parametra se potpuno ignorisu.

Biranje između podrazumevanih vrednosti i preklopljenih funkcija

Listinzi 10.1 i 10.2 proizvode isti efekat, ali preklopljene funkcije u listingu 10.1 su lakše za razumevanje i korišćenje. Takode, ako je potrebna treća varijacija - možda korisnik želi da obezbedi ili širinu, ili visinu, ali ne i jedno i drugo, lako je proširiti preklopljene funkcije. Ipak, podrazumevana vrednost će brzo postati neupotrebljivo kompleksna kad se budu dodavale nove varijacije.

Kako da odlučite da li da koristite preklapanje funkcija, ili podrazumevane vrednosti? Evo praktičnog pravila:

Preklapanje funkcija koristite

- kada ne postoji razumna podrazumevana vrednost
- kada su Vam potrebni različiti algoritmi
- kada je potrebno da podržite različite varijante tipova u Vašoj listi parametara.

Podrazumevani konstruktor

Kao što je rečeno u Danu 6, "Osnovne klase", ako ne deklarišete eksplicitno konstruktor za Vašu klasu, kreira se podrazumevani konstruktor, koji ne prihvata parametre i ne radi ništa. Vi imate slobodu da napravite sopstveni podrazumevani konstruktor, koji ne prihvata argumente, ali "postavlja" Vaš objekat prema zahtevu.

Konstruktor, obezbeden za Vasm se naziva "podrazumevani" konstruktor. Po konvenciji, to je svaki konstruktor koji ne prihvata parametre. Ovo može biti malo zbunjujuće, ali obično je jasno iz konteksta na koji se misli.

Obratite pažnju da, ako napravite bilo kakav konstruktor, kompajler ne pravi podrazumevani konstruktor. Zato, ako želite konstruktor koji ne prihvata parametre, a kreirali ste neke druge konstruktore, morate sami napraviti podrazumevani konstruktor.

Preklapajući konstruktori

Namena konstmktora je da ustanovi objekat; na primer, namena konstruktora klase Rectangle je da napravi pravougaonik. Pre izvršenja konstruktora, nema pravougaonika, samo područje memorije. Kada konstruktor završi, postoji kompletan, spreman za korišćenje objekat pravougaonika.

Na primer, možda ćete imati objekat rectangle sa dva konstruktora. Prvi prihvata dužinu i širinu i pravi pravougaonik te veličine. Drugi ne prihvata nikakve vrednosti i pravi pravougaonik podrazumevane veličine. Listing **10.3** implementira ovu ideju

Listing 10.3: Preklapanje konstruktora.

```
1:    // Listing 10.3
2:     // Preklapajući konstruktori
3:
4:    #include <iostream.h>
5:
6:    class Rectangle
7:    {
8:    public:
9:         Rectangle();
10:        Rectangle(int width, int length);
11:        -Rectangle() {}
12:        int GetWidth() const { return itsWidth; }
13:        int GetLength() const { return itsLength; }
14:   private:
15:        int itsWidth;
16:        int itsLength;
17:   };
18:
19:   Rectangle::Rectangle()
20:   {
21:        itsWidth = 5;
22:        itsLength = 10;
23:   }
24:
25:   Rectangle::Rectangle (int width, int length)
26:   {
27:        itsWidth = width;
28:        itsLength = length;
29:   }
30:
31:   int main()
32:   {
33:        Rectangle Recti;
34:        cout « "Recti width: " « Recti.GetWidthQ « endl;
35:        cout « "Recti length: " « Recti.GetLengthQ « endl;
36:                                                    nastavlja se
```

Listing 10.3: Preklapanje konstruktora. *nastavak*

```
37          int aWidth, aLength;
38          cout « "Enter a width: ";
39          •cin » aWidth;
40          cout « "\nEnter a length: ";
41          cin » aLength;
42
43          Rectangle Rect2(aWidth, aLength);
44          cout « "\nRect2 width: " « Rect2.GetWidth() « endl;
45          cout « "Rect2 length: • « Rect2.GetLength() « endl;
46      return 0;
47
```

```
Recti width: 5
Recti length: 10
Enter a width: 20

Enter a length: 50

Rect2 width: 20
Rect2 length: 50.
```

Klasa Rectangle je deklarisana u linijama 6-17. Deklarisana su dva konstruktora: "podrazumevani konstruktor" u liniji 9 **I** konstruktor koji prihvata dve celobrojne promenljive.

U liniji 33 kreira se pravougaonik, korišćenjem podrazumevanog konstruktora, i štampaju se njegove vrednosti u linijama 34 i 35. U linijama 37-41 od korisnika se traže širina i dužina i poziva se konstruktor, koji prihvata dva parametra u liniji 43. Konačno, širina i dužina za ovaj pravougaonik se štampaju u linijama 44 i 45.

Kao što radi sa svakom preklopljenom funkcijom, kompajler bira ispravni konstruktor, bazirajući se na broju i tipu parametara.

Inicijalizacija objekata

Do sada ste promenljive članice objekata postavljali u telu konstruktora. Konstruktori se pozivaju u dve etape: inicijalizaciona etapa i telo.

Većina promenljivih se može postaviti u bilo kojoj etapi, bilo inicijalizacijom u inicijahzacijonoj etapi, ili dodeljivanjem u telu konstruktora. Čistije je i, često, efikasnije micijahzovati promenljive članice u inicijalizacionoj etapi. Sledeći primer prikazuje kako micijahzovati promenljive članice:

```
CAT0:          // parametri i ime konstruktora
itsAge(5),     // inicijalizaciona lista
itsWeight(8)
^ }               // telo konstruktora
```


Posle zatvarajućih zagrada u konstruktorovoj listi parametara, navedite dve tačke. Onda navedite ime promenljive članice i par zagrada. Unutar zagrada, navedite izraz koji će se koristiti za inicijalizaciju te promenljive članice. Ako ima vise od jedne inicijalizacije, rastavite ih zarezima. Listing 10.4 prikazuje definiciju konstruktora iz listinga 10.3, sa inicijalizacijom promenljivih članica, umesto sa dodelom.

Listing 10.4: Isečak koda koji pokazuje inicijalizaciju promnljivih clanica.

```
Rectangle::Rectangle():
    itsWidth(5),
    itsLength(IO)



Rectangle::Rectangle (int width, int length)
    itsWidth(width),
    itsLength(length)



        Nema izlaza.
```

Postoje neke promenljive koje se moraju micijahzovati i nad kojima se ne može obaviti dodeljivanje: reference i konstante. Uobičajeno je da druge dodele, ili akcioni iskazi budu u telu konstruktora; ipak, najbolje je koristiti inicijalizaciju, koliko je to moguće.

Konstruktor kopije

Pored obezbedivanja podrazumevanih konstruktora i destruktora, kompajler obezbeduje podrazumevani konstruktor kopije, koji se poziva svaki put kada se napravi kopija objekta.

Kada prosledujete objekat po vrednosti, bilo da ga predajete funkciji, ili kao povratnu vrednost iz funkcije, pravi se privremena kopija tog objekta. Ako je objekat korisnički definisan objekat, poziva se konstruktor kopije njegove klase, kao što ste to juče videli u listingu 9.6.

Svi konstruktori kopije prihvataju jedan parametar, referencu na objekat iste klase. Dobra ideja je učiniti ga konstantnom referencom, zato što konstruktor neće morati da promeni predati objekat. Na primer:

```
CAT(const CAT & theCat);
```

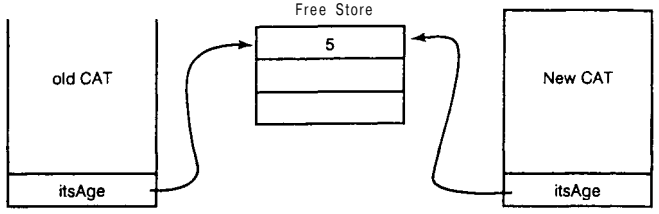
Ovde CAT konstruktor prihvata konstantnu referencu na postojeći CAT objekat. Cilj konstruktora kopije je da napravi kopiju od theCat.

Podrazumevani konstruktor kopije, jednostavno, kopira svaku promenljivu članicu iz objekta, koji je predat kao parametar u promenljive članice novog objekta. Ovo se

naziva člansko-odnosna (ili plitka) kopija i, iako je ovo dobro za većinu promenljivih članica, ona pada veoma brzo za promenljive članice, koje su pokazivači na objekte sa slobodnog skladišta.

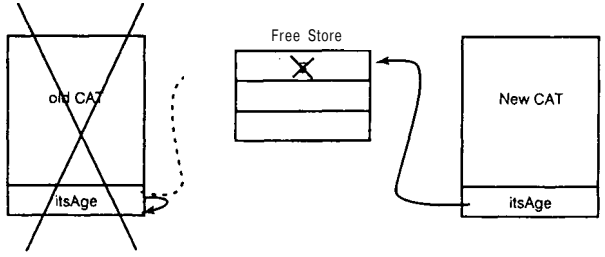
Plitka, ili člansko-odnosna kopija kopira tačne vrednosti promenljivih, koje su članice objekta, u drugi objekat. Pokazivači u oba objekta završavaju, pokazujući na istu memoriju. *Duboka kopija* kopira vrednosti alocirane na gomili u novo alociranu memoriju.

Kada bi klasa **CAT** imala promenljivu članicu, **itsAge,** koja pokazuje na celobrojnu vrednost sa slobodnog skladišta, podrazumevani konstruktor kopije bi kopirao predatu promenljivu članicu **i tsAge** klase **CAT** u novu promenljivu članicu **i tsAge** klase **CAT.** Dva objekta bi tada pokazivala na istu memoriju, kao što je ilustrovano na slici 10.1.



Slika 10.1. Korišćenje podrazumevanog konstruktora kopije

Ovo će dovesti do katastrofe kada bilo koji **CAT** izade iz opsega. Kao što je pomenuto u Danu 8, "Pokazivači", posao destruktora je da očisti ovu memoriju. Ako destruktor originalnog **CAT** oslobodi ovu memoriju, a novi **CAT** još uvek pokazuje na nju, kreiran je izgubljeni pokazivač, a program je u smrtnoj opasnosti. Slika 10.2 ilustruje ovaj problem.



Slika 10.2. Kreiranje izgubljenog pokazivača

Rešenje je da kreirate sopstveni konstruktor kopije i da alocirate memoriju prema potrebi. Pošto je memorija alocirana, stare vrednosti se mogu kopirati u novu memoriju. Ovo se naziva duboka kopija. Listing 10.5 ilustruje kako se ovo radi.

Listing 10.5: Konstruktori kopije.

```
// Listing 10.5
// Konstruktori kopije

#include<iostream.h>

class CAT
{
    publ ic:
        CAT();                          //podrazumevani konstruktor
        CAT (const CAT &);      // konstruktor kopije
        -CAT();                          // destruktor
        int GetAge() const { return *itsAge; }
        int GetWeight() const { return *itsWeight; }
        void SetAge(int age) { *itsAge = age; }

    private:
        int *itsAge;
        int  *itsWeight;


CAT::CAT()

    itsAge = new int;
    itsWeight = new int;
    *itsAge · 5;
    *itsWeight = 9;


CAT::CAT(const CAT & rhs)

    itsAge = new int;
    itsWeight = new int;
    *itsAge = rhs.GetAgeQ ;
    ·itsWeight = rhs.GetWeightQ;


CAT::-CAT()

    delete itsAge;
    itsAge = 0;
    delete  itsWeight;
    itsWeight = 0;


int main()
{
    CAT frisky;
    cout « "friskyjeva starost: " « frisky.GetAge() « endl;
```

nastavlja se

Listing 10.5: Konstruktori kopije. *nastavak*

```
49:       cout « "Postavljanje friskyja na 6...V;
50:       frisky.SetAge(6);
51:       cout « "Kreiranje boots-a iz frisky-ja\nM;
52:       CAT boots(frisky);
53:       cout « "friskyjeva starost: " «      frisky.GetAge() « endl;
54:       cout « "bootsova starost: " « boots.GetAge() « endl;
55:       cout « "setting frisky to 7...\n";
56:       frisky.SetAge(7);
57:       cout « "friskyjeva starost: " «      frisky.GetAgeQ « endl;
58:       cout « "bootsova starost: " « boots.GetAgeQ « endl;
59:    return 0;
60: }
```

```
friskyjeva starost: 5
Postavlja friskyja na  6...
Kreiranje boots-a za friskyja
friskyjeva starost: 6
bootsova starost:  6
postavlja friskyja na 7...
friskyjeva starost: 7
bootsova starost: 6
```

U linijama 6-19 deklariše se klasa CAT. Uočite da je u liniji 9 deklansan podrazumevani konstruktor, a u liniji 10 konstruktor kopije.

U linijama 17 i 18 deklarišu se dve promenljive članice, svaka kao pokazivač na celobrojnu vrednost. Obično ima malo razloga da klasa čuva int promenljive članice kao pokazivače, ali ovo je uradeno da bi se ilustrovalo kako upravljati promenljivim članicama na slobodnom skladištu.

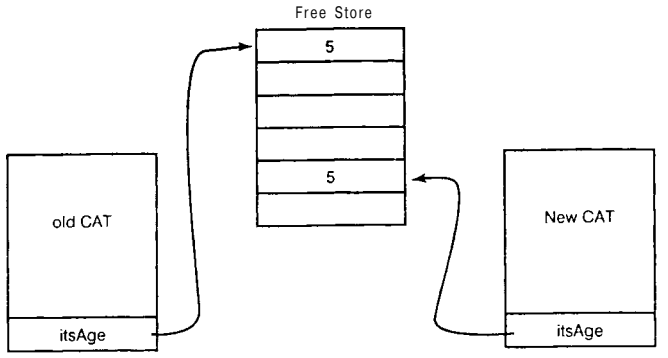
Konstruktor kopije počinje u liniji 29. Uočite da je parametar rhs. Uobičajeno je kao parametar konstruktora kopije koristiti rhs, što znači desnu stranu (engl. right-hand side). Kada pogledate dodele u linijama 33 i 34, videćete da je objekat predat kao parametar na desnoj strani znaka jednakosti.

U linijama 31 i 32 alocira se memorija na slobodnom skladištu. Zatim, u linijama 33 i 34 vrednosti na novoj memorijskoj lokaciji se dodeljuju vrednosti iz postojećeg CAT.

Parametar rhs je CAT koji se predaje konstruktoru kopije kao konstantna referenca. Funkcija članica rhs.GetAge() vraća vrednost čuvanu u memoriji, na koju pokazuje itsAge promenljiva članica od rhs. Kao CAT objekat, r h s ima sve promenljive članice kao i bilo koja CAT.

Kada se pozove konstruktor kopije za kreiranje novog CAT, postojeći CAT se predaje kao parametar. Novi CAT može direktno upućivati na sopstvene promenljive članice; ipak, on mora pristupiti promenljivim članicama od r h s , korišćenjem javnih metoda pristupa.

Slika 10.3 prikazuje dijagram o tome šta se ovde dešava. Vrednosti na koje pokazuje postojeći CAT se kopiraju u memoriju. koja je alocirana za novi CAT.



Slika 10.3. Ilustrovana duboka kopija

U liniji 47 kreira se CAT, nazvan fri sky. Štampa se Fri skyjeva starost, a onda se ona postavlja na 6 u liniji 50. U liniji 52 kreira se novi CAT nazvan boots, korišćenjem konstruktora kopije i predavanjem frisky-ja. Da je frisky predat kao parametar funkciji, ovaj isti poziv konstruktora kopije napravio bi kompajler.

U linijama 53 i 54 štampaju se starosti oba CAT. Stvarno, boots ima frisky-jevu starost, 6, a ne podrazumevanu starost 5. U liniji 56 frisky-jeva starost se postavlja na 7, a onda se starosti ponovo štampaju. Ovog puta frisky-jeva starost je 7, ali boots-ova je još uvek 6, što demonstrira da se one čuvaju u odvojenim područjima memorije.

Kada CAT-ovi izadu iz opsega, njihovi destruktori se automatski pozivaju. Implementacija destruktora klase CAT je prikazana u linijama 37-43; delete se poziva za oba pokazivača, itsAge i itsWeight, što vraća alociranu memoriju slobodnom skladištu. Takođe, zbog bezbednosti, pokazivačima se dodeljuje vrednost NULL.

Preklapanje operatora

C++ ima nekoliko ugrađenih tipova, uključujući i nt, real, char i tako dalje. Svaki od njih ima nekoliko ugradenih operatora, kao što su sabiranje (+) i množenje (*). C++ Vam omogućava da dodate ove operatore Vašim klasama.

Da bismo potpuno proučili preklapanje operatora, listing 10.6 kreira novu klasu, Counter. Counter objekat će se koristiti za brojanje (iznenađenje!) u petljama i drugim aplikacijama, gde broj mora biti inkrementiran, dekrementiran, ili na neki drugi način održavan.

Listing 10.6. Klasa Counter.

```
1:    // Listing 10.6
2:    // Klasa Counter
3:
4:    typedef unsigned short  USHORT;
5:    linclude <iostream.h>
6:
7:    class Counter
8:    {
9:       public:
10:         Counter();
11:         ~Counter(){}
12:         USHORT GetItsVal()const { return itsVal; }
13:         void SetItsVal(USHORT x) (itsVal = x; }
14:
15:      private:
16:         USHORT itsVal;
17:
18:   };
19:
20:   Counter;:Counter():
21:    itsVal(0)
22:   {};
23:
24:    int main()
25:   {
26:      Counter  i;
27:      cout « "The value of i is " « i.GetItsVal() « endl;
28:     return 0;
29:   }
```

IZLAZ

```
The value of i is 0
```

Kao što se vidi, ovo je prilično beskorisna klasa. Definisana je u linijama 7-18. Njena jedina promenljiva članica je USHORT. Podrazumevani konstruktor, koji je deklarisan u liniji 10 i čija je implementacija u liniji 20, inicijalizuje jednu promenljivu članicu, itsVal, sa nula.

Za razliku od USHORT, Counter objekat se ne može inkrementirati, dekrementirati, sabrati, dodeliti, ili se na neki drugi način njime manipulisati. On u velikoj meri otežava štampanje njegove vrednosti.

Pisanje funkcije za inkrementiranje

Preklapanje operatora vrača funkcionalnost, koja je bila oduzeta od ove klase. Na primer, postoje dva načina da se doda sposobnost za inkrementiranje Counter objekta. Prvi je da se napiše metoda za inkrementiranje, kao što je prikazano u listingu 10.7.

Listing 10.7. Dodavanje inkrement operatora.

```
// Listing 10.7
// Klasa Counter

typedef unsigned short  USHORT;
#include <iostream.h>

class Counter
{
   public:
      Counter();
      -Counter(){}
      USHORT GetItsVal()const { return itsVal; }
      void SetItsVal(USHORT x) {itsVal = x; }
      void IncrementQ { ++itsVal; }

   private:
      USHORT itsVal;

};

Counter::Counter():
itsVal(0)
{};

int  main()
{
   Counter  i;
   cout « "The value of i is " « i.GetItsVal () « endl;
   i.Increment();
   cout « "The value of i is " « i.GetItsVal() « endl;
 return 0;
}
```

```
The value of i is 0
The value of i is 1
```

Listing 10.7 dodaje funkciju Increment, koja je definisana u liniji 14. Iako ovo funkcioniše nezgodno je za korišćenje. Ovom programu je veoma potreban operator ++ i, naravno, to se može uraditi.

Preklapanje prefiksnog operatora

Prefiksni operatori se mogu preklopiti deklarisanjem funkcija sa oblikom:

```
returnType Operator op (parameters)
```

Operator ++ se može preklopiti sa sledećom sintaksom:

```
void operator++ ()
```

Listing 10.8 prikazuje ovu alternative

```
1    // Listing 10.8
2    // Klasa Counter
3
4    typedef unsigned short USHORT;
5    #include <iostream.h>
6
7    class Counter
8    {
9       public:
10         Counter();
11         -CounterQU
12         USHORT GetItsVal Oconst { return itsVal;
13         void SetItsVal(USHORT x) (itsVal = x; }
14         void Increment() ( ++itsVal; }
15         void operator++ () ( ++itsVal; }
16
17      private:
18         USHORT itsVal;
19
20   1;
21
22   Counter::Counter():
23   itsVal(0)
24   (};
25
26   int main()
27   (
28      Counter i;
29      cout « "The value of i is " « i.GetItsVal() « endl
30      i.Increment();
31      cout « "The value of i is " « i.GetItsVal() « endl
32      ++i;
33      cout « "The value of i is " « i.GetItsVal () « endl
34    return 0;
35   }
```

```
The value of i is 0
The value of i is 1
The value of i is 2
```

*(ЛШjA^* U liniji 15 preklopljen je `operator++`, a upotrebljen je u liniji 32. U ovom trenutku ćete možda razmotriti stavljanje dodatnih mogućnosti, za koje je prvenstveno `Counter` i bila kreirana, kao što je detektovanje slučaja kada `Counter` prekorači svoju maksimalnu veličinu.

Ipak, postoji značajan defekt u načinu na koji je inkrement operator napisan. Ako želite da stavite `Counter` na desnu stranu dodele, to neće uspeti. Na primer:

```
Counter a = ++i;
```


Ovaj kod namerava da kreira novi `Counter`, a onda da mu dodeli vrednost u i posle inkrementiranja i. Ugradeni konstruktor kopije će rešiti ovu dodelu, ali tekući inkrement operator ne vraća `Counter` objekat. On vraća `void`. Ne možete dodeliti `void` objekat `Counter` objektu (ne možete napraviti nešto ni od Čega!)

Vraćanje tipova u preklopljenim operatorskim funkcijama

Jasno, ono što želite je da vratite `Counter` objekat tako da on može biti dodeljen drugom `Counter` objektu. Koji objekat bi trebalo da bude vraćen? Jedan pristup bi bio da kreirate privremeni objekat i njega da vratite. Listing 10.9 ilustruje ovaj pristup.

Listing 10.9: Vraćanje privremenog objekta.

```
1:    // Listing 10.9
2:    // operator++ vraća privremeni objekat
3:
4:    typedef unsigned short  USHORT;
5:    #include <iostream.h>
6:
7:    class Counter
8:    (
9:       public:
10:         Counter();
11:         -Counter(){}
12:         USHORT GetItsVal()const ( return itsVal; }
13:         void SetItsVal(USHORT x) (itsVal = x; }
14:         void IncrementQ ( ++itsVal; }
15:         Counter operator++ ();
16:
17:      private:
18:         USHORT itsVal;
19:
20:   };
21:
22:   Counter::Counter():
23:    itsVal(0)
24:   (};
25:
26:   Counter Counter::operator++()
27:   {
28:       ++itsVal;
29:       Counter temp;
30:        temp.SetItsVal(itsVal);
31:       return temp;
32:   }
33:
34:    int main()
35:   (
36:      Counter i;
```

nastavlja se

Listing 10.9: Vraćanje privremenog objekta. *nastavak*

```
37:       cout « "The value of i is " « i.GetItsVal() «  endl;
38:       i.Increment();
39:       cout « "The value of i is " « i.GetItsVal() «  endl;
40:       ++i;
41:       cout « "The value of i is " « i.GetItsVal() «  endl;
42:       Counter a = ++i;
43:       cout « "The value of a: " « a.GetItsVal();
44:       cout « " and i: " « i.GetItsVal() « endl;
45:     return 0;
46:   }
```

wAifr
```
The value of i is 0
The value of i is 1
The value of i is 2
The value of a: 3 and i: 3
```

*kxmtijjn^** U ovoj verziji operator++ je deklarisan u liniji 15 da vraća Counter objekat. U liniji 29 kreira se privremena promenljiva, temp, a njena vrednost se postavlja da odgovara vrednosti u tekućem objektu. Ta privremena promenljiva se vraća i odmah dodeljuje objektu a u liniji 42.

Vraćanje bezimenih privremenih

Stvarno nema potrebe imenovati privremeni objekat kreiran u liniji 29. Kada bi Counter imao konstruktor koji prihvata vrednost, mogli biste, jednostavno, vratiti rezultat tog konstruktora kao povratnu vrednost inkrement operatora. Listing 10.10 ilustruje ovu ideju.

Listing 10.10. Vraćanje bezimenog privremenog objekta.

```
1:    // Listing 10.10
2:    // operator++ vraća bezimeni privremeni objekat
3:
4:    typedef unsigned short  USHORT;
5:    #include <iostream.h>
6:
7:    class Counter
8:    {
9:       public:
10:         Counter();
11:         Counter(USH0RT  val);
12:         -CounterOU
13:         USHORT GetItsVal()const { return itsVal; }
14:         void SetItsVal(USHORT x) (itsVal = x; }
15:         void Increment() ( ++itsVal; }
16:         Counter operator++ ();
17:
18:      private:
```



```
      USHORT itsVal;

};

Counter::Counter():
itsVal(0)
{}

Counter::Counter(USH0RTval):
itsVal(val)
(I

Counter Counter::operator++()
(
   ++itsVal;
   return Counter (itsVal);


int mainQ
(
   Counter i;
   cout « "The value of i is " « i.GetItsVal() « endl
   i.Increment();
   cout « "The value of i is " « i.GetItsVal() « endl
   ++i;
   cout « "The value of i is " « i.GetItsVal() « endl
   Counter a = ++i;
   cout « "The value of a: " « a.GetItsVal();
   cout « " and i: " « i.GetItsVal() « endl;
return 0;
```

```
The value of i is 0
The value of i is 1
The value of i is 2
The value of a: 3 and i:
```

U liniji 11 deklariše se novi konstruktor, koji prihvata USHORT. Implementacija se nalazi u linijama 27-29. Ona inicijalizuje itsVal predanom vrednošću.

Implementacija od operator++ je sada pojednostavljena. U liniji 33 itsVal se inkrementira. Onda se u liniji 34 kreira privremeni Counter objekat, inicijalizovan na vrednost u itsVal, a onda vraćen, kao rezultat od operator++.

Ovo je elegantnije, ali postavlja se pitanje: "Zašto, uopšte, kreirati privremeni objekat?" Zapamtite da svaki privremeni objekat mora biti konstruisan, a kasnije i uništen - potencijalno skupa operacija. Objekat i ve§ postoji i već ima pravu vrednost, pa zašto ne vratiti njega? Ovaj problem ćemo rešiti korišćenjem pokazivača this.

Korišćenje pokazivača this

Pokazivač this, kao što je juče objašnjeno, bio je predat funkciji članici operator++ kao i svim funkcijama članicama. Pokazivač this pokazuje na i i, ako se dereferencira, vratiće objekat i, koji već ima pravu vrednost u svojoj promenljivoj članici itsVal. Listing 10.11 ilustruje vraćanje dereferenciranog pokazivača this i zaobilaženje kreiranja nepotrebnog privremenog objekta.

Listing 10.11. Vraćanje pokazivača this.

```
1   // Listing 10.11
2   // Vraćanje dereferenciranog pokazivača this
3
4   typedef unsigned short  USHORT;
5   #include <iostream.h>
6
7   class Counter
8   {
9      public:
10        CounterQ;
11        -Counter(){}
12        USHORT GetItsVal Oconst { return itsVal; }
13        void SetItsVal(USHORT x) (itsVal = x; }
14        void IncrementO { ++itsVal; }
15        const Counters operator++ ();
16
17     private:
18        USHORT itsVal;
19
20  };
21
22  Counter::Counter():
23  itsVal(0)
24  (};
25
26  const Counter& Counter::operator++()
27  (
28     ++itsVal;
29     return  *this;
30
31
32  int  main()
33  (
34     Counter  i;
35     cout « "The value of i is " « i.GetItsVal() « endl;
36     i. IncrementO;
37     cout « "The value of i is " « i.GetItsVal() « endl;
38     ++i;
39     cout « "The value of i is " « i.GetItsVal () « endl;
40     Counter a = ++i;
```



```
41     cout « "The value of a:  " « a.GetItsVal();
42     cout « " and i:  " « i.GetItsVal() « endl;
43  return  0;
44  }
```

```
The value of i is 0
The value of i is 1
The value of i is 2
The value of a: 3 and i: 3
```

IVM1fi:^ Implementacija od operator-н- u linijama 26-30 je promenjena, tako da sada dereferencira pokazivač this i vraća tekući objekat. Obezbeduje da se Counter objekat dodeli objektu a. Kao što je već objašnjeno, kada bi Counter objekat alocirao memoriju, bilo bi važno da se redefiniše konstruktor kopije. U ovom slučaju, podrazumevani konstruktor kopije radi dobro.

Uočite da je vraćena vrednost Counter referenca, čime se izbegava kreiranje dodatnog privremenog objekta. To je const referenca, jer vrednost ne bi trebalo da bude promenjena u funkciji koja koristi ovaj Counter.

Preklapanje postfiksnog operatora

Do sada ste preklapali prefiksni operator. Sta ako želite da preklopite postfiksni inkrement operator? Ovde kompajler ima problem: kako razlikovati prefiks i postfiks? Po konvenciji, celobrojna promenljiva se obezbeduje kao parametar deklaraciji operatora. Vrednost parametra se ignoriše; ona je samo signal da se radi o postfiksnom operatoru.

Razlika između prefiksa i postfiksa

Pre nego što možemo upotrebiti postfiksni operator, moramo razumeti po čemu se on razlikuje od prefiksnog operatora. Pregledali smo ovo detaljno u Danu 4, "Izrazi i iskazi" (pogledajte listing 4.3).

Podsećamo Vas: prefiks kaže: "Inkrementiraj, a onda vrati", dok postfiks kaže: "Vrati, a onda inkrementiraj".

Dok prefiksni operator može jednostavno inkrementirati vrednost, a onda vratiti sam objekat, postfiksni mora vratiti vrednost koja je postojala *pre* nego što je inkrementirana. Da bismo ovo uradili, moramo kreirati privremeni objekat koji će čuvati originalnu vrednost; onda će se inkrementirati vrednost originalnog objekta, a onda vratiti privremeni.

Prodimo to ponovo. Razmotrite sledeću liniju koda:

```
a = x++;
```

Ako je x bilo 5, posle ovog iskaza a je 5, ali x je 6. Vratili smo vrednost u x i dodelili je a, a onda smo uvećali vrednost od x. Ako je x objekat, njegov postfiksni inkrement

operator mora smestiti originalnu vrednost (5) u privremeni objekat, inkrementirati vrednost objekta x na 6, a onda vratiti taj privremeni objekat, da bi se njegova vrednost dodelila a.

Uočite da privremeni objekat moramo vratiti po vrednosti, a ne po referenci, jer ce privremeni objekat izaći iz opsega odmah po povratku iz funkcije.

Listing 10.12: Prefiksni i postfiksni operatori.

```
1:    // Listing 10.12
2:    // Vraćanje dereferenciranog pokazivača this
3:
4:    typedef unsigned short  USHORT;
5:    #include <iostream.h>
6:
7:    class Counter
8:    {
9:    public:
10       Counter();
11       -Counter(){}
12       USHORT GetItsVal(Jconst { return itsVal; }
13       void SetItsVal(USHORT x) (itsVal - x; }
14       const Counters operator++ ();       // prefix
15       const Counter operator** (int); // postfix
16
17    private:
18       USHORT itsVal;
19    };
20
21    Counter::Counter():
22    itsVal(0)
23    {}
24
25    const Counters Counter::operator++()
26    (
27       ++itsVal;
28       return *this;
29
30
31    const Counter Counter::operator++(int)
32    (
33       Counter temp(*this);
34       ++itsVal;
35       return temp;
36
37
38    int main()
39    (
40       Counter  i;
41       cout « "The value of i is " « i.GetItsVal() « endl;
```



```
42       i++;
43       cout  « "The value of i is " « i.GetItsVal () «  endl;
44       ++i;
45       cout  « "The value of i is " « i.GetItsVal () «  endl;
46       Counter a = ++i;
47       cout  « "The value of a:  " « a.GetItsValQ;
48       cout  « " and i: " « i.GetItsVal()  « endl;
49       a = i++
50       cout  « "The value of a:  " « a.GetItsVal();
51       cout  « " and i: " « i.GetItsVal()  « endl;
52    return 0;
53    }
```

```
The value of i is 0
The value of i is 1
The value of i is 2
The value of a: 3 and i: 3
The value of a: 3 and i: 4
```

Postfiksni operator je deklarisan u liniji 15, a implementiran u linijama 31-36. Uočite da poziv prefiksnog operatora u liniji 14 ne uključuje celobrojnu zastavicu (x), nego se koristi sa svojom normalnom sintaksom. Postfiksni operator koristi zastavicu (x) da signalizira da se radi o postfiksu, a ne prefiksu. Zastavica (x) se nikada ne koristi.

Preklapanje unarnih operatora

Deklarišite preklopljeni operator kao što bi ste deklarisali funkciju. Upotrebite ključnu reč operator, praćenu operatorom koji će biti preklopljen. Unarne operatorske funkcije ne prihvataju parametre, sa izuzetkom postfiksnog inkrementa i dekrementa, koji prihvataju celobrojnu promenljivu kao zastavicu.

Primer 1:

```
const Counters Counter::operator++ ();
```

Primer 2:

```
Counter Counter::operator-(int);
```

◁f **PAzm** Upotrebite parametar za operator++, ako želite postfiksni operator.

Vratite const referencu na objekat iz operatora++.

Nemojte kreirati privremene objekte kao povratne vrednosti iz operatora++.

Operator sabiranja

Inkrement operator je unarni operator. On operiše samo nad jednim objektom. Operator sabiranja (+) je binarni operator, gde su umešana dva objekta. Kako da implementirate preklapanje operatora 4- za Counter?

Cilj je biti u mogućnosti da se deklarišu dve Counter promenljive, a da se onda saberu, kao u ovom primeru:

```
Counter varOne, varTwo, varThree;
VarThree = VarOne + VarTwo;
```

Mogli biste početi pisanjem funkcije, Add(), koja bi prihvatala Counter kao svoj argument, sabiranjem vrednosti, a onda vratiti Counter sa rezultatom. Listing 10.13 ilustruje ovaj pristup.

Listing 10.13: Funkcija Add().

```
// Listing 10.13
// Funkcija za sabiranje

typedef unsigned short  USHORT;
#include <iostream.h>

class Counter
{
publi c:
   Counter();
   Counter(USH0RT  initial Value);
   ~Counter(){}
   USHORT GetItsVal()const { return itsVal; }
   void SetItsVal(USHORT x) (itsVal = x; }
   Counter Add(const Counter &);

pri vate:
   USHORT itsVal;



Counter::Counter(USHORT  initialValue)
itsVal (initialValue)
()

Counter::Counter() :
itsVal(0)


Counter Counter::Add(const Counter & rhs)
{
   return  Counter(itsVal+  rhs.GetItsVal());
}

int  main()
{
   Counter var0ne(2), varTwo(4), varThree;
   varThree = varOne.Add(varTwo);
   cout « "varOne:  " « varOne.GetItsVal ()« endl;
   cout « "varTwo:  " « varTwo.GetItsVal() « endl;
   cout « "varThree:  " « varThree.GetItsVal() « endl;

return  0;
```

```
varOne: 2
varTwo: 4
varThree: 6
```

l.i:f:iy/.j^ Funkcija Add() je deklarisana u liniji 15. Ona prihvata konstantnu Counter referencu - ona je broj koji treba da se doda tekućem objektu. Ona vraća Counter objekat - rezultat koji treba dodeliti levoj strani iskaza dodele, kao što je prikazano u liniji 38. To znači da je varOne objekat, varTwo je parametar za funkciju Add(), a rezultat se dodeljuje objektu varThree.

Da bi se kreirao objekat varThree, bez potrebe da se inicijalizuje vrednost za njega, zahteva se podrazumevani konstruktor. Podrazumevani konstruktor inicijalizuje itsVal na 0, kao što je prikazano u linijama 26-28. Pošto je potrebno da varOne i varTwo budu inicijalizovani na ne-nula vrednost, bio je kreiran drugi konstruktor, kao što je prikazano u linijama 22-24. Drugo rešenje ovog problema je obezbedivanje podrazumevane vrednosti 0 konstruktoru, koji je deklarisan u liniji 11.

Preklapanje operatora+

Sama funkcija Add() je prikazana u linijama 30-33. Ona radi, ali njena upotreba je neprirodna. Preklapanje operatora + bi učinilo upotrebu klase Counter prirodnijom.

Listing 10.14: Operator+.

```
// Listing 10.14
//Preklopljeni  operator plus  (+)

typedef unsigned short  USHORT;
#include <iostream.h>

class Counter
(
public:
   Counter();
   Counter(USH0RT  initialValue);
   -Counter(){}
   USHORT GetItsVal()const ( return itsVal; }
   void SetItsVal(USHORT x) (itsVal « x; }
   Counter operator* (const Counter &);
private:
   USHORT itsVal;
};

Counter:: Counter (USHORT initialValue):
```

nastavlja se

Listing 10.14: Operator*. *nastavak*

```
21:    UsVal(initialValue)
22:    {}
23:
24:    Counter::Counter():
25:    itsVal(0)
26:    {}
27:
28:    Counter Counter::operator* (const Counter & rhs)
29:    {
30:       return Counter(itsVal + rhs.GetItsVal());
31:    }
32:
33:    int main()
34:    {
35:       Counter var0ne(2), varTwo(4), varThree;
36:       varThree * varOne + varTwo;
37:       cout « "varOne: " « varOne.GetItsVal ()« endl;
38:       cout « "varTwo: " « varTwo.GetItsVal() « endl;
39:       cout « "varThree: " « varThree.GetItsVal() « endl ·
40:
41:    return 0;
42:    }
```

```
varOne: 2
varTwo: 4
varThree: 6
```

operator* se deklariše u liniji 15, a definiše u linijama 28-31. Uporedite ovo sa deklaracijom i definicijom funkcije Add () u prethodnom listingu; one su skoro identične. Ipak, sintaksa njihove upotrebe se prilično razlikuje! Prirodnije je reći ovo:

```
varThree « varOne + varTwo;
```

nego reći:

```
varThree ' varOne.Add(varTwo);
```

Nije velika promena, ali je dovoljna da program bude lakši za korišćenje i razumevanje.

^|ИАР0М1МА^ Tehnike korišćene za preklapanje operatora++ se mogu primeniti na druge unarne operatore, kao što je operator-.

Preklapanje operatora: binarni operatori

Binarni operatori se kreiraju kao unarni, osim što oni prihvataju parametar Parametar je konstantna referenca na objekat istog tipa.

Primer 1:

```
Counter Counter::operator+ (const Counter & rhs);
```

Primer 2:

```
Counter Counter::operator-(const Counter & rhs);
```

Izdanja u preklapanju operatora

Preklopljeni operatori mogu biti funkcije članice, kao što je opisano u ovom Danu, ili funkcije ne-članice koje će biti objašnjene u Danu 14, "Specijalne klase i funkcije", kada budemo govorili o friend funkcijama.

Jedini operatori koji moraju biti članovi klase su operator dodele (=), podskriptni operator[ ], operator poziva funkcije (()) i operator indirekcije (->).

Ograničenja u preklapanju operatora

Operatori za ugradene tipove (kao što je int) se ne mogu preklopiti. Redosled prvenstva se ne može promeniti, kao ni i aritativnost operatora, to jest, da li je unarni, ili binarni. Ne možete izmisliti nove operatore, pa tako ne možete deklarisati ** kao operator "snaga od"

Ju'jj ii^wJMj: *Aritativnost* se odnosi na to koliko operanada se koristi u operatoru. Neki C++ operatori su unarni i koriste samo jedan operand (myValue++). Neki operatori su binarni i koriste dva operanda (a+b). Samo jedan operator je ternarni i koristi tri operanda. Operator ? se, često, naziva ternarni operator, jer je on jedini ternarni operator u C++ (a > b ? x : y).

v

Sta preklopiti?

Preklapanje operatora je jedan od aspekata C++-a koji novi programed preterano koriste i zloupotrebljavaju. Izazovno je kreirati nove i interesantne upotrebe za neke nejasne operatore, ali ovo neizbežno vodi do koda koji je zbunjujući i težak za čitanje.

Naravno, kreirati od operatora + operator oduzimanja i od operatora * operator sabiranja može biti zabavno, ali ni jedan profesionalni programer ne bi to uradio. Veća opasnost leži u dobronamerenoj, ali idiosinkratičnoj upotrebi operatora - korišćenje + za povezivanje serije slova, ili / za rastavljanje stringa. Postoji dobar razlog da se razmotre ove upotrebe, ali postoji čak i bolji razlog da se zadrži opreznost. Zapamtite: cilj preklapajućih operatora je da se uveća upotrebljivost i razumevanje.

<j| **PAZITI** *U* Upotrebite preklapanje operatora kada će ono razjasniti program.

Nemojte kreirati brojač-intuitivne operatore.

Nemojte vrac'ati objekat klase iz preklopljenih operatora.

Operator dodele

Poslednja funkcija koju kompajler obezbeduje, ako je Vi ne specificirate, je operator dodele (operators. On se poziva uvek kada obavljate dodeljivanje objektu:

```
CAT cat0ne(5,7);
CAT catTwo(3,4);
// ... ovde dolazi ostali kod
catTwo = catOne;
```

Ovde se catOne kreira i inicijalizuje sa itsAge, koji je jednak 5, i itsWeight, koji je jednak 7. Onda se kreira catTwo i dodeljuju mu se vrednosti 3 i 4.

Posle izvesnog vremena, objektu catTwo se dodeljuju vrednosti u catOne. Ovde se javljaju dva problema: šta se dešava ako je itsAge pokazivač i šta se dešava sa originalnim vrednostima u catTwo?

Rukovanje promenljivim Članicama koje Čuvaju svoje vrednosti na slobodnom skladištu je objašnjeno ranije, tokom ispitivanja konstruktora kopije. Isti problemi se i ovde pojavljuju, kao što ste videli na slikama 10.1 i 10.2.

C++ programeri uočavaju razliku izmedu plitke, ili članstvo-odnosne, i duboke kopije. Plitka kopija kopira samo članove i oba objekta završavaju, pokazujući na isto područje na slobodnom skladištu. Duboka kopija alocira neophodnu memoriju.

Postoji dodatni problem sa operatorom dodele. Objekat catTwo već postoji i ima alociranu memoriju. Ta memorija se mora obrisati, da ne bi došlo do memorijske pukotine. Ali šta se dešava ako objekat catTwo dodelite samom sebi?

```
catTwo = catTwo;
```

Niko ovo neće uraditi zbog neke svrhe, ali program mora biti sposoban da reši i to. Što je još važnije, moguće je da se ovo desi slučajno, kada reference i dereferencirani pokazivači sakriju činjenicu da se radi o dodeli objekta samom sebi.

Ako ne rešite ovaj problem pažljivo, catTwo će obrisati svoju memorijsku alokaciju. Kada on bude bio spreman za kopiranje memorije sa desne strane dodele, doći će do velikog problema: memorije neće biti.

Da biste se zaštitili od ovoga, Vaš operator dodele mora proveriti da li je desna strana operatora dodele sam objekat. On radi ovo, ispitujući pokazivač `this`. Listing 10.15 prikazuje klasu sa operatorom dodele.

Listing 10.15: Operator dodele.

```
// Listing 10.15

// Konstruktori kopije

linclude <iostream.h>

class CAT
{
public:
    CATQ.                          // podrazumevani konstruktor
// izostavljeni konstruktor kopije i destruktor!
    int GetAgeO const { return *itsAge; }
    int GetWeightO const { return *itsWeight; }
    void SetAge(int age) { *itsAge = age; }
    CAT operator=(const CAT &);
private:
    int *itsAge;
    int *itsWeight;
};

CAT::CAT()
{
    itsAge = new int;
    itsWeight = new int;
    *itsAge = 5;
    *i tsWeight = 9;
}

CAT CAT::operator=(const CAT & rhs)
{
  if (this == &rhs)
    return *this;
  delete itsAge;
  delete itsWeight;
  itsAge = new int;
  itsWeight = new int;
  *itsAge = rhs.GetAgeO;
  *itsWeight = rhs.GetWeightO;
  return *this;
}

int main()
{
    CAT frisky;
    cout « "frisky's age: " « frisky.GetAgeO « endl;
    cout « "Setting frisky to 6...\n";
    frisky.SetAge(6);
    CAT whiskers;
    cout « "whiskers' age: " « whiskers.GetAgeO « endl;
    cout « "copying frisky to whiskers...\n";
    whiskers = frisky;
    cout « "whiskers' age: " « whiskers.GetAgeO « endl;
  return 0;
}
```

```
frisky's age: 5
Setting frisky to 6...
whiskers' age: 5
copying frisky to whiskers...
whiskers' age: 6
```

Listing 10.15 vraća klasu CAT i isključuje konstruktor kopije i destruktor, da bi se uštedeo prostor. U liniji 14 deklariše se operator dodele, a u linijama 30-41 se definiše.

U liniji 32 tekući objekat (CAT kome se dodeljuje) se testira, da bi se videlo da li je isti kao i CAT koji se dodeljuje. Ovo se ostvaruje proverom da li je adresa od rhs ista kao i adresa koja se čuva u pokazivaču thi s.

Ovo funkcioniše dobro za jednostruko nasledivanje, ali ako koristite višestruko nasleđivanje, kao što je objašnjeno u Danu 13, "Polimorfizam", ovaj test neće uspeti. Alternativni test je da dereferencirate pokazivač thi s i vidite da li su ova dva objekta ista:

```
if (*this == rhs)
```

Naravno, operator jednakosti (==) se, takođe, može preklopiti., što Vam dozvoljava da odredite za sebe šta znači za Vaše objekte da su jednaki.

Operatori konverzije

Sta se dešava kada pokušate da dodelite promenljivu ugradenog tipa, kao što je i nt, ili unsigned short, objektu korisnički-definisane klase? Listing 10.16 vraća klasu Counter i pokušava da dodeli promenljivu tipa USHORT objektu klase Counter.

<**UPOZORENJE**| Listing 10.16 se neće kompajliroti!

Listing 10.16: Pokušaj da se klasa Counter dodeli tipu USH0R

```
1:    // Listing 10.16
2:    // Ovaj kod se neće kompajlirati!
3:
4:    typedef unsigned short  USHORT;
5:    #include <iostream.h>
6:
7:    class Counter
8:    {
9:       public:
10:         Counter();
11:         -CounterQU
12:         USHORT GetItsVal Oconst { return itsVal; }
13:         void SetItsVal(USHORT x) {itsVal = x; }
14:      private:
15:         USHORT itsVal;
```



```
};

Counter::Counter():
itsVal(0)
{}

int  main()
{
   USHORT theShort = 5;
   Counter theCtr = theShort;
   cout « "theCtr:  " « theCtr.GetItsVal()  « endl;
return ;0
}
      Compiler error! Unable to convert USHORT to Counter
```

ШСТЕф^ Klasa Counter deklarisana u linijama 7-17 ima samo podrazumevani : konstruktor. Ona ne deklariše ni jednu posebnu metodu za pretvaranje USH0RT-a u Counter objekat, pa, zato, linija 26 prouzrokuje grešku pri kompajliranju. Kompajler ne može znati, osim ako mu ne kažete, da bi on trebalo da dati USHORT dodeli promenljivoj članici itsVal.

Listing 10.17. Konvertovanje USHORT u Counter.

```
I     // Listing 10.17
2     // Konstruktor kao operator konverzije
3
4     typedef unsigned short  USHORT;
5     #include <iostream.h>
6
7     class Counter
8     {
9        public:
10          Counter();
11          Counter(USH0RT val);
12          -Counter(){}
13          USHORT GetItsValOconst { return itsVal; }
14          void SetItsVal(USHORT x) {itsVal = x; }
15       private:
16          USHORT itsVal;
17
18
19
20    Counter::Counter():
21    itsVal(0)
22
23
24 Counter: :Counter(USHORT val):
```

nastavlja se

Listing 10.17. Konvertovanje USHORT u Counter.

nastavak

```
25      itsVal(val)
26      {}
27
28
29      int  main()
30      {
31         USHORT theShort = 5;
32         Counter theCtr = theShort;
33         cout « "theCtr:  " « theCtr.GetItsVal()  «  endl;
34       return  0;
35      }
```

```
theCtr: 5
```

Važna promena je u liniji 11, gde je konstruktor preklopljen da prihvata USHORT, i u linijama 24-26, gde je konstruktor implementiran. Efekat ovog konstruktora je da kreira Counter iz USHORT.

Sa ovim kompajler je sposoban da pozove konstruktor koji prihvata USHORT kao svoj argument. Ipak, šta se dešava ako izokrenete dodelu sa sledećim?

```
Counter  theCtr(5);
USHORT theShort = theCtr;
cout « "theShort  :  " « theShort  «  endl;
```

Ovo će generisati grešku pri kompajliranju. Iako kompajler sada zna kako da kreira Counter iz USHORT-a, on ne zna kako da izokrene proces.

Operatori konverzije

Da bi se rešili ovaj i slični problemi, C++ obezbeduje operatore konverzije, koji se mogu dodati Vašoj klasi. Ovo dozvoljava Vašoj klasi da odredi kako da obavi implicitne konverzije u ugradene tipove. Listing 10.18 ilustruje ovo. Ipak, jedna napomena: operatori konverzije ne specifikuju povratnu vrednost, čak iako, u stvari, vraćaju konvertovanu vrednost.

Listing 10.18. Konvertovanje iz Counter u unsigned short().

```
1:  // Listing  10.18
2:  // operator  konverzije
3:
4:  typedef unsigned short  USHORT;
5:  #include <iostream.h>
6:
7:  class Counter
8:  {
9:     public:
10:       Counter();
```



```
      Counter(USH0RT val);
      -Counter(){}
      USHORT GetItsValOconst  {  return itsVal;  }
      void SetItsVal(USHORT x)  {itsVal = x;  }
      operator  unsigned  short();
   private:
      USHORT itsVal;


  };

  Counter::Counter():
  itsVal(0)
  {}

Counter::Counter(USH0RT val):
itsVal(val)
{}

Counter::operator unsigned short ()
{
   return  (  USHORT  (itsVal)  );


int  main()
{
   Counter  ctr(5);
   USHORT theShort = ctr;
   cout  «  "theShort:  "  «  theShort  «  endl;
    return  0;
}
```

```
theShort: 5
```

U liniji 15 se deklariše operator konverzije. Uočite da on nema povratnu vrednost. Implementacija ove funkcije je u linijama 29-32. Linija 31 vraća vrednost od itsVal, konvertovanu u USHORT.

Sada kompajler zna kako da pretvori USHORT u Counter objekte i obratno i oni se mogu slobodno dodeliti jedan drugom.

Rezime

Danas ste naučili kako da preklopite funkcije članice Vaših klasa. Takode ste naučili kako da funkcijama obezbedite podrazumevane vrednosti i kako da odlučite kada da koristite podrazumevane vrednosti, a kada da obavite preklapanje.

Preklapajući konstruktori Vam dozvoljavaju da kreirate fleksibilne klase, koje se mogu kreirati iz drugih objekata. Inicijalizacija objekata se obavlja u inicijalizacionoj etapi konstrukcije i ona je efikasnija od dodeljivanja vrednosti u telu konstruktora.

Kompajler obezbeduje konstruktor kopije i operator dodele, ako Vi ne kreirate sopstvene, ali oni obavljaju člansko-odnosnu kopiju klase. U klasama, u kojima podaci članovi uključuju pokazivače na slobodno skladište, ovi metodi se moraju redefinisati, tako da alocirate memoriju za odredišni objekat.

Skoro svi C++ operatori se mogu preklopiti, iako je potrebno da budete oprezni da ne kreirate operatore čija je upotreba brojačko-intuitivna. Ne možete promeniti aritativnost operatora, niti možete izmisliti nove operatore.

Pokazivač this se odnosi na tekući objekat i on je nevidljiv parametar u svim funkcijama članicama. Dereferencirani pokazivač this se često vraća iz preklopljenih operatora.

Operatori konverzije Vam dozvoljavaju da kreirate klase - one se mogu koristiti u izrazima koji očekuju druge tipove objekata. Oni su izuzeci od pravila da sve funkcije vraćaju eksplicitnu vrednost; kao konstruktori i destruktori, oni nemaju povratni tip.

Pitanja i odgovori

P Zašto biste ikada koristili podrazumevane vrednosti, s obzirom da možete preklopiti funkciju?

0 Lakše je održavati jednu funkciju, nego dve i često je lakše razumeti funkciju sa podrazumevanim parametrima, nego proučavati tela dve funkcije. Osim toga, ažuriranje jedne od funkcija i nemarnost da se ažurira druga je uobičajeni izvor bagova.

P Zbog problema sa preklopljenim funkcijama, zašto umesto njih ne koristiti uvek podrazumevane vrednosti?

O Preklopljene funkcije obezbeđuju mogućnosti koje ne postoje sa podrazumevanim vrednostima, kao što je variranje liste parametara po tipu, a ne samo po broju.

P Pri pisanju konstruktora klase, kako određujete šta da stavite u inicijalizaciju, a šta da stavite u telo konstruktora?

0 Jednostavno praktično pravilo je da se uradi koliko je moguće vise u inicijalizacionoj fazi, što znači da tu treba inicijalizovati sve promenljive članice. Neki, kao što su proračuni i iskazi za štampanje, moraju biti u telu konstruktora.

P **Da li preklopljena funkcija može imati** podrazumevani **parametar?**

O Da. Nema razloga da se ove moćne karakteristike ne kombinuju. Jedna, ili vise preklopljenih funkcija članica mogu imati sopstvene podrazumevane vrednosti, sledeći normalna pravila za podrazumevane promenljive u svakoj funkciji.

P **Zašto se neke funkcije članice definišu unutar klasne deklaracije,** a **druge ne?**

O Definisanje implementacije funkcije članice unutar deklaracije čini da ona bude inline. Generalno, ovo se radi samo ako je funkcija krajnje jednostavna. Uočite da takođe možete učiniti funkciju inline , korišćenjem ključne reči inline, čak iako je funkcija deklarisana izvan deklaracije klase.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje obradene teme i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i proverite da li razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Kada preklopite funkcije članice, po čemu se one moraju razlikovati?
2. Kakva je razlika izmedu deklaracije i defmicije?
3. Kada se konstruktor kopije poziva?
4. Kada se poziva destruktor?
5. Po čemu se konstruktor kopije razlikuje od operatora dodele (=)?
6. Šta je pokazivač this?
7. Kako pravite razliku između preklapanja prefiksnog i postfiksnog inkrement operatora?
8. Da li možete preklopiti operator* za kratke celobrojne vrednosti?
9. Da li je legalno u C++-u preklopiti operator++ tako da on dekrementira vrednost u Vašoj klasi?
10. Koju povratnu vrednost moraju imati operatori konverzije u svojim deklaracijama?

Vežbe

1. Napišite deklaraciju klase `SimpleCircle` samo sa jednom promenljivom članicom: `itsRadius`. Uključite podrazumevani konstruktor, destruktor i metode pristupa za radijus.
2. Korišćenjem klase koju ste kreirali u Vežbi 1, napišite implementaciju podrazumevanog konstruktora, inicijalizujući itsRadius vrednošću 5.
3. Koristeći istu klasu, dodajte drugi konstruktor, koji prihvata vrednost kao svoj parametar i dodeljuje tu vrednost promenljivoj `itsRadius`.

4. Kreirajte prefiksni i postfiksni inkrement operator za Vašu klasu **SimpleCi rcle koji** inkrementiraju **i tsRadi us.**

5. Promenite **SimpleCircle, tako da** čuva **itsRadius** na slobodnom skladištu, i popravite postojeće metode.

6. Obezbedite konstruktor kopije za **Si mpl eCi** rcl **e.**

7. Obezbedite operator dodele za **SimpleCircle.**

8. Napišite program **koji** kreira dva **SimpleCircle** objekta. Upotrebite podrazumevani konstruktor za jedan i instancirajte drugi sa vrednošću 9. Pozovite inkrement operator za svaki, a, onda odštampajte njihove vrednosti. Konačno, dodelite drugi prvom i odštampajte njegove vrednosti.

9. ISTERIVAČ BAGOVA: Šta nije u redu sa ovom implementacijom operatora dodele?

```
SQUARE SQUARE ::operator=(const SQUARE & rhs)
{
      itsSide = new int;
      *itsSide = rhs.GetSide();
      return *this;
}
```

10. ISTERIVAČ BAGOVA: Šta nije u redu sa ovom implementacijom operatora sabiranja?

```
VeryShort  VeryShort::operator+ (const VeryShort& rhs)
{
   itsVal += rhs.GetItsVaU);
   return *this;
}
```


Dan 11

Nizovi

U prethodnim poglavljima deklarisali ste jednostruke `int`, `char` i druge objekte. Često ste poželeli da deklarišete kolekciju objekata, kao što je 20 `int`-ova, ili tuce mačaka. Danas ćete naučiti:

- šta su nizovi i kako se deklarišu
- šta su stringovi i kako se nizovi karaktera koriste za njihovo pravljenje
- odnos izmedu nizova i pointera
- kako se koristi aritmetika pointera sa nizovima.

v

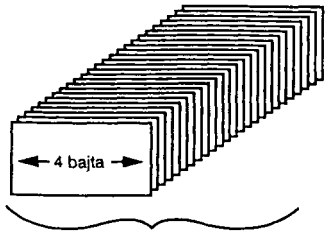
Sta je niz?

Niz je kolekcija lokacija za smeštanje podataka, od kojih svaka sadrži isti tip podataka. Sve lokacije za smeštanje podataka se nazivaju elementi niza.

Deklaracija niza se vrši odredivanjem tipa, iza čega slede ime niza i subscript. Subscript je broj elemenata niza koji je smešten u srednje zagrade. Na primer,

```
1ong LongArray[25];
```

deklariše niz od 25 long integer-a, sa imenom `LongArray`. Kada kompajler primeti ovu deklaraciju, on će odvojiti dovoljno memorije koja će sadržati svih 25 elemenata. Pošto svaki `long` integer zahteva četiri bajta, ova deklaracija će odvojiti 100 povezanih bajtova memorije, kao što je ilustrovano na slici 11.1.



Slika 11.1
Deklarisanje niza

Elementi niza

Svakom elementu niza se može pristupiti referenciranjem ofseta u imenu niza. Elementi niza se broje od nule. Stoga, prvi element niza je arrayName *[0]. U* primeru za LongArray, LongArray[0] je prvi element, LongArray[1] je drugi i tako dalje.

Ovo deluje malo konfuzno. Niz SomeArray[3] ima tri elementa. Ti elementi su: SomeArray [0], SomeArray [1] i SomeArray [2]. Uopšteno govoreći, SomeArray [n] ima *n* elemenata koji su numerisani od SomeArray [01 do SomeArray [n-1].

Na primer, LongArray[25] je numerisano od LongArray[0] do LongArray[24]. U listingu 11.1 prikazano je kako se deklariše niz od 5 integer-a i kako se popunjava svaki od njih.

Listing 11.1; Korišćenje celobrojnog niza.

```
1:    //Listing 11.1 - Nizovi
2:    #include <iostream.h>
3:
4:    int main()
5:    {
6:       int myArray[5];
7:       int i;
8:       for ( i=0; i<5; i++) //0-4
9:       {
10:          cout « "Value for myArray[" « i « "J: ";
П:          cin » myArray[ i ];
12:      }
13:      for (i = 0; i<5; i++)
14:         cout « i « ": " « myArray[i] « "\n";
15:   return 0;
16:   }
```

JH^ПJJ^
```
Value for myArray[ 0] : 3
Value for myArray[1]:   6
Value for myArray[2]:   9
Value for myArray[3]:   12
Value for myArray[4]:   15
0: 3
1: 6
2: 9
3: 12
4: 15
```

III M III ^ Linija 6 deklariše niz, pod imenom myArray, koji sadrži pet celobrojnih promenljivih. Linija 8 ostvaruje petlju koja broji od 0-4, što je odgovarajući set ofset-a za petočlani niz. Od korisnika je traženo da unese vrednost i ta vrednost se smešta u odgovarajući ofset unutar niza.

Prva vrednost je sačuvana u MyArray [0], druga u MyArray [1] i tako dalje. Druga for petlja prikazuje sve vrednosti na ekranu.

NAPOMENA^ Nizovi se broje od nule, **a** ne od 1. Ovo je uzrok mnogih bagova u programima koji **pišu** početnici u C+ +. Kad god koristite nizove, imajte na umu da, na primer, niz sa 10 elemenata počinje brojanje od ArrayName[0] do ArrayName[9] i da ne postoji ArrayName[10].

Pisanje iza kraja niza

Kada upisujete vrednost u jedan element niza, kompajler izračunava gde da smesti vrednost na osnovu veličine svakog elementa i subscript-a. Pretpostavimo da ste tražili da upišete vrednost u LongArray[5], koja je šesti element. Kompajler će pomnožiti ofset 5 sa veličinom svakog elementa (u ovom slučaju 4) i zatim će se pomeriti toliko bajtova (20) od početka niza i upisati novu vrednost na tu lokaciju.

Ako zatražite da upišete u LongArray [50], kompajler će ignorisati činjenicu da postoji takav element. On će izračunati koliko je ta lokacija udaljena od prvog elementa (200 bajtova) i zatim će izvršiti upis na tu lokaciju, bez obzira šta se u njoj nalazi. To može biti praktično bilo koji podatak i Vaš upis na tu lokaciju može dovesti do nepredvidivih rezultata. Ako imate dovoljno sreće, Vaš program će trenutno pasti. U suprotnom, dobićete čudne rezultate, i to mnogo kasnije u Vašem programu, i biće Vam potrebno mnogo vremena i napornog rada da ustanovite šta je krenulo naopako.

Kompajler je kao slep čovek koji pronalazi put do svoje kuće. On kreće od prve kuće MainStreet[0]. Kada od njega zatražite da ode do šeste kuće u MainStreetu, on to razume na sledeći način: "Moram da odem pet kuća napred, a svaka kuća duga je četiri velika koraka, što znači da moram da predem 20 koraka.". Ako od njega zatražite da ode do MainStreet [100], a MainStreet ima samo 20 kuća, on će preći 400 koraka.

Listing 11.2 prikazuje šta se može dogoditi kada pišete iza kraja niza.

vUPOZORENJEy Nemojte startovati ovaj program. On može da obori Vaš sistem.

Listing 11.2: Zapisivanje izvan opsega niza.

```
1:    //Listing 11.2
2:    // Demonstrira ono Sto se deava kada zapisujete izvan opsega
3:    //  niza
4:
5:    linclude <iostream.h>
6:    int main()
7:    {
8:       // stražari
9:       long  sentinel0ne[3];
10:      long TargetArray[25]; // niz koji treba popuniti
11:      long sentinelTwo[3];
12:      int  i;
13:      for (i=0; i<3; i++)
14:         sentinelOne[i] = sentinelTwo[i] = 0;
15:
16:      for (i=0;  i<25;  i++)
17:         TargetArray[i] = 0;
18:
19:      cout « "Test 1: \n";   // testiraju se tekuće vrednosti (trebalo bi da budu
0)
20:      cout « "TargetArray[0]: " « TargetArray[0] « "\n";
21:      cout « "TargetArray[24]: " « TargetArray[24] « "\n\n";
22:
23:      for (i = 0; i<3; i++)
24:      {
25:         cout « "sentinelOne[" « i « "]: ";
26:         cout « sentinel0ne[ i ] « "\n";
27:         cout « "sentinelTwo[" « i « "]: ";
28:          cout « sentinelTwo[i ]« "\n";
29:      }
30:
31:      cout « "\nAssigning...";
32:      for (i = 0; i<=25; i++)
33:         TargetArray[i] = 20;
34:
35:      cout « "\nTest 2: \n";
36:      cout « "TargetArray[0]: " « TargetArray[0] « "\n";
37:      cout « "TargetArray[24]: " « TargetArray[24] « "\n";
38:      cout « "TargetArray[25]: " « TargetArray[25] « "\n\n";
39:      for (i = 0; i<3; i++)
40:      {
41:         cout « "sentinelOne[" « i « "]: ";
42:         cout « sentinelOne[ i ]« "\n";
43:         cout « "sentinelTwo[" « i « "]: ";
44:         cout « sentinel Two [i ]« "\n";
45:      }
46:
47:   return 0;
48:   }
```

```
Test 1:
TargetArray[0]: 0
TargetArray[24]: 0

SentinelOne[0]: 0
SentinelTwo[0]: 0
SentinelOne[1]: 0
SentinelTwo[1]: 0
SentinelOne[2]: 0
SentinelTwo[2]: 0

Assigning...
Test 2:
TargetArray[0]: 20
TargetArray[24]: 20
TargetArray[25]: 20

SentinelOne[0]: 20
SentinelTwo[0]: 0
SentinelOne[1]: 0
SentinelTwo[1]: 0
SentinelOne[2]: 0
SentinelTwo[2]: 0
```

Linije 9 i 10 deklarisu dva niza od tri integer-a, koji se ponašaju kao tampon-zona oko TargetArray. Tampon-zona je inicijalizovana vrednošću 0. Ako se izvrši upis izvan TargetArray, tampon-zona će biti promenjena. Neki kompajleri memoriju broje "nadole", drugi pak "nagore." Stoga je tampon-zona smeštena sa obe strane TargetArray. Linije 19-29 potvrduju vrednosti tampon-zone u testu 1. U liniji 33 članovi TargetArray su svi inicijalizovani na vrednost 20, ali je brojač došao do ofseta 25, koji ne postoji u TargetArray.

Linije 36-38 prikazuju vrednosti iz TargetArray u testu 2. Primetićete da će TargetArray od 25 vrlo rado ispravno prikazati vrednost 20. Medutim, kada se prikažu Sentinel One i Sentinel Two, Sentinel Two [0] će pokazati svoju promenjenu vrednost. Razlog je, što je 25. element posle TargetArray [0] ista memorijska lokacija kao i Sent inel Two [0]. Kada je pristupljeno nepostojećem elementu TargetArray, u stvari se pristupilo elementu Senti nel Two [0].

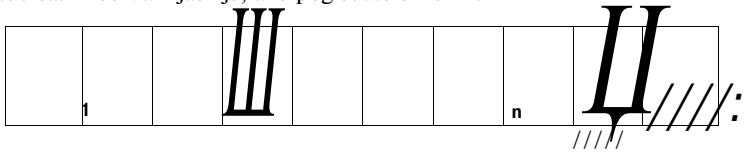
Ovaj nezgodan bag je vrlo teško pronaći, pošto je vrednost Sent i nel Two[0] izmenjena u delu koda koji se uopšte nije pozivao na niz Sentinel Two.

Ovaj kod koristi magične brojeve, kao što je 3, za veličinu Sentinel niza i 25 za veličinu TargetArray. Sigurnije je koristiti konstante, tako da možete izmeniti sve ove vrednosti na jednom mestu.

Fence Post Errors

Često se događa da se upis vrši za jednu lokaciju vise, nego što je to veličina niza i taj bag ima sopstveno ime. On se naziva *Fence Post Error.* Ovim se označava problem u brojanju stubića, koji su Vam potrebni za 10 metara ograde, ako Vam je potrebna jedna tabla ograde za svaki metar. Većina ljudi će odgovoriti 10, ali Vam treba 11 stubića. Biće Vam jasnije, ako pogledate sliku 11.2

Slika 11.2
Fence Post Errors



Ova vrsta brojanja je mučnina za svakog programera. Vremenom, međutim, shvatićete ideju, da 25-elementni niz sadrži samo do elementa 24 i da se sve broji od nule.

HAPOMEMAy Neki programeri ArrayName [0] nazivaju nultim elementom. Preuzimanje ove prakse je velika greška. Ako je ArrayName [0] nulti element, šta je ArrayName [1]? Ako je tako, kada vidite ArrayName [24], da li ćete shvatiti da to nije 24. nego 25. element. Mnogo je bolje reći da je ArrayName [0] prvi element sa ofsetom 0.

Inicijalizacija nizova

Možete izvršiti inicijalizaciju jednostavnih nizova ugradenih tipova, kao što su integers i karakteri, kada prvi put deklarišete niz. Posle imena niza stavite znak jednakosti (=) i listu zarezima odvojenih vrednosti, koje se nalaze u vitičastim zagradama. Na primer,

```
int IntegerArray[5] = { 10, 20, 30, 40, 50 };
```

će deklarisati IntegerArray kao niz od 5 integer-a, zatim će se IntegerArray[0] dodeliti vrednost 10, IntegerArray[1] vrednost 20 i tako dalje.

Ako izostavite veličinu niza, niz će biti dovoljno veliki da sadrži inicijalizaciju, koju ste naveli, pa će, stoga, ako napišete

```
int IntegerArray[ ] = { 10, 20, 30, 40, 50 };
```

biti kreiran potpuno isti niz kao i u prethodnom primeru.

Ako je neophodno da znate veličinu niza, možete tražiti od kompajlera da Vam je izračuna. Na primer,

```
const USHORT IntegerArrayLength;
IntegerArrayLength = sizeof(IntegerArray)/sizeof(IntegerArray[0]);
```

će postaviti promenljivu USHORT na rezultat koji će nastati deljenjem veličine celog niza sa veličinom individualnog elementa niza.

Ne možete izvršiti inicijalizaciju vise elemenata nego što ste deklarisali u nizu.

```
int IntegerArray[5] = { 10, 20, 30, 40, 50, 60};
```

dovesti do greške u kompilaciji, s obzirom da ste deklarisali petoelementni niz, a izvršili inicijlalizaciju šest vrednosti. Medutim, sasvim ispravno je napisati

```
int IntegerArray!5] = { 10, 20, 30, 40, 50, 60};
```

Pošto neinicijalizovani članovi niza nemaju garantovane vrednosti, oni će biti inicijalizovani na 0. Ako ne izvršite inicijalizaciju nekog člana niza, njegova vrednost će biti postavljena na nulu.

<jt; PAПП Dozvolite kompajleru da odredi veličinu inicijalizovanih nizova. Nemojte pisati iza kraja niza.

Dajte nizovima imena sa smislom, kao što biste dali bilo kojoj drugoj promenljivoj.

Zapamtite da prvi element niza ima ofset 0.

Deklarisanje nizova

Nizovi mogu imati bilo koje legalno ime promenljive, ali ne mogu imati isto ime kao i druge promenljive, ili nizovi unutar svog opsega. Stoga, ne možete imati niz pod imenom MyCats[5] i promenljivu pod imenom MyCats u istom trenutku.

Možete dimenzionisati veličinu niza sa const, ili sa nabrojanim tipom. Listing 11.3 ovo ilustruje.

Listing 11.3: Korišćenje const i enum u nizovima.

```
// Listing 11.3
// Dimenzionisanje nizova sa konstantama i enumeracijama.

#include <iostream.h>
int main()
{
   enum WeekDays { Sun, Mon, Tue,
        Wed, Thu, Fri, Sat, DaysInWeek };
   int ArrayWeek[DaysInWeek] = { 10, 20, 30, 40, 50, 60, 70 };

   cout « "The value at Tuesday is: " « ArrayWeek[Tue];
 return 0;
}
```

SBB^fr **The va1ue at Tuesday is: 30**

IIIIIE^p Linija 7 kreira nabrojani niz pod imenom WeekDays. On ima sedam članova. Nedelja je jednaka nuli, a DaysInWeek je jednak 7.

Linija 11 koristi nabrojanu konstantu Tue kao ofset u niz. Pošto se Tue pretvara u 2, treći element niza, DaysInWeek[2], je vraćen i prikazan u liniji 11.

Nizovi

Da biste deklarisali niz, napišite tip objekta iza koga sledi ime niza i subscript sa bro jem objekata, koji će biti sadržani u nizu.

Primer 1:

```
int MyIntegerArray[90];
```

Primer 2:

```
long * ArrayOfPointersToLongs[100];
```

Da biste pristupili članovima niza, koristite subscript operator.

Primer 1:

```
int theNinethInteger = MyIntegerArray[8];
```

Primer 2:

```
long * pLong = ArrayOfPointersToLongs[8]
```

Nizovi se broje od nule. Niz od n članova je numerisan od 0 do n-1.

Nizovi objekata

Svi objekti, bez obzira da li su ugradeni, ili korisnički definisani, mogu se smestiti u niz. Kada deklarišete niz, Vi kompajleru saopstavate tip objekta i broj objekata za koji želite da alocirate prostor. Kompajler zna koliko prostora je neophodno za svaki objekat, na osnovu deklaracije klase. Klasa mora da ima podrazumevani konstruktor, koji ne uzima elemente, tako da objekti mogu da budu kreirani kada se definiše niz.

Pristup podacima članovima jednog niza objekata je proces iz dva koraka. Identifikacija člana niza se vrši korišćenjem index operatora ([ ]) i zatim se dodaje operator (.), da bi se pristupilo odredenoj promenljivoj članu. U listingu 11.4 demonstrirano je kako bi se mogao kreirati niz od pet CAT-ova.

Listing 11.4: Kreiranje niza objekata.

```
1:  // Listing 11.4 - Niz objekata
2:
3:  linclude <iostream.h>
4:
5:  class CAT
6:  {
7:     public:
8:        CAT() { itsAge = 1; itsWeight=5; }
9:        -CATO {}
10:       int GetAgeO const { return itsAge; }
11:       int GetWeightO const { return itsWeight; }
12:       void SetAge(int age) { itsAge = age; }
```


```
   private:
      int itsAge;
      int itsWeight;
};

int main()
{
   CAT Litter[5];
   int i;
   for (i = 0; i < 5; i++)
      Litter[i].SetAge(2*i +1);

   for (i = 0; i < 5; i++)
   {
      cout « "Cat F « i+l« ": ";
      cout « Litter[i ] .GetAgeO « endl;
   }
 return 0;
}
```

```
cat #1: 1
cat #2: 3
cat #3: 5
cat #4: 7
cat #5: 9
```

frarafcK U linijama 5-17 deklarisana je CAT klasa, koja mora da ima podrazumevani konstruktor, tako da CAT objekti mogu da budu kreirani u nizu. Primetićete da, ako kreirate bilo koji drugi konstruktor, podrazumevani konstruktor koji kreira kompajler neće biti uključen; moraćate da kreirate sopstveni.

Prva for petlja (linije 23 i 24) postavlja godine za svaku od pet mačaka u nizu. Druga for petlja (linije 26 i 27) pristupa svim članovima niza i poziva GetAgeO.

Svaki pojedinčni GetAgeO metod je pozvan tako što je pristupljeno odgovarajućem članu niza `Litter[i]`, koji je sledila (.) i funkcija clan.

Višedimenzionalni nizovi

Moguće je imati nizove sa vise dimenzija. Svaka dimenzija se predstavlja kao subscript u nizu. Stoga, dvo-dimenzionalni niz ima dva subscript-a; tro-dimenzionalni ima tri subscript-a i tako dalje. Nizovi mogu imati bilo koji broj dimenzija, ali ipak je verovatno da će većina nizova koji ćete kreirati biti jedno ili dvo-dimenzionalni.

Dobar primer dvo-dimenzionalnog niza je šahovska tabla. Jedna dimenzija predstavlja osam redova; druga dimenzija predstavlja osam kolona. Ova ideja je ilustrovana na slici 11.3.

Pretpostavimo da imate klasu pod imenom Square. Deklaracija niza pod imenom Board bi bila

```
SQUARE Board[8] [8];
```

Takode je moguće iste podatke predstaviti u jednoj dimenziji sa nizom 64 kvadrata. Na primer,

```
SQUARE Board[64]
```

Medutim, ovo ne odgovara objektu iz realnog sveta, kao što je to slučaj sa tro-dimenzionalnim nizom. Kada igra započne, kralj je lociran u 4. poziciji 1. reda. Računajući od nule, ta pozicija odgovara

```
Board[0][3];
```

i podrazumeva se da prvi subscript odgovara redovima, a drugi kolonama. Izgled i pozicije za celu tablu su ilustrovani na slici 11.3

Slika 11.3
Šahovsko tabla i dvo- dimenzionolni niz

Inicijalizacija višedimenzionalnih nizova

Moguće je izvršiti inicijalizaciju višedimenzionalnih nizova. Dodelite listu vrednosti elementima niza u redosledu, tako da se poslednji subscript niza menja, dok svi ostali ostaju isti. Stoga, ako imate niz

```
int  theArray[5][3]
```

prva tri elementa idu u `theArray[0]`, a sledeća tri u `theArray[l]` i tako dalje.

Inicijalizaciju ovog niza ćete izvršiti sa

```
int theArray[5][3] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 }
```

Da bi bilo jasnije, moguće je grupisati inicijalizacije vitičastim zagradama, na primer,

```
int theArray[5][3] = {  {1,2,3},
{4,5,6},
{7,8,9},
{10,11,12},
{13,14,15}  };
```


Kompajler će ignorisati unutrašnje vitičaste zagrade, a ovaj prikaz je jednostavniji za razumevanje kako će brojevi biti distribuirani.

Sve vrednosti moraju biti odvojene zarezima, bez obzira na vitičaste zagrade. Ceo inicijalizacioni set mora biti unutar vitičastih zagrada i mora se završavati znakom tačka-zarez.

Listing 11.5 kreira dvo-dimenzionalni niz. Prva dimenzija je set brojeva od 0 do 5. Druga dimenzija se sastoji od duple vrednosti iz prve dimenzije.

Listing 11.5: Kreiranje višedimemionog niza.

```
#include <iostream.h>
int  main()
{
   int SomeArray[5][2] = { {0,0}, {1,2}, {2,4}, {3,6}, {4,8}};
   for (int i = 0; i<5; i++)
      for (int j=0; j<2; j++)
      {
         cout « "SomeArray [" « i « "][" « j « "]:  ";
         cout « SomeArray [ i ] [ j ]« endl;
      }

 return 0;
}
```

```
SomeArray[0]
SomeArray[0]
SomeArray[1]
SomeArray[1]
SomeArray[2]
SomeArray[2]
SomeArray[3]
SomeArray[3]
SomeArray[4]
SomeArray[4]
```

TTJHП^jjB* U liniji 4 `SomeArray` je deklarisana kao dvo-dimenzionalni niz. Prva dimenzija se sastoji od druga od dva integer-a. Ovim je kreirana 5x2 mrežica, kao na slici 11.4.

Slika 11.4
Niz veličine 5x2 Some Array [5] [2]

Vrednosti su inicijalizovane u parovima, iako su mogle biti izračunate. U linijama 5 i 6 nalazi se ugnježdena for petlja. Spoljna for petlja bira sve članove prve dimenzije. Za svakog člana u toj dimenziji unutrašnja for petlja bira svakog člana druge dimenzije. Ovo je konzistentno sa izlazom. Iza SomeArray[0] [0] sledi SomeArray [0] [1]. Prva dimenzija se inkrementira, tek pošto je druga dimenzija inkrementirana za 1. Druga dimenzija tada kreće ispočetka.

Nešto o memoriji

Kada deklarišete niz, Vi kažete kompajleru koliko objekata želite da smestite u njega. Kompajler odvaja memoriju za sve objekte, čak i ako ih nikada nećete koristiti. Ovo nije problem sa nizovima u kojima znate koliko objekata će Vam biti potrebno. Na primer, šahovska tabla ima 64 kvadrata, a mačka ima izmedu 1 i 10 mačića. Međutim, kada nemate predstavu koliko objekata će Vam biti potrebno, neophodno je da koristite naprednije strukture podataka.

U ovoj knjizi prikazani su nizovi pointera, nizovi izgrađeni u slobodnom prostoru i različite druge kolekcije. Druge naprednije strukture podataka, koje rešava probleme "vezane za" velike smeštajne kapacite su izvan okvira ove knjige. Dve najlepše u programiranju su da uvek postoje novine koje treba naučiti i knjige koje treba pročitati.

Nizovi pointera

Nizovi o kojima smo do sada pričali sve svoje članove smeštaju na "stek". Stek memorija je, obično, veoma limitirana, dok je slobodna memorija mnogo veća. Moguće je deklarisati sve objekte u slobodnoj memoriji i zatim smestiti samo pointere na objekte u niz. Ovim se dramatično redukuje količina iskorišćene stek memorije. U listingu 11.6 na drugi način je kreiran niz iz listinga 11.4, tako da su svi objekti smešteni u slobodnu memoriju. Kao indikacija o veličini memorije koju Vam ovaj metod omogućava, niz je proširen sa 5 na 500 i ime je promenjeno iz Li tter u Family.

Listing 11.6: Cuvanje niza na slobodnom skladištu.

```
// Listing 11.6 - Niz pokazivača na objekte

linclude <iostream.h>
```



```
4
5   class CAT
6   {
7      public:
8         CAT() { itsAge = 1; itsWeight=5; }
9         -CAT() {}                                  // destruktor
10        int GetAgeO const { return itsAge; }
11        int GetWeightO const { return itsWeight; }
12        void SetAge(int age) { itsAge = age; }
13
14     private:
15        int itsAge;
16        int itsWeight;
17  };
18
19  int main()
20  {
21     CAT * Family[500];
22     int i;
23     CAT * pCat;
24     for (i = 0; i < 500; i++)
25     {
26        pCat = new CAT;
27        pCat->SetAge(2*i +1);
28        Family[i] = pCat;
29     }
30
31     for (i = 0; i < 500; i++)
32     {
33        cout « "Cat #" « i+1 «      ";
34        cout « Family[i]->GetAge() « endl;
35     }
36   return 0;
37  }
```

```
Cat #1: 1
Cat #2: 3
Cat #3: 5

Cat #499: 997
Cat #500: 999
```

Objekat CAT, koji je deklarisan u linijama 5-17, identičan je sa CAT objektom deklarisanim u listingu 11.4. Medutim, niz deklarisan u liniji 21 se zove Fami ly i sadrži 500 pointera na CAT objekte.

U inicijalnoj petlji (linije 24-29) 500 novih CAT objekata je kreirano u slobodnom prostoru i svaki od njih ima godine postavljene na dva puta index plus jedan. Stoga, prva mačka i m a godine postavljene na 1, druga na 3, treća na 5 i tako dalje. Na kraju, pointer je dodat nizu.

Pošto je niz deklarisan da koristi pointere, pointer, umesto dereferencirane vrednosti u pointeru, dodat je u niz.

Druga petlja (linije 31 i 32) prikazuje svaku od ovih vrednosti. Pointeru se pristupa korišćenjem Family[i]. Ta adresa je zatim iskorišćena za pristup GetAgeO metodu.

U torn primeru niz Family i svi njegovi pointeri su smešteni na stek, ali 500 CAT kreiranih objekata je smešteno u Slobodan prostor.

Deklarisanje nizova u slobodnom prostoru

Takode je moguće cele nizove smestiti u Slobodan prostor, koji se, takođe, zove i heap. Ovo možete učiniti tako što ćete pozvati New, uz pomoć subscript operatora. Rezultat je pointer na zonu u slobodnom prostoru, koja sadrži niz. Na primer,

```
CAT *Family = new CAT[500];
```

deklariše Family kao pointer na prvi član niza od 500 mačaka. Drugim rečima, Family ukazuje na - ili ima adresu od - Family[0].

Prednost korišćenja Family na ovaj način je u mogućnosti da koristite aritmetiku pointera za pristup svakom članu Family. Na primer, možete napisati

```
CAT *Family = new CAT[500];
CAT *pCat = Family;             // pCat pokazuje na Family[0]
pCat->SetAge(10);               // postavlja Family[0] na 10
pCat++;                         // prelazi na Family[l]
pCat->SetAge(20);               // postavlja Family[l] na 20
```

Ovim ćete deklarisati novi niz od 500 CAT-ova i pointer koji ukazuje na početak niza. Korišćenjem tog pointera prva CAT-ova Set Age 0 funkcija se poziva sa vrednošću 10. Pointer se, zatim, inkrementira, kako bi ukazao na sledeći CAT i poziva se drugi Set Age 0 metod.

Pointer na niz protiv niza pointera

Pogledajte sledeće tri deklaracije:

```
1:  Cat    FamilyOne[500]
2:  CAT * FamilyTwo[500];
3:  CAT * FamilyThree = new CAT[500];
```

FamilyOne je niz od 500 CAT-ova. FamilyTwo je niz od 500 pointera na CAT-ove i Family Three je pointer na niz od 500 CAT-ova.

U trećem slučaju Family Three je pointer na niz. Adresa u Family Three je adresa prve stavke u torn nizu. To je isti slučaj i sa Family One.

Pointeri i imena nizova

U C++ -u ime niza je konstantan pointer na prvi element niza. Stoga, u deklaraciji

```
CAT Family[50];
```

Family je pointer na &[0], što je adresa prvog elementa niza Family.

Legalno je koristiti imena nizova kao konstantne pointere i obratno. Stoga, Family+4 je legitiman način za pristup podacima Family [4].

Kompajler će odraditi svu aritmetiku kada dodajete na, inkrementirate i dekrementirate pointere. Adresa kojoj će biti pristupljeno kada napišete Family + 4 nije četiri bajta iza adrese od Family - ona je četiri objekta iza. Ako je svaki objekat dugačak četiri bajta, Family + 4 je 16 bajtova. Ako je svaki objekat CAT koji ima 4 long promenljive, po četiri bajta svaka, i dve short promenljive, po dva bajta svaka, onda je svaki CAT 20 bajtova i Family+4 je 80 bajtova iza početka niza.

U listingu 11.7 ilustrovani su deklaracija i korišćenje niza u slobodnom prostoru.

Listing 11.7: Kreiranje niza korišćenjem new.

```
1    // Listing 11.7 - Niz na slobodnom skladištu
2
3    iMnclude <iostream.h>
4
5    class CAT
6    {
7       public:
8          CAT() { itsAge = 1; itsWeight=5; }
9          -CAT();
10         int GetAgeO const { return itsAge; }
11         int GetWeightO const { return itsWeight; }
12         void SetAge(int age) { itsAge = age; }
13
14      private:
15         int itsAge;
16         int itsWeight;
17   };
18
19   CAT :: -CAT()
20   {
21     // cout « "Destructor called!\n";
22   }
23
24   int main()
25   {
26      CAT * Family = new CAT[500];
27      int i;
28      CAT * pCat;
```

nastavlja se

M***»MM **Listing 11.7: Kreiranje niza korišćenjem new.**

nastavak

```
29:       for (i = 0; i < 500; i++)
30:       {
31:          pCat = new CAT;
32:          pCat->SetAge(2*i +1);
33:          Family[i] = *pCat;
34:          delete pCat;
35:       }
36:
37:       for (i = 0; i < 500; i++)
38:       {
38:          cout « "Cat Г « i+1 « M: ";
39:          cout « Family[i].GetAgeO K< endl;
40:       }
41:
42:       delete [] Family;
43:
44:     return 0;
45:    }
```

```
            Cat #1: 1
m m m *     Cat #2: 3
            Cat #3: 5

            Cat #499: 997
            Cat #500: 999
```

*IIIIIIIIj^* Linija 26 deklariše niz Family, koji sadrži 500 CAT objekata. Ceo niz je kreiran u slobodnom prostoru sa pozivom newCAT[500].

Svaki CAT objekat, koji je dodat nizu, takode je kreiran u slobodnom prostoru (linija 31). Primetičete, medutim, da pointer ovoga puta nije dodat nizu, dok objekat jeste. Ovaj niz nije niz pointera na CAT. Ovo je niz CAT-ova.

Brisanje nizova iz slobodog prostora

Family je, pointer na niz u slobodnom prostoru. Kada je u liniji 33 pointer pCAT dereferenciran, CAT objekat je smešten u niz (Zašto da ne? Niz je u slobodnom prostoru). Ali pCat je ponovo iskorišćen u sledećoj iteraciji petlje. Da li postoji opasnost, pošto ne postoji pointer na Cat objekat, i da li može doći do "curenja" memorije?

Ovo može biti veliki problem, osim kada brisanje `Family` vrati svu memoriju niza. Kompajler je dovoljno pametan da uništi svaki objekat niza i da vrati njegovu memoriju slobodnom prostoru.

Da biste ovo videli, promenite veličinu niza iz 500 u 10 u linijama 26, 29 i 37. Zatim, skinite komentar sa cout naredbe u liniji 21. Kada se dosegne linija 40 i niz se uništi, biće pozvan destruktor za svaki CAT objekat.

Kada kreirate stavku na heap-u, uz pomoć naredbe new, uvek tu stavku brišete i oslobadate njenu memoriju, uz pomoć delete. Na sličan način, kada kreirate niz korišćenjem new <class>[size], on se briše i oslobada se njegova memorija, uz pomoć naredbe delete[]. Srednje zagrade sugeriraju kompajleru da se niz briše.

Ako izostavite srednje zagrade, samo prvi objekat niza će biti obrisan. Ovo možete proveriti tako što ćete ukloniti srednje zagrade u liniji **40.** Ako ste editovali liniju 21, tako da dekstruktor vrši prikaz, sada ćete videti da je samo jedan CAT objekat uništen.

Ironija! Upravo ste doveli do, "curenja" memorije.

4| **PAZin** Zapamtite da je niz od n elemenata numerisan od 0 do n-1.

Nemojte pisati, ili čitati iza kraja niza.

Nemojte mešati niz pointera sa pointerom na niz.

Koristite indeksiranje niza sa pointerima koji ukazuju na niz.

char nizovi

String je serija karaktera. Jedini stringovi koje ste do sada videli su bile bezimene string konstante, korišćene u cout naredbama, kao na primer,

```
cout « "hello world.\n";
```

U C++ -u string je niz, koji se sastoji od char-ova i završava se null karakterom. String možete deklarisati i inicijalizovati baš kao i bilo koji drugi niz. Na primer,

```
char Greeting!] =
{ 'H', 'e', Ч\ 'Г, V, ' \ 'WVoVrVT.'d', '\0' };
```

Poslednji karakter, "\0" je null karakter koji vecina C++ funkcija prepoznaje kao terminator stringa. Iako ovaj karakter-po-karakter pristup funkcioniše, teško je izbeći mnogo mogućnosti za grešku. C++ Vam omogućava korišćenje prečice za prethodnu liniju koda i ona je

```
char Greeting[] = "Hello World";
```

Možete primetiti dve karakteristike bitne za ovu sintaksu:

- umesto jednostrukih karaktera pod navodnicima, koji su odvojeni zapetama i okruženi vitičastim zagradama, sada imate string pod duplim navodnicima, bez zapeta i vitičastih zagrada;
- nije potrebno da dodajete null karakter, pošto će kompajler to učiniti za Vas.

String Hel 1 o Worl d je dugačak 12 bajtova. Hel 1 o je 5 bajtova, razmak je 1, Worl d je 5 i null karakter je 1.

Takođe je moguće kreirati neinicijalizovani niz karaktera. Kao i kod svih drugih nizova važno je proveriti da li stavljate vise karaktera u bafer, nego što za njih ima mesta.

U listingu 11.8 demonstrira se korišćenje neinicijalizovanog bafera.

Listing 11.8: Popunjovanje niza.

```
1       //Listing 11.8 baferi za niz karaktera
2
3       linclude <iostream.h>
4
5       int main()
6       {
7          char buffer[80];
8          cout « "Enter the string: *
9          cin » buffer;
10         cout « "Here's the buffer:    « buffer « endl;
11       return 0;
12      }
```

```
Enter the string: Hello World
Here's the buffer:  Hello
```

U liniji 7 deklarisan je bafer koji može da sadrži 80 karaktera. On je dovoljno veliki da prihvati 79 karaktera dugačak string i null karakter za terminaciju.

U liniji 8 od korisnika je zatraženo da unese string koji je prenet u bafer u liniji 9. Ovo je sintaksa naredbe `cin`, koja upisuje null karakter u bafer, pošto upiše string.

Postoje dva problema sa programom iz listinga 11.8. Prvi je kada korisnik unese vise od 79 karakter a- ci n će upisati iza kraja bafera. Drugi je kada korisnik unese prazno mesto - ci n će to shvatiti kao kraj stringa i prestaće da upisuje u bafer.

Da biste rešili ove probleme, morate pozvati specijalni metodi `cin:get()`, koji uzima tri parametra:

- bafer koji treba popuniti
- maksimalni broj karaktera koji treba preuzeti
- delimiter koji terminira ulaz.

Podrazumevani delimiter je newline. U listingu 11.9 ilustrovano je njegovo konšćenje.

Listing 11.9. Popunjgvanje niza.

```
//Listing 11.9 korišćenje cin.get()

#include <iostream.h>

int main()
```


I

```
6:      {
7:         char buffer[80];
8:         cout « "Enter the string:  ";
9:         cin.get(buffer, 79);        // uzima do 79 ili novog reda
10:        cout « "Here's the buffer:    " « buffer « endl;
11:     return 0;
12:     }
```

```
Enter the string: Hello world
'Here's the buffer:  Hello World
```

'fTTITO^hfc Linije 9 poziva cin-ov get() metod. Bafer deklarisan u liniji 7 je pro-
^ sleden kao prvi argument. Drugi argument je maksimalni broj karaktera koji treba preuzeti. U ovom slučaju on mora da bude 79, kako bi ostalo prostora za null karakter. Nije potrebno obezbediti karakter za terminaciju, pošto je podrazumevana vrednost newline.

Cin i sve njegove varijacije će biti opisani u Danu 17, "Pretprocesor", kada se detaljno bude diskutovalo o stremovima.

strcpyO i strncpy()

C++ j_e od C-a nasledio biblioteku funkcija, koje služe za rad sa stringovima. Medu mnogima od njih postoje dve za kopiranje jednog stringa u drugi: **strcpyO** i **strncpyO, strcpyO** kopira kompletan sadržaj jednog stringa u željeni bafer, a u listingu 11.10 je demonstrirano korišćenje te funkcije.

Listing 11.10: Korišćenje strcpyO.

```
1:      linclude <iostream.h>
2:      #include <string.h>
3:      int main()
4:      {
5:         char StringI[] = "No man is an island";
6:         char String2[80];
7:
8:         strcpy(String2,StringI);
9:
10:        cout « "StringI:  " « StringI « endl;
11:        cout « "String2:  " « String2 « endl;
12:     return 0;
13:     }
```

```
StringI: No man is an island
String2: No man is an island
```

ifMte, Datoteka zaglavlja string.h je uključena u liniji 2. Ova datoteka sadrži
' "" " prototip **strcpyO** funkcije, koja uzima dva niza karaktera - odredište koje sledi izvor. Ako je izvor veći od odredišta, **strcpyO** će pisati iza kraja bafera.

Da biste se zaštitili od ovoga, standardna biblioteka takode sadrži strncpy(). Ova varijacija preuzima maksimalni broj karaktera koji treba kopirati; vršiće kopiranje, dok ne naide na prvi null karakter, ili na maksimalni broj karaktera, koji je specificiran.

Listing 11.11 ilustruje korišćenje strncpy()

Listing 11.11: Korišćenje `strncpy()`.

```
#include <iostream.h>
linclude <string.h>
int  main()
{
   const int MaxLength = 80;
   char StringI[ ] = "No man is an island"
   char String2[MaxLength+I];


   strncpy(String2,StringI,MaxLength);

   cout «   "StringI:   «  StringI  «  endl;
   cout «   "String2:   «  String2  «  endl;
 return 0;
}
```

```
JEEEjj^ StringI: No man is an island
        String2: No man is an island
```

U liniji 10 poziv **strcpyO** je zamenjen pozivom funkcije strncpy(), koja zahteva i treći parametar: maksimalni broj karaktera za kopiranje. Bafer string2 je deklarisan da može da primi MaxLenght+1 karakter. Ekstra karakter je za null koji će **strcpyO** i **strncpy()** automatski dodati na kraj stringa.

String klase

Većina C++ kompajlera se isporučuje sa bibliotekom klase, koja uključuje veliki set klasa za manipulaciju podacima. Standardna komponenta **class** biblioteke je string klasa.

C++ je nasledio null terminirane stringove i biblioteku funkcija koja uključuje **strcpyO** iz C-a, ali ove funkcije nisu integrisane u objektno-orijentisani okvir. String klasa obezbeduje enkapsulirani set podataka i funkcija za manipulaciju tim podacima, kao i pristupne funkcije, tako da su sami podaci sakriveni od klijenata string klase.

Ako Vaš kompajler ne podržava string klasu, a ako možda čak i podržava, možete poželeti da napišete sopstvenu. U preostalom delu ovog poglavlja diskutovaćemo o dizajnu i parcijalnoj implementaciji string klasa.

Kao minimum, string klasa mora da reši osnovna ograničenja nizova karaktera. Kao i svi drugi, i ovi nizovi su statički. Vi definišete koliko će oni biti veliki. Uvek uzimaju toliko prostora u memoriji, čak i ako Vam to nije potrebno. Pisanjem iza kraja niza dovodite Vaš program do katastrofe.

Dobra string klasa alocira samo onoliko memorije koliko je neophodno i uvek je ima dovoljno, kako bi prihvatila ono što joj se daje. Ako nije u mogućnosti da alocira dovoljno memorije, trebalo bi da korektno završi sa radom.

U listingu 11.12 obezbedena je prva aproksimacija string klase.

Listing 11.12: Korišćenje klase `String`.

```
//Listing  11.12

linclude <iostream.h>
linclude <string.h>

// Osnovna klasa string
class String
{
   public:
      //  konstruktori
      StringO;
      String(const char *const);
      String(const String &);
      -String();

      //  preklopljeni  operatori
      char & operator[](unsigned short offset);
      char operator[] (unsigned short offset)  const;
      String operator+(const String&);
      void operator+=(const String&);
      String & operator= (const String &);

      // 0p{te metode pristupa
      unsigned short GetLen()const { return itsLen; }
      const char * GetStringO const { return itsString; }

   private:
      String (unsigned short);         //  privatni  konstruktor
      char * itsString;
      unsigned short itsLen;
};

// podrazumevani  konstruktor kreira string od 0 bajtova
String::String()
{
   ItsString = new char[1];
   itsString[0] = '\0';
   itsLen=0;
}
```

nastavlja se

Listing 11.12: Korišćenje klase String.

```
40:
41:    // privatni (pomo}ni) konstruktor, koji se koristi samo od strane
42:    // metoda klase za kreiranje novog stringa
43:    // tražene veli-ine. Puni se sa null.
44:    String::String(unsigned short len)
45:    {
46:       itsString = new char[len+l);
47:       for (unsigned short i = 0; i<=len; i++)
48:          itsStringfi] = '\0';
49:       itsLen=len;
50:    }
51:
52:    // Konvertuje niz karaktera u String
53:    String::String(const char * const cString)
54:    {
55:       itsLen = strien(cString);
56:       itsString = new char[itsLen+1];
57:       for (unsigned short i = 0; i<itsLen; i++)
58:          itsString[i] = cStringfi];
59:       itsString[itsLen]='\0';
60:    }
61:
62:    // konstruktor kopije
63:    String: .-String (const String & rhs)
64:    {
65:       itsLen=rhs.GetLen();
66:       itsString = new char[itsLen+1];
67:       for (unsigned short i = 0; i<itsLen;i++)
68:          itsString[i] = rhs[i];
69:       itsString[itsLen] = '\0';
70:    }
71:
72:    // destruktor, oslobada alociranu memoriju
73:    String: :-String ()
74:    {
75:       delete [] itsString;
76:       itsLen = 0;
77:    }
78:
79:    // operator jednakosti, oslobađa postoje}u memoriju
80:    // zatim kopira string i veličinu
81:    Strings String::operator=(const String & rhs)
82:    {
83:       if (this == &rhs)
84:          return *this;
85:       delete [ ] itsString;
86:       itsLen=rhs.GetLen();
87:       itsString = new char[itsLen+1] ;
```



```
88:       for (unsigned short i = 0; i<itsLen;i++)
89:          itsString[i] = rhs[i];
90:       itsString[itsLen] = '\0';
91:       return *this;
92:    }
93:
94:    // operator nekonstantnog pomaka, vra}a
95:    // referencu na karakter tako da se on može
96:    // promeniti!
97:    char & String::operator[](unsigned short offset)
98:    {
99:       if (offset > itsLen)
100:         return itsString[itsLen-IJ;
101:      else
102:         return itsString[offset];
103:   }
104:
105:   // operator konstantnog pomaka za upotrebu
106:   // sa konstantnim objektima (pogledati konstruktor kopije!)
107:   char String::operator[](unsigned short offset) const
108:   {
109:      if (offset > itsLen)
110:         return itsStringfitsLen-1];
111:      el se
112:         return itsString[offset];
113:   }
114:
115:   // kreira novi string dodaju}i teku}i
116:   // string referenci rhs
117:   String String::operator+(const String& rhs)
118:   {
119:      unsigned short totalLen = itsLen + rhs.GetLen();
120:      String temp(totalLen);
121:      for (unsigned short i = 0; i<itsLen; i++)
122:         temp[i ] = itsString[i ];
123:      for (unsigned short j = 0; j<rhs.GetLen(); j++, i++)
124:         temp[i ] = rhs[j ];
125:      temp[totalLen]='\0';
126:      return temp;
127:   }
128:
129:   // menja teku}i string, ne vra}a ni{ta
130:   void String::operator+=(const String& rhs)
131:   {
132:      unsigned short rhsLen = rhs.GetLen();
133:      unsigned short totalLen = itsLen + rhsLen;
134:      String temp(totalLen);
135:      for (unsigned short i = 0; i<itsLen; i++)
136:         temp[i ] = itsString[i ];
137:      for (unsigned short j = 0; j<rhs.GetLen(); j++, i++)
```

nastavlja se

Listing 11.12: Korišćenje klase String.

nastavak

```
138:          tempfi] = rhs[i-itsLen];
139:        temp[totalLen]='\0';
140:        *this = temp;
141:    }
142:
143:    int main()
144:    {
145:       String si(-Initial test'*);
146:       cout « "SI:\t" « si.GetStringQ « endl;
147:
148:       char * temp = "Hello World";
149:       si = temp;
150:       cout « "SI:\t" « si.GetStringQ « endl;
151:
152:       char tempTwo[20];
153:       strcpy(tempTwo,"; nice to be here!");
154:       si += tempTwo;
155        cout « "tempTwo:\t" « tempTwo « endl;
156        cout « "SI:\t" « si.GetStringQ « endl;
157
158        cout « "SI[4]:\t" « sI[4] « endl;
159:     sts[l4]}= Vx;';
160:
161:       cout « "SI:\t" « si.GetStringQ « endl;
162:
163:       cout « "SI[999]:\t" « si[999] « endl;
164:       String s2(" Another string");
165:       String s3;
166:       s3 = sI+s2;
167:       cout « "S3:\t" « s3.GetStringQ « endl
168:
169:       String s4;
170:       s4 = "Why does this work?";
171:       cout « "S4:\t" « s4.GetStringQ « endl;
172:     return 0;
173:  }
```

```
SI:        initial test
SI:        Hello world
tempTwo:          ; nice to be here!
SI-'       Hello world; nice to be here'
SI[4]:     o
SI:        Helix World; nice to be here!
SI[999]:
S3:        Helix World; nice to be here! Another string
S4:        Why does this work?
```


*"ЦШ№[]^* Linije 7-31 su deklaracija jednostavne string klase. Linije 11-13 sadrže tri konstruktora: podrazumevani konstruktor, copy konstruktor i konstruktor koji uzima postojeći null terminirani sting u C-stilu.

String klasa vrši overload ofset operatora ([ ]), operatora plus (+) i operatora plus-jednako (+=). Overload ofset operatora je uraden dvaput: jednom kao konstantna funkcija, koja vraća char, i drugi put kao nekonstantna funkcija, koja vraća referencu na char.

Nekonstantna verzija se koristi u naredbama nalik na

```
SomeString[4]='x';
```

kao što se vidi i u liniji 159. Ovim Vam je obezbeden direktan pristup bilo kojem karakteru u stringu. Referenca na karakter je vraćena, tako da pozivajuća funkcija može njime manipulisati.

Konstantna verzija se koristi kada se pristupa konstantnom string objektu, kao na primer, u implementaciji copy konstruktora u liniji 63. Primetite da se pristupa rhs[i], iako je rhs deklarisan kao const string &. Neispravno je pristupati ovom objektu, korišćenjem nekonstantne funkcije člana. Stoga se mora izvršiti overload operatora konstantnom pristupnom funkcijom.

Ako je vraćeni objekat veliki, možete poželeti da deklarišete da vraćena vrednost bude konstantna referenca. Međutim, pošto je char dugačak samo jedan bajt, nema smisla ovako nešto raditi.

Podrazumevani konstruktor je implementiran u linijama 33 do 39. On kreira string dužine nula. Konvencija za ovu string klasu je da prijavljuje svoju dužinu, ne računajući null karakter. Ovaj podrazumevani string sadrži samo null karakter.

Copy konstruktor je implementiran u linijama 63-70. On postavlja novu dužinu stringa na onu od postojećeg stringa +1 za null karakter. Kopira sve karaktere postojećeg stringa u novi string i zatim null karakterom terminira novi string.

Linije 53-60 implementiraju konstruktor koji uzima postojeći string C-stila. Ovaj konstruktor je sličan copy konstruktoru. Dužina postojećeg stringa se ustanovljava pozivom standardne bibliotečke funkcije stri **en** () iz string biblioteke.

U liniji **28** deklarisan je još jedan konstruktor string **(unsigned short),** kao private funkcija član. Namera dizajnera ove klase je da klijent klase ne može da kreira string proizvoljne dužine. Ovaj konstruktor postoji samo da bi pomogao u internom kreiranju stringa, na primer, uz pomoć operatora +=, u liniji 130. Ovo će detaljnije biti opisano kada bude prikazan operator +=.

String **(unsigned short)** konstruktor popunjava svaki clan niza sa null. Stoga, **for** petlja proverava za **i<=len** umesto **i** <1 en.

Destruktor koji je implementiran u linijama 73-77 briše karakter string klase. Proverite da li ste uključili uglaste zagrade u poziv operatora za brisanje, kako bi se obrisao svaki clan niza, umesto samo prvog.

Operator za dodeljivanje prvo proverava da li je desna strana dodeljivanja ista kao i leva strana. Ako nije, tekući string će biti obrisan, a novi će biti kreiran i kopiran na mesto. Referenca je vraćena da bi omogućila sledeća dodeljivanja.

```
StringI = String2 = String3;
```

Dva puta je izvršen overload ofset operatora. Osnovne provere su izvršene oba puta. Ako korisnik pokuša da pristupi karakteru na lokaciji koja je iza kraja niza, poslednji karakter koji je len-1 će biti vraćen.

U linijama 117-127 implementiran je operator plus (+) kao operator za konkatenaciju. Zgodno je da imate mogućnost da napišete

```
String3 = StringI + String2;
```

i da dobijete sting 3 koji je konkatenacija druga dva stringa. Da biste ovo izvršili, funkcija operatora plus izračunava kobinovanu dužinu dva stringa i kreira privremeni string temp. Ovo poziva private konstruktor, koji uzima integer i kreira sting popunjen null-ovima. Null-ovi će, zatim, biti zamenjeni sadržajem dva stringa. String sa leve strane (*this) će se kopirati prvi, a slediće ga string sa desne strane (rhs).

Prva for petlja prebrojava string sa leve strane i dodaje svaki njegov karakter u novi string. Druga for petlja se kreće kroz sting sa desne strane. Primetiće da i nastavlja da broji mesta u novom stringu, dok j prebrojava rhs string. Operator plus (+) vraća string temp po vrednosti koja je dodeljena stringu sa leve strane dodeljivanja (stringl). Operator plus-jednako (+=) operiše postojećim stingom, tj. sa stringom koji se nalazi sa leve strane naredbe stringl+=sting2. On radi baš kao i operator plus, osim što je temp vrednost dodeljena tekućem stringu (*this=temp) u liniji 140.

Funkcija main() (linije 143-173) se ponaša kao test drajver program za ovu klasu. Linija 145 kreira string objekat, korišćenjem konstruktora koji uzima null terminiran C-style sting. Linija 146 prikazuje njegov sadržaj, uz pomoć pristupne funkcije GetString(). Linija 148 kreira još jedan C-style string. Linija 149 testira operator dodeljivanja, a linija 150 prikazuje rezultat.

Linija 152 kreira treći C-style string tempTwo. Linija 153 poziva strcpy, da bi popunila bafer sa karakterima ; nice to be here. Linija 154 poziva operator plus-jednako i vrši konkatenaciju postojećeg stringa si i tempTwo. Linija 156 prikazuje rezultat. U liniji 158 četvrti karakter iz si je prikazan. On je dobio novu vrednost u liniji 159. Ovo se izvodi pozivanjem nekonstantnog ofset operatora ([ ]). Linija 160 prikazuje rezultat koji dokazuje da je stvarna vrednost promenjena.

Linija 162 pokušava da pristupi karakteru iza kraja niza. Poslednji karakter niza je vraćen, kao što se i očekivalo.

Linije 164-165 kreiraju dva nova string objekta, a linija 166 poziva operator za sabiranje. Linija 167 prikazuje rezultat.

Linija 169 kreira novi string objekat s4. Linija 170 poziva operator za dodeljivanje, linija 171 prikazuje rezultat. Možda ste pomislili da je operator za dodeljivanje definisan da preuzme konstantnu string referencu u liniji 21, ali ovde program prosleduje string C-stila. Da li je ovo legalno?

Odgovor je da kompajler očekuje string, ali dobija niz karaktera. Stoga, on proverava da li može da kreira string iz toga što mu je dato. U liniji 12 deklarisali ste konstruktor, koji kreira string iz niza karaktera. Kompajler kreira privremeni string iz niza karaktera, i prosleduje ga operatoru dodeljivanja. Ovo se naziva promocija. Ako niste deklarisali i obezbedili implementaciju za konstruktor koji uzima niz karaktera, dodeljivanje će dovesti do greške u kompilaciji.

Povezane liste i druge strukture

Nizovi su odlični kontejneri, ali su fiksne veličine. Ako izaberete kontejner koji je preveliki, "bacićete" prostor. Ako izaberete premali, njegov sadržaj će se prelivati svuda unaokolo i imaćete pravi haos.

Jedan od načina da rešite ovaj problem je uz pomoć povezanih lista. Povezane liste su struktura podataka, koja se sastoji od malih kontejnera dizajniranih da svuda mogu da se smeste i da mogu medusobno da se povežu, kada je potrebno. Osnovna ideja je da napišete klasu koja sadrži jedan objekat Vaših podataka, kao što je jedna mačka, ili jedan pravougaonik, i da, zatim ukažete na sledeći kontejner. Kreiraćete po jedan kontejner za svaki objekat, koji Vam je neophodan, i međusobno ćete ih povezati kada je to potrebno.

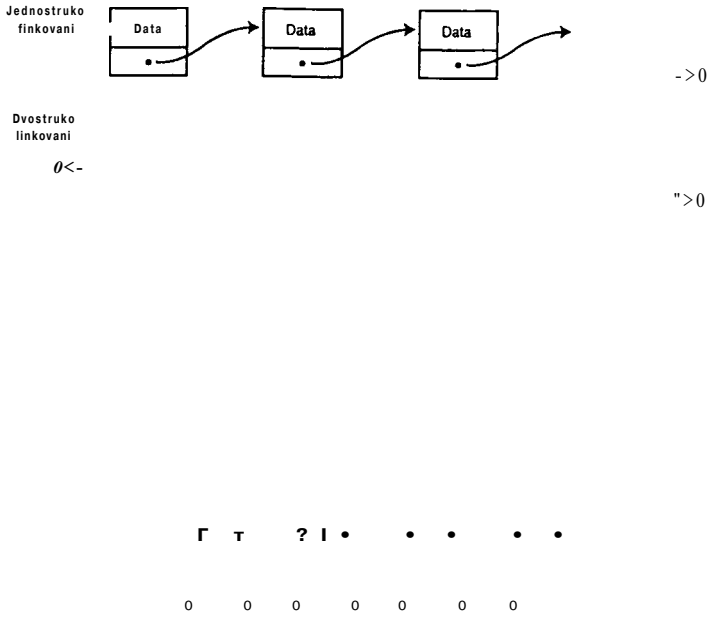
Kontejneri se nazivaju i čvorovi. Prvi čvor u listi se naziva glava, dok se poslednji naziva rep.

Liste dolaze u tri fundamentalne forme - od jednostavnih prema složenijima i to su:

- jednostruko povezane
- dvostruko povezane
- stable

U jednostruko povezanim listama svaki čvor ukazuje unapred na sledeći, ali ne i na prethodni. Da biste pronašli odredeni Čvor, počinjete ispočetka i idete od čvora do čvora, kao u "lovu" na blago. Dvostruko povezane liste Vam omogućavaju da se krećete i unapred i unazad unutar lanca. Stablo je složena struktura sastavljena od čvorova, od kojih svaki može ukazivati na dva, ili tri pravca. U slici 11.5 prikazane su ove tri osnovne strukture.

Naučnici u računarskoj tehnologiji su kreirali kompleksnije i pametnije strukture podataka, koje se, praktično, sve sastoje od medusobno povezanih čvorova. U listingu 11.13 prikazano je kako da kreirate i koristite jednostavnu povezanu listu.



Slika 11.5

Povezane liste

Listing 11.13: Implementiran je povezane liste.

```
1:  // Listing 11.13
2:  // Jednostavna implementacija povezane liste
3:
4:  #include <iostream.h>
5:
6:  // objekat koji }e se dodati listi
7:  class CAT
8:  {
9:  public:
10:    CAT() { itsAge = 1;}
11:    CAT(int age):itsAge(age){}
12:    ~CAT(){};
13:    int GetAgeO const { return itsAge; }
14: private:
15:    int itsAge;
16: };
17:
18:
19: // upravlja listom, ureduje je prema godi{tu mačke!
    class Node
20:  {
21:  public:
22:     Node (CAT*);
23:     -Node();
24:     void SetNext(Node * node) { itsNext = node; }
25:     Node * GetNextO const { return itsNext; }
26:     CAT * GetCat() const { return itsCat; }
27:     void Insert(Node *);
28:     void DisplayQ;
29:  private:
30:     CAT *itsCat;
31:     Node * itsNext;
32:  };
33:
34:
35:  Node::Node(CAT* pCat):
36:  itsCat(pCat),
37:  itsNext(O)
38:  {}
39:
40:  Node::~Node()
41:  {
42:     cout « "Deleting node...\n";
43:     delete itsCat;
44:     itsCat = 0;
45:     delete itsNext;
46:     itsNext = 0;
47:  }
48:
49:  ii  **********
50:  //  Insert
51:  // Uređuje mačke baziraju}i se na njihovom godi{tu
52:  // Algoritam: Ako ste poslednji u liniji, mačka se dodaje
53:  // Inače, ako je nova mačka starija od vas
54:  // i mlada od sledece u liniji, unosi se posle ove.
55:  // Inače, poziva se Insert za slede}u u liniji
56.  // **********
57:  void Node::Insert(Node* newNode)
58:  {
59:     if (MtsNext)
60:        itsNext = newNode;
61:     else
62:     {
63:        int NextCatsAge = itsNext->GetCat()->GetAge();
64:        int NewAge = newNode->GetCat()->GetAge();
65:        int ThisNodeAge = itsCat->GetAge();
66:
67:        if ( NewAge >= ThisNodeAge && NewAge < NextCatsAge )
68:        {
```

nastavlja se

Listing 11.13: Implementiranje povezane liste.

```
69              newNode->SetNext(i tsNext);
70              itsNext « newNode;
71           }
72           else
73              itsNext->Insert(newNode);
74
75     }
76
77     void Node::Display()
78     {
79        if (itsCat->GetAge() > 0)
80        {
81           cout « "My cat is ";
82           cout « itsCat->GetAge() « " years old\n"
83        }
84        if (itsNext)
85           itsNext->Display();
86     }
87
88     int main()
89     {
90
91;       Node *pNode = 0;
92        CAT * pCat = new CAT(0);
93        int age;
94
95        Node *pHead = new Node(pCat);
96
97        while (1)
98        {
99           cout « "New Cat's age? (0 to quit): ";
100           cin » age;
101           if (!age)
102              break;
103          pCat = new CAT(age);
104          pNode = new Node(pCat);
105          pHead->Insert(pNode);
106       }
107       pHead->Display();
108       delete pHead;
109       cout « "Exiting...\n\n";
110    return 0;
111

        New Cat's age? (0 to quit): 1
        New Cat's age? (0 to quit): 9
        New Cat's age? (0 to quit): 3
        New Cat's age? (0 to quit): 7
```


nastavak

```
New Cat's age? (0 to quit): 2
New Cat's age? (0 to quit): 5
New Cat's age? (0 to quit): 0
My cat is 1 years old
My cat is 2 years old
My cat is 3 years old
My cat is 5 years old
My cat is 7 years old
My cat is 9 years old
Deleting node...
Deleting node...
Deleting node...
Deleting node...
Deleting node...
Deleting node...
Deleting node...
Exiting...
```

II'M^jjplfc Linije 7-16 deklarišu pojednostavljenu CAT klasu. Ona ima dva konstruktora: podrazumevani konstruktor, koji inicijalizuje promenljivu člana itsAge na 1, i konstruktor koji uzima integer i inicijalizuje itsAge na tu vrednost.

Linije 19-32 deklarišu klasu Node. Node je specijalno dizajnirana da sadrži CAT objekte u listi. Naravno, Vi ćete sakriti Node unutar CatList klase. Ona ovde nije sakrivena, kako bi se ilustrovao rad povezanih lista.

Moguće je napraviti generičku Node klasu, koja bi sadržala bilo koju vrstu objekata u listi. Ovo ćete naučiti u Danul4, "Specijalne klase i funkcije", kada se bude diskutovalo o templejtima. Node konstruktor uzima pointer na CAT objekat. Copy konstruktor i operator dodeljivanja su izostavljeni, da bismo sačuvali prostor. U stvarnim aplikacijama oni bi morali da budu uključeni.

Definisane su tri pristupne funkcije. SetNextO postavlja promenljivu člana itsNext da ukazuje na Node objekat, koji je obezbeden kao njegov parametar. Get Next () i GetCat() vraćaju odgovarajuće promenljive članove. GetNextO i GetCat() su deklarisane kao const, pošto one ne menjaju Node objekat.

Insert() je najjača funkcija clan klase. Insert() održava povezanu listu i dodaje Nodeove u listu na osnovu godina CAT, a na koji ukazuje.

Program počinje od linije 88. Pointer pNode je kreiran i inicijalizovan na nulu. Takode je kreiran lažni objekat CAT i njegove godine su inicijalizovane na nulu, kako bismo se obezbedili da pointer na glavu liste (phead) uvek bude prvi.

Počevši od linije 99, korisnik je upitan za godine. Ako korisnik pritisne nulu, to će biti signal da treba prestati sa kreiranjem CAT objekata. Za sve druge vrednosti u liniji 103 se kreira CAT objekat i promenljiva clan itsAge se postavlja na unetu vrednost. CAT objekti se kreiraju u slobodnom prostoru. Za svaki kreirani CAT kreira se Node objekat u liniji 104.

Pošto su CAT i Node kreirani objekti, prvom Nodeu iz liste je rečeno da insertuje novokreirani čvor u liniji 105.

Primetićete da program niti zna, niti ga je briga kako će Node biti insertovan, ili kako se lista održava. To je, u celini,, stvar" samog Node objekta.

Poziv Insert () prouzrokuje da izvršenje programa "skoči" na liniju 57. Insert () se uvek poziva kada je pHead prvi.

Test u liniji 59 neće uspeti prvi put i dodaće se novi Node. Stoga, pHead ukazuje na prvi novi Node. U izlazu to je čvor čija je i tsAgeVal ue bila postavljena na 1.

Kada je promenljiva itsAge drugog CAT objekta postavljena na 9, ponovo je pozvan pHead. Ovoga puta njegova promenljiva član i tsNext nije nul 1 i el se naredba u linijama 61-74 je izvršena.

Tri lokalne promenljive, NextCatAge, NewAge i ThisNodeAge, su popunjene vrednostima

- godine tekućeg Nodea - godine pHead Cata, tj. 0.
- godine CAT-a koji sadrži novi Node - u ovom slučaju 9.
- godine CAT objekta koji sadrži sledeći Node u redosledu - u ovom slučaju 1.

Test u liniji 67 trebalo bi da bude napisan na sledeći način

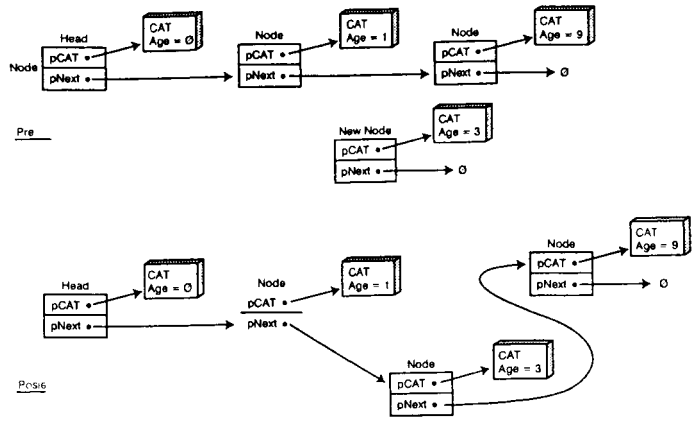
```
if ( newNode->GetCat()->GetAge() > itsCat->GetAge() && \\
newNode->GetCat()->GetAge()< itsNext->GetCat()->GetAge())
```

što bi dovelo do eliminacije tri privremene promenljive, ali bi kreiralo i kod, koji je zbunjujući i težak za čitanje. Neki C + + programeri ovo vide kao "mačo" stvar, sve dok ne naidu na bag i tada shvate da ni na koji način ne mogu da pronadu koja je vrednost neispravna.

Ako je nova godina CAT-a veća od tekuće i manja od sledeće, odgovarajuće mesto za insertovanje nove godine je odmah iza tekućeg Nodea. U torn slučaju, i f naredba vraća True. Novi Node je postavljen da ukazuje na ono na šta tekući Node ukazuje, a tekući Node je postavljen da ukazuje na novi Node. Ovo je ilustrovano na slici 11.6.

Ako test ne uspe, to nije odgovarajuće mesto za insertovanje Nodea i Insert() će biti pozvana iz sledećeg Nodea u listi. Primetite da se tekući poziv Insert () ne vraća, sve dok se rekurzivni poziv Insert () ne vrati. Stoga, ovi pozivi prepunjavaju stek. Ako lista postane predugačka, to može dovesti do pada programa. Postoje drugi načini koji ne barataju toliko sa stekom, ali su oni izvan polja interesovanja ove knjige.

Kada korisnik završi sa dodavanjem CAT objekata, prikaz je pozvan od prvog Node:pHead. Godine CAT objekata su prikazane, ako tekući Node uklazuje na CAT (pHead to ne čini); ako tekući Node ukazuje na neki drugi Node, Display() je pozvan za taj Node.



Slika 11.6
Insertovanje Node-a

Na kraju, pozvano je brisanje pHead. Pošto destruktor briše pointer za sledeći Node, brisanje će biti pozvano i za taj Node. Ono "seta" kroz celu listu, elieminišući svaki Node i oslobađajući memoriju za itsCat. Primetićete da poslednji Node ima promenljivu člana itsNext, postavljenu na nulu, i brisanje će biti pozvano i za taj pointer. Uvek je sigurno pozvati brisanje za nulu, s obzirom da ono nema efekte.

Nizovi klasa

Pisanje Vaše Array klase ima vise prednosti nad korišćenjem ugradenih nizova. Za početak, možete sprečiti prekoračenja niza. Takođe, možete razmisliti o tome da Vaša niz klasa ima dinamičku veličinu: u trenutku kreiranja, ona može imati samo jedan član i, zatim, rasti kako to zahtevaju potrebe programa.

Možete poželeti da sortirate, ili na neki drugi način složite članove niza. Postoji veliki broj snažnih Array varijanti, o kojima možete razmisliti. Najpopularnije su:

- složene kolekcije: svi članovi su u sortiranom redosledu;
- set: nijedan član se ne pojavljuje vise od jednom;
- sadržaj: ovde se koriste odgovarajući parovi, kod kojih se jedna vrednost ponaša kao ključ za dobijanje druge vrednosti;
- veliki nizovi: namenjena je za velike setove, ali samo one vrednosti koje su dodate u niz zauzimaju memoriju. Stoga, Vi možete tražiti SparseArray[5], ili SparseArray [200], ali je moguće da je ta memorija alocirana samo za manji broj ulaza;
- torba: nesortirana kolekcija, koja se dodaje i iščitava slučajnim redosledom.

Kada izvršite overload index operatora ([ ]), povezana lista se može pretvoriti u složenu kolekciju. Isključujući duplikate, ovu kolekciju možete pretvoriti u set. Ako svaki objekat liste ima par sa odgovarajućom vrednošću, povezanu listu možete koristiti za sadržaje ili, za velike nizove.

Rezime

Danas ste naučili kako se u C-f+-u kreiraju nizovi. Niz je kolekcija objekata fiksne veličine, koji su svi istog tipa.

Nizovi ne proveravaju svoje okvire. Stoga, sasvim je legalno, iako je katastrofalno, čitati, ili pisati iza kraja niza. Nizovi se broje od **0.** Uobičajena greška je pisanje na ofset n u nizu od n članova.

Nizovi mogu biti jednodimenzionalni, ili višedimenzionalni. U oba slučaja, članovi niza mogu biti inicijalizovani dokle god niz sadrži ugradene tipove, kao što je `int`, ili objekte klase, koji imaju podrazumevani konstruktor.

Nizovi i njihov sadržaj mogu biti u slobodnom prostoru, ili na steku. Ako obrišete niz u slobodnom prostoru, morate koristiti srednje zagrade, kada pozivate `Delete`.

Imena nizova su konstantni pointer na prvi element niza. Pointeri i nizovi koriste aritmetiku pointera za pronalaženje sledećeg elementa niza.

Možete kreirati povezane liste za upravljanje kolekcijama, čiju veličinu nećete znati u trenutku kompilacije. Iz povezanih lista možete kreirati bilo koji broj složenijih struktura podataka.

Stringovi su nizovi karaktera. C++ obezbeduje specijalne osobine za upravljanje char nizovima, uključujući i mogućnost za njihovu inicijalizaciju, uz pomoć stringova pod navodnicima.

Pitanja i odgovori

P Sta će se dogoditi ako upišem element 25 u 24-članom nizu?

0 Pisaćete u drugu memoriju, sa potencijalnim katastrofalnim efektima po Vaš program.

P Sta je u neinicijalizovanom elementu niza?

O Bilo šta što se dogodilo u u memoriji datog trenutka. Rezultat korišćenja ovog člana, bez dodeljivanja vrednosti, je nepredvidiv.

P Da li mogu kombinovati nizove?

O Da. Sa jednostavnim nizovima možete koristiti pointere, da biste ih iskombinovali u nove, veće nizove. Sa stringovima možete koristiti neke od ugradenih funkcija, kao na primer, strcat, za kombinaciju stringova.

P Zašto da kreiram povezane list kada će i nizovi funkcionisati?

O Nizovi moraju da imaju fiksnu veličinu, dok povezane liste veličinu mogu menjati dinamički, **u** fazi izvršenja.

P Zašto bih koristio ugrađene nizove, kada mogu da napravim bolje nizove klase?

O Ugradeni nizovi su brzi i jednostavni za korišćenje.

P Da li string klasa mora da koristi char* da bi sadržala sadržaj stringa?

O Ne. Ona može koristiti ono što za šta dizajner misli da je najbolje.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje obrađene teme i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i proverite da li razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Koji su prvi i poslednji element u SomeArray [25] ?
2. Kako ćete deklarisati višedimenzionalni niz?
3. Inicijalizujte članove niza iz pitanja 2?
4. Koliko elemenata sadrži niz SomeArray [10], [5] i [20].
5. Koji je maksimalni broj elemenata koji se može dodati u povezanu listu?
6. Može li se koristiti subscript notacija u radu sa povezanim listama?
7. Koji je poslednji karakter u stringu "Brad is a nice guy?"

Vežbe

1. Deklarišite dvodimenzionalni niz, koji predstavlja "iks-oks" tablu za igru.
2. Napišite kod koji inicijalizuje sve elemente niza kreirane u Vežbi 1 na vrednost 0.
3. Napišite deklaraciju za Node klasu, koja sadrži unsigned short integers.
4. **ISTERIVAČ BAGOVA:** Šta nije u redu sa sledećim delom koda:

```
unsigned short SomeArray[5][4];
for (int i = 0; i<4; i++)
    for (int j = 0; j<5; j++)
        SomeArray[ i][j}  =  i+j;
```

5. **ISTERIVAČ BAGOVA:** Šta nije u redu sa sledećim delom koda:

```
unsigned short SomeArray[5][4];
for (int i • 0; i<=5; i++)
     for (int j « 0; j<=4; j++)s
          SomeArray[i][j]  = 0;
```


Dan 12

Nasledivanje

Fundamentals aspekt ljudske inteligencije je da pronalazi, prepoznaje i kreira relacije izmedu koncepata. Mi gradimo hijerarhije, mreže, matrice i druge medusobne relacije, kako bismo objasnili i razumeli načine na koje stvari vrše uticaj jedne na druge. C+ + pokušava da ovo obuhvati, uz pomoć hijerarhije nasledivanja. Danas ćete učiti:

- šta je nasledivanje
- kako se jedna klasa izvodi iz druge
- šta su zaštićeni pristupi i kako se koriste
- šta su virtuelne funkcije.

v

Sta je nasledivanje?

Šta je pas? Kada pogledate svog ljubimca, šta vidite? Biolog će videti mrežu medusobno povezanih organa, fizičar će videti atome **I** sile koje deluju, a klasifikator će videti predstavnika vrste *canine domesticus.*

Ovo je poslednja stvar koja nas u ovom trenutku interesuje. Pas pripada porodici canine, koja spada u grupu sisara i tako dalje. Klasifikatori dele svet živih bića u carstva, plemena, klase, redove, familije, rodove i primerke.

Ova hijerarhija ostvaruje;e relaciju. Pas/e primerak iz porodice canine. Ovu relaciju možemo pronaći gde god da pogledamo: "toyota"/č vrsta kola, a kola su vrsta vozila. Puding;'ć vrsta deserta, koji *je* vrsta hrane.

Šta podrazumevamo pod tim kada kažemo da je nešto vrsta nečega drugog? Mi podrazumevamo da je to specijalizacija odredene stvari. Iz ovoga proizilazi da je automobil posebna vrsta vozila.

Nasledivanje i izvodenje

Koncept psa se *nasleduje,* što će reći da on automatski preuzima sve osobine sisara. S obzirom da je u pitanju sisar, mi znamo da se on kreće, da udiše vazduh, pošto se svi sisari kreću i udišu vazduh, po definiciji. Koncept psa nam u postojeću definiciju dodaje nove ideje o lajanju, mahanju repa i tako dalje. Dalje, psi se mogu podeliti na lovačke i terijere, a terijeri se dalje mogu podeliti na jorksirske terijere, Dandie Dinmont terijere i tako dalje.

Jorkširski terijer je vrsta terijera, pa je, prema tome, on i vrsta psa, i sisar, pa je stoga i vrsta životinje i stoga je, na kraju, i vrsta živog bića. Ova hijerarhija je prikazana na slici 12.1.

Slika 12.1.
Hijerarhija
životinja

C++ pokušava da predstavi ove relacije tako što Vam omogućava da definišete klase koje se izvode jedne iz drugih. Ovo izvodenje je način predstavljanja/e relacije. Novu klasu Pas izvešćete iz klase Sisar. Nećete morati eksplicitno da navodite da se psi kreću, pošto su ovu osobinu oni *nasledili* od Si sara.

y^MШjmIJ Za klasu koja dodaje novu funkcionalnost postojećoj klasi kaže se da je *izvedena* iz originalne klase. Za originalnu klasu se kaže da je *bazna klasa* nove klase.

Ako se klasa Pas izvodi iz klase Sisar, tada je Sisar bazna klasa klase Pas. Izvedene ™^ klase su nadset svojih baznih klasa. Kao što pas dodaje određene karakteristike pojmu *sisara,* klasa Pas će dodati određene metode, ili podatke klasi Sisar.

Obično, bazne klase imaju vise od jedne izvedene klase. Baš kao što su psi, mačke i konji tipovi sisara, tako će i njihove klase biti izvedene iz klase Sisara.

Životinjsko carstvo

Da bismo učinili jasnijom diskusiju o izvođenju i nasledivanju, ovo poglavlje će fokusirati relacije između raziličitih klasa koje predstavljaju životinje. Možete zamisliti da je od Vas traženo da dizajnirate dečiju igru, koja je simulacija farme.

Razvićete ceo niz životinja sa farme: konje, krave, pse, mačke, ovce, itd. Kreiraćete metode za ove klase tako da se oni mogu ponašati na načine koje deca mogu da očekuju, ali ćete, za sada, svaki metod definisati jednostavnom print naredbom.

Definisanjem funkcija na ovaj način, dobićete samo onoliko informacija koliko je neophodno da biste prikazali da je funkcija pozvana, a detalje ćete ostaviti za kasnije, kada budete imali vise vremena. Slobodno proširite minimalni kod, koji je prikazan u ovom poglavlju, kako biste životinje prikazali realističnije.

Sintaksa izvođenja

Kada deklarišete klasu, u mogućnosti ste da označite iz koje klase je ona izvedena, tako što ćete napisati; (tačka-zapeta) iza imena klase, tip izvodenja (publ ic, ili neki drugi) i ime klase iz koje je izvedena. Evo jednog primera:

```
class Dog : public Mammal
```

Tip izvođenja će biti objašnjen kasnije u ovom poglavlju. Za sada, koristite public. Klasa iz koje ste izvršili izvodenje mora već biti deklarisana, ili ćete dobiti grešku u kompilaciji. U listingu 12.1 ilustrovano je kako se deklariše klasa Dog, koja je izvedena iz klase Mammal.

Listing 12.1: Jednostavno nasledivanje.

```
1:    //Listing 12.1 Jednostavno nasledivanje
2:
3:    linclude <iostream.h>
4:    enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
5:
6:    class Mammal
7:    {
8:    public:
9:       // konstruktori
10:      Mammal();
11:      -Mammal();
```

jo. *nastavlja se*

Listing 12.1: Jednostavno nasledivanje. *nastavak*

```
13      //metode pristupa
14      int   GetAge()const;
15      void  SetAge(int);
16      int   GetWeightO   const;
17      void  SetWeightQ;
18
19      //Druge  metode
20      void  Speak();
21      void  Sleep();
22
23
24    protected:
25       int itsAge;
26       int itsWeight;
27
28
29    class Dog : public Mammal
30
31    publlc:
32
33       // Konstruktori
34       Dog();
35       -Dog();
36
37       // Metode pristupa
38       BREED GetBreed()  const;
39       void SetBreed(BREED);
40
41       // Druge metode
42       // WagTail();
43       // BegForFoodO ;
44
45    protected:
46       BREED  itsBreed;
47
```

Ovaj program ne daje nikakav izlaz, budući da je on samo mz deklaracija klase bez njene implementacije. No. bez obzira na to, u njemu je prikazano dosta stvari koje treba upoznati.

'JT7:T/ŠJ1K U linijama 6-27 deklarisana je klasa Mammal. Primetićete da se u ovom primeru klasa Mammal ne izvodi iz neke druge klase. U realnom svetu, sisari su izvedeni, s obzirom da su deo klase životinja. U C++ programima možete predstaviti samo deo raspoloživih informacija o bilo kojem datom objektu. Realnost je previse kompleksna, da biste obuhvatili sve informacije, pa, stoga, svaka C++ hijerarhija predstavlja odgovarajuću reprezentaciju raspoloživih podataka. Trik za dobar dizain je u tome da zone koje su Vam interesantne prikažete mapirajući ih nazad u realnost u razumnom obimu.

Hijerarhija mora negde da započne; ovaj **program** započinje **sa** Mammal. **Zbog ove** odluke, odredene promenljive **članstva, koje bi trebalo da pripadaju višoj baznoj** klasi, su predstavljene ovde. **Na primer,** praktično **sve životinje imaju godište i** težinu, te, stoga, ako je Mammal izveden **iz** Animal, **možemo** očekivati **da atribute. Zato** se ovi atributi pojavljuju u klasi Mammal.

Da bi program bio jednostavan za razumevanje, u klasu Mammal je dodato **samo** šest metoda - Četiri pristupna metoda, Speak 0 i Sleep 0-

Klasa Dog se nasleduje iz klase Mammal, kao što je naznačeno u liniji 29. Svaki objekat Dog će imati tri promenljive članstva: itsAge, itsWeight i itsBreed. Primetićete da deklaracija za klasu Dog ne uključuje promenljive članstva i tsAge i i tsWei ght. Objekti Dog će ove promenljive naslediti iz klase Mammal, zajedno sa svim Mammal metodima, izuzev operatora za kopiranja i konstruktora i destrukotora.

Private protiv Protected

Verovatno ste primetili novu ključnu reč protected u linijama 24 i 45 listinga 12.1. Kao prvo, podaci o klasi su bili deklarisani kao pri vate. Medutim, private članovi nisu na raspolaganju izvedenim klasama. Možete itsAge i itsWeight proglasiti za public, ali nije poželjno. Nećete želeti da druge klase direktno pristupaju ovim članovima podataka.

Ono što vi želite je da to bude vidljivo ovoj klasi i klasama koje su izvedene iz nje. Zaštićeni članovi podataka i funkcije su potpuno vidljivi u izvedenim klasama, a u svim drugim slučajevima se ponašaju kao private.

Ovde imamo ukupno tri specifikacije pristupa:*public, protected [private.* Ako funkcija sadrži jedan objekat Vaše klase, ona može pristupiti svim public članovima podataka i funkcijama. Funkcije članovi, na dalje, mogu pristupiti svim private članovima podataka i funkcijama sopstvene klase i svim *protected* članovima podataka i funkcijama bilo koje klase iz koje su izvedeni.

Stoga, funkcija Dog::WagTai 1() može da pristupi private podatku itsBreed, kao i protected podacima klase Mammal.

Cak i ako su druge klase postavljene izmedu klasa Mammal i Dog (na primer Domaće životinje), klasa Dog će i dalje biti u mogućnosti da pristupi protected članovima klase Mammal, podrazumevajući da sve te druge klase koriste public nasledivanje. O private nasleđivanju će se diskutovati u Danu 15, "Napredno nasledivanje".

Listing 12.2 demonstrira kako da kreirate objekat tipa Dog i kako da, zatim, pristupite podacima i funkcijama tog tipa.

Listing 12.2: Korišćenje izvedenog objekta.

```
1:    //Listing 12.2 KoriSćenje izvedenog objekta
2:
3:    linclude <iostream.h>
4:    enum BREED { YORKIE, CAIRN, DANOIE, SHETLAND, DOBERMAN, LAB };
5:
6:    class Mammal
7:    {
8:     public:
9:        // konstruktori
10:       Mammal():itsAge(2), itsWeight(5){}
11:       -Mammal(){}
12:
13:       //metode pristupa
14:       int GetAge()const { return itsAge; }
15:       void SetAge(int age) { itsAge « age; }
16:       int GetWeightO const { return itsWeight; }
17:       void SetWeight(int weight) { itsWeight = weight; }
18:
19:       //Druge metode
20:       void SpeakQconst { cout « "Mammal sound!\n"; }
21:       void SleepQconst ( cout « "shhh. I'm sleeping.\n"; }
22:
23:
24:    protected:
25:       int itsAge;
26:       int itsWeight;
27:    };
28:
29:   class Dog : public Mammal
30:   {
31:    public:
32:
33:       // Konstruktori
34:       Dog():itsBreed(YORKIE){}
35:       -DogOO
36:
37:       // Metode pristupa
38:       BREED GetBreedQ const { return itsBreed; }
39:       void SetBreed(BREED breed) { itsBreed = breed; }
40:
41:       // Druge metode
42:       void WagTailQ { cout « "Tail wagging.. An"; }
43:       void BegForFood() { cout « "Begging for food...\n"; }
44:
45:   private:
46:      BREED itsBreed;
47:   };
48:
```



```
int main()
{
   Dog fido;
   fi do. Speak 0;
   fido.WagTaiK);
   cout « "Fido is fido.GetAgeO years old\n"
return 0;
```

```
Mammal sound!
Tail wagging...
Fido is 2 years old
```

U linijama 6-27 deklarisana je klasa Mammal (sve njene funkcije su zbijene kako bismo sačuvali prostor). U linijama 29-47 deklarisana je klasa Dog, koja je izvedena iz klase Mammal; stoga, po ovim deklaracijama, svi psi imaju godište, težinu i pasminu.

U liniji 51 deklarisan je pas Fido, koji je nasledio sve atribute klase Mammal, kao i sve atribute klase Dog. Stoga, Fido zna kako da WagTail (), ali takode zna kako da Speak() i Sleep().

Konstruktori i destruktor!

Dog objekti *su* Mammal objekti. Ovo je suština relacije *je*. Kada je kreiran Fi do, najpre je pozvan njegov bazni konstruktor, koji je kreirao Mammal. Zatim je pozvan konstrukor Dog, koji je kompletirao konstrukciju objekta Dog. Pošto Fidou nismo dodelili parametre, u svim slučajevima su pozvani podrazumevani konstruktori. Fido ne može postojati, dok ne bude kompletno konstruisan, što će reći da oba njegova dela, Mammal i Dog, moraju biti konstruisani. Stoga, oba konstruktora moraju biti pozvana.

Kada uništavamo Fidoa, prvo moramo pozvati destruktor za Dog, a zatim dekstruktora za Mammal deo Fidoa. Svakom destruktoru je data mogućnost da očisti sopstveni deo Fidoa. Nemojte zaboraviti da počistite iza Vašeg psa! Ovo će biti demonstrirano u listingu 12.3.

Listing 12.3: Poiivanje konstruktora i destruktora.

```
1:    //Listing 12.3 Pozivanje konstruktora i destruktora.
2:
3:    #include <iostream.h>
4:    enum BREED {x YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB );
5:
6:    class Mammal
7:    {
8:     public:
9:        // konstruktori
10:       Mammal();
11:       -Mammal();
```

nastavljase

Listing 12.3: Pozivanje konstruktora i destruktora. *nastavak*

```
        //metode pristupa
        int GetAgeO const { return itsAge; )
        void SetAge(int age) { itsAge = age; }
        int GetWeightO const { return itsWeight; }
        void SetWeight(int weight) { itsWeight = weight; }
18
19      //Druge metode
20      void Speak() const } cout « "Mammal sound!\n"; }
21      void Sleep() const } cout « "shhh. I'm sleeping.\n"
22
23
24    protected:
25      int itsAge;
26      int itsWeight;
27    };
28
29    class Dog : public Mammal
30    }
31    publi c:
32
33      // Konstruktori
34      Dog();
35      -Dog();
36
37      // Metode pristupa
38      BREED GetBreed() const } return itsBreed; }
39      void SetBreed(BREED breed) } itsBreed = breed; }
40
41:     // Druge metode
42      void WagTail() } cout « "Tail wagging...\n"; }
43      void BegForFood() } cout « "Begging for food...\n"
44
45    private:
46      BREED itsBreed;
47    };
48
49    Mammal::Mammal ():
50:   itsAge(l),
51    i tsWei ght(5)
52    }
53      cout « "Mammal constructor...\n"·
54    }
55
56    Mammal ::-Mammal()
57    }
58      cout « "Mammal destructor...\n"·
59    }
60:
61:   Dog::Dog():
62:   itsBreed(YORKIE)
63:   }
64:     cout « "Dog constructor...\n";
65:   }
66:
67:   Dog::-Dog()
68:   }
69:     cout « "Dog destructor...\n";
70:   }
71:   int main()
72:   }
73:     Dog fido;
74:     fido.Speak();
75:      fido.WagTail();
76:     cout « "Fido is " « fido.GetAgeO « " years old\n";
77:   return 0;
```

```
Mammal constructor...
Dog constructor...
Mammal sound!
Tail wagging...
Fido is 1 years old
Dog destructor...
Mammal destructor...
```

Listing 12.3 je sličan listingu 12.2, osim što konstuktori i destrukton sada prikazuju na ekranu poruku da su pozvani. Prvo je pozvan konstruktor Mammal, a, zatim, Dog. U ovom trenutku Dog je potupuno funkcionalan i njegovi metodi mogu biti pozvani. Kada Fido izade iz opsega, bi}e pozvan destuktor za Dog, a, odmah zatim, iza Mammal.

Prosleđivanje argumenata baznim konstruktorima

Moguće je da ćete poželeti da konstruktor za Mammal preuzme specifični uzrast i da konstruktor za Dog preuzme pasminu. Kako ćete dobiti parametre o godinama i težini koji su prosleđeni odgovarajućem konstruktoru za Mammal? Šta ako Dog želi da inicijalizuje težinu, ali Mammal neće?

Bazna inicijalizacija klase može se izvršiti tokom inicijalizacije klase, tako što ćete napisati ime bazne klase. Njega slede parametri koje očekuje bazna klasa. Ovo je demonstrirano u listingu 12.4.

Listing 12.4: Preklapajuli konstruktori u izvedenim klasama.

```
1:    //Listing 12.4 PreklapajuĆi konstruktori u izvedenim klasama
2:
3:    #include <iostream.h>
4:    enum BREED } YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
5:
6:    class Mammal
7:    }
8;    public:
9:       //  konstruktori
10:      Mammal();
11:      Mammal(int age);
12:      -Mammal();
13:
14:      //metode pristupa
15:      int GetAgeO const } return itsAge; }
16:      void SetAge(int age) } itsAge = age; }
17:      int GetWeightO const } return itsWeight; }
18:      void SetWeight(int weight) } itsWeight = weight; }
19:
20:      //Druge metode
21:      void Speak() const } cout « "Mammal sound!\n"; }
22:      void SleepO const } cout « "shhh. I'm sieeping.\n"; }
23:
24:
25:   protected:
26:      int itsAge;
27:      int itsWeight;
28:   };
29:
30:   class Dog : public Mammal
31:   }
32:   public:
33:
34:      //  Konstruktori
35:      Dog();
36:      Dog(int age);
37:      Dog(int age, int weight);
38:      Dog(int age, BREED breed);
39:      Dog(int age, int weight, BREED breed);
40:      -Dog();
41:
42:      // Metode pristupa
43:      BREED GetBreed() const } return itsBreed; }
44:      void SetBreed(BREED breed) } itsBreed = breed; }
45:
46:      // Druge metode
47:      void WagTailO } cout « "Tail wagging...\n\ }
48:      ˇoid BegForFood() } cout « "Begging for food...\n"; }
private:
   BREED itsBreed;
};

Mammal::Mammal():
itsAge(l),
itsWeight(5)
}
   cout « "Mammal constructor...\n";
}

Mammal::Mammal(int age):
itsAge(age),
itsWeight(5)
}
   cout « "Mammal(int) constructor...\n";
>

Mammal::-Mammal()
}
   cout « "Mammal destructor...\n";
}

Dog::Dog():
Mammal(),
itsBreed(YORKIE)
}
   cout « "Dog constructor...\n";
}

Dog::Dog(int age):
Mammal(age),
itsBreed(YORKIE)
}
   cout « "Dog(int) constructor...\n";
}

Dog::Dog(int age, int weight):
Mammal(age),
itsBreed(YORKIE)
}
   itsWeight = weight;
   cout « "Dog(int, int) constructor...\n"
}

Dog::Dog(int age, int weight, BREED breed):
Mammal(age),
i tsBreed(breed)
```


nastaiija se

Listing 12.4: Preklapajući konstruktori u izvedenim klasama. ***nastavak***

```
98:
99:       itsWeight = weight;
100       cout « "Dog(int, int, BREED) constructor.\n";
101    }
102
103    Dog::Dog(int age, BREED breed):
104    Mammal(age),
105    itsBreed(breed)
106    }
107       cout « "Dog(int, BREED) constructor...\n";
108    }
109
110    Dog::-Dog()
111    }
112       cout « "Dog destructor...\n";
113    }
114    int  main()
115    }
116       Dog fido;
117       Dog rover(5);
118       Dog buster(6,8);
119       Oog yorkie  (3,YORKIE);
120       Dog dobbie (4,20,DOBERMAN);
121       fido.Speak();
122       rover.WagTai1(),
123       cout « "Yorkie is " « yorkie.GetAgeO << " years old\n"
124       cout « "Dobbie weighs ";
125       cout « dobbie.GetWeightO « " pounds\n";
126     return 0;
-.27
```

^MAPOMEMA^ Izlaz je ovde numerisan tako da svaka linija može biti prokomentarisana u analizi.

```
Mammal constructor...
Dog constructor...
Mammal(int) constructor...
Dog(int) constructor...
Mammal(int) constructor...
Dog(int, int) constructor...
Mammal(int) constructor...
Dog(int, BREED) constructor____
Mammal(int) constructor...
Dog(int, int, BREED) constructor.
Mammal sound!
Tail wagging...
Yorkie is 3 years old.
Dobbie weighs 20 pounds.
Dog destructor. . .
16: Mammal destructor.
17: Dog destructor...
18: Mammal destructor.
19: Dog destructor...
20: Mammal destructor.
21: Dog destructor...
22: Mammal destructor.
23: Dog destructor...
24: Mammal destructor.
```

tl'MIW^K U listingu 12.4, u liniji 11, konstruktor **Mammal** je napravljen **tako** da preuzme celobrojnu vrednost godine. Rešenje u linijama 61-66 inicijalizuje **i t** sAge vrednošću koja je prosleđena u konsturktor i inicijalizuje **i tsWei ght** vrednošću 5.

Dog je napunio pet konstruktora u linijama 35-39. Prvi je podrazumevani konstruktor, a drugi preuzima godine (one su isti parametar koji je preuzeo i konstruktor Mammal). Treći konstruktor preuzima i godine i težinu, četvrti godine i rasu i peti godine, težinu i rasu.

Primetićete da u liniji 74 podrazumevani konstruktor **za** Dog poziva podrazumevani konstruktor za Mammal. Iako **ovo** nije striktno neophodno, služi **kao** dokumentacija **da** ste nameravali da pozovete bazni konstruktor, koji ne zahteva parametre. Bazni **konstruktor** bi bio pozvan u svakom slučaju, ali, time **što** ste **ovo** uradi Vi ste Vašu nameru učinili eksplicitnom.

Implementacija za Dog konstruktor, koji preuzima celobrojnu vrednost, je u linijama 80-85. U njegovoj fazi inicijalizacije (linije 81-82) Dog će inicijalizovati baznu klasu, proslediti parametar i zatim će inicijalizovati pasminu.

Još jedan Dog konstruktor je u linijama 87-93. Ovaj preuzima dva parametra. On inicijalizuje svoju baznu klasu, pozivanjem odgovarajućeg konstruktora, ali u ovom slučaju on, takode, dodeljuje težinu svojoj promenljivoj bazne klase itsWeight. Primetićete da u fazi inicijalizacije ne možete dodeliti vrednost promenljivoj bazne klase. Pošto Mammal ne poseduje konstruktor, koji bi preuzeo ovaj parametar, ovo morate učiniti unutar Dog konstruktora.

Prošetajte se kroz ostale konstruktore, kako biste se uverili da ste razumeli način na koji oni rade. Primetićete šta se može inicijalizovati, a šta mora da čeka, kako bi se izvršilo unutar konstruktora.

Izlaz je numerisan, tako da se lakše može analizirati. Prve dve linije izlaza predstavljaju Fidoa, korišćenjem podrazumevanog konstruktora.

U izlazu linije 3 i 4 predstavljaju kreiranje Rovera. Linije 5 i 6 predstavljaju buster. Primetićete da konstruktor Mammal preuzima celobrojnu vrednost, dok konstruktor Dog preuzima dve celobrojne vrednosti.

Pošto su svi objekti kreirani, oni su i iskorišćeni i, nakon toga, napušteni. Tokom uništavanja svih objekata, prvo je pozvan destruktor Dog, a, zatim, destruktor Mammal, ukupno pet.

з з г Overriding funkcije

Dog objekat može da pristupi svim funkcijama članstva klase Mammal, kao i bilo kojoj funkciji članstva, kao što je WagTai 1 ( ), koju deklaracija klase Dog može da dodeli. On, takođe, poseduje mogućnost overriding-a funkcije bazne klase funkcije. Kada vršite override funkcije, to znači da vršite izmenu implementacije bazne klase funkcije u izvedenoj klasi. Kada pravite neki objekat izvedene klase, Vi pozivate ispravnu (tačnu) funkciju.

JIIIIIMIII Kada izvedena klasa kreira funkciju sa istim povratnim tipom i potpisom, kao što je funkcija koja je Clan bazne klase, ali sa novom implementacijom, onda kažemo da je u pitanju *overriding* ove metode.

Kada dode do override-a funkcije, ona mora da se složi prema povratnom tipu i potpisu sa baznom klasom funkcije. Potpis je prototip funkcije, različit od povratnog tipa, a to su: ime, parametar lista i ključna reč const, ukoliko je upotrebljena.

Potpis funkcije predstavlja njeno ime, a, istovremeno, i broj i tip njenih parametara. Potpis ne uključuje povratni tip.

Listing 12.5 prikazuje šta će se dogoditi ako klasa Dog override-uje Speak() metod u Mammal u. Da bismo uštedeli prostor, izostavili smo funkcije pristupa iz ovih klasa.

listing 12.5: Preklapanje metode osnovne klase u izvedenoj klasi.

```
1:    //Listing 12.5 Preklapanje metode osnovne klase u izvedenoj klasi
2:
3:    #include <iostream.h>
4:    enum BREED } YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
5:
6:    class Mammal
7:    }
8:    public:
9:       //  konstruktori
10:      Mammal() } cout « "Mammal constructor...\n"; }
11:      -Mammal() } cout « "Mammal destructor...\n"; }
12:
13:      //Druge metode
14:      void Speak()const } cout « "Mammal sound!\n"; }
15:      void Sleep()const } cout « "shhh. I'm sleeping.\n"; }
16:
17:
18:   protected:
19:      int itsAge;
20:      int itsWeight;
21:   };
222:
23:   class Dog : public Mammal
24:   }
25:   public:
26:
27:      //  Konstruktori
28:      DogQ} cout « "Dog constructor...\n"; }
29:      -Dog()} cout « "Dog destructor...\n"; }
30:
31:      // Druge metode
32:      void WagTail() } cout « "Tail wagging...\n"; }
33:      void BegForFood() } cout « "Begging for food...\n"
34:      void Speak()const } cout « "Woof!\n";  }
35:
36:   private:
37:      BREED itsBreed;
38:   };
39:
40:   int  main()
41:   }
42:      Mammal bigAnimal;
43:      Dog fido;
44:      bigAnimal.Speak();
45:      fido.Speak();
46:    return 0;
47:   }
```

```
Mammal constructor...
Mammal constructor...
Dog constructor...
Mammal sound!
Woof!
Dog destructor...
Mammal destructor...
Mammal destructor...
```

Il'HUJ^i»i» U liniji 34 klasa Dog vrši override Speak metoda, i primorava Dog objekat da izgovori Woof! kada je pozvan Speak() metod. U liniji 43 kreiran je Mammal objekat bigAnimal, koji prouzrokuje prvu liniju izlaza, kada je pozvan Mammal konstruktor. U liniji 43 kreiranje Dog objekat Fido - on prouzrokuje dve linije izlaza, ako su pozvani prvo Mammal konstruktor, a, zatim, i Dog konstruktor.

U liniji 44 Mammal objekat poziva svoj Speak() metod, zatim u liniji 45 Dog objekat poziva svoj Speak() metod. Izlaz prikazuje kako su ispravne metode pozvane. Na kraju, dva objekta prestaju sa radom i pozivaju se njihovi destrukton.

Overloading protiv Overriding-a

Ova dva pojma su vrlo sliČna. Kada vršite overload metode, Vi kreirate vise od jednog metoda sa istim imenom, ali sa različitim potpisom. Kada vršite override metode, Vi kreirate metod u izvedenoj klasi sa istim imenom kao što je metod u baznoj klasi i sa istim potpisom.

Sakrivanje metoda bazne klase

U prethodnom listingu Speak () metod klase Dog sakrio je metod bazne klase. Ovo je upravo ono što se traži, ali su mogući neočekivani rezultati. Ako Mammal ima metod Move() nad kojim je izvršen overload i Dog vrši override nad tim metodom, Dog metod će sakriti sve Mammal metode pod ovim imenom.

Ako Mammal izvrši overload Move() pomoću tri metoda - jednog koji ne uzima parametre, drugog koji uzima celobrojnu vrednost i trećeg koja uzima celobrojnu vrednost i pravac, i Dog vrši override samo Move() metoda koji ne uzima parametre, neće biti jednostavno pristupiti drugim metodima uz pomoć Dog objekta. U listingu 12.6 je ilustrovan ovaj problem.

Listing 12.6.: Metode sktivanja

```
1:  //Listing 12.6. Metode skrivanja
2:
3:  #include <iostream.h>
4:
5:  class Mammal
6:  }
7:  publi c:
8:     void Move() const } cout « "Mammal move one step\n"; }
9:     void Move(int distance) const
10:    }
11:       cout « "Mammal move ";
12:       cout « distance «" steps.\n";
13:    }
14: protected:
15:    int itsAge;
16:    int itsWeight;
17: };
18:
19: class Dog : public Mammal
20: }
21: publi c:
22: // Dobićete poruku upozurenja da ste sakrili fukciju!
23:    void Move() const } cout « "Dog move 5 steps.\n"; }
24:
25:
26; int main()
27: }
28:    Mammal bigAnimal;
29:    Dog fido;
30:    bigAnimal.Move();
31:    bi gAnimal.Move(2) ;
32:    fido.Move();
33:    // fido.Move(IO);
34:  return 0;
35:
```



```
Mammal move one step
Mammal move 2 steps.
Dog move 5 steps.
```

jSEjEftfr Sve dodatne metode i podaci su udaljeni iz ovih klasa. U linijama 8 i 9 Mammal klasa deklariše Move() metod, nad kojim je izvršen overload. U liniji 18 Dog vrši override verzije Move() bez parametara. One su pozvane u linijama 30-32 i u izlazu je prikazano njihovo izvršenje.

Međutim, linija 33 je iskomentarisana, pošto bi prouzrokovala grešku u kompilaciji. Dok je Dog klasa mogla da pozove Move(int) metod, dok nije izvršila override verzije Move() bez parametara, sada je to uradeno tako da ona mora da izvrši override obe, ukoliko želi da ih koristi. Ovo podseća na pravilo koje kaže da, ako obezbedite bilo koji konstruktor, kompajler vise neće obezbedivati podrazumevani konstruktor.

Uobičajena greška je da sakrijete metod bazne klase kada ste nameravali da uradite override, zaboravljajući da ukljudte ključnu reč const - ona je deo potpisa i njeno izostavljanje menja potpis i stoga ćete izvršiti sakrivanje metoda umesto override-a koji ste želeli.

Override protiv Sakrivanja

U sledećem odeljku opisane su virtuelne metode. Override virtuelne metode podržava polimorfizam, dok sakrivanje podriva (slabi) polimorfizam. Uskoro ćete o ovome saznati i nešto vise.

Pozivanje baznog metoda

Ukoliko ste izvršili override baznog metoda, još uvek postoji mogućnost da ga pozovete, tako što ćete ga pozvati (pod) punim imenom. Ovo ćete učiniti tako što ćete napisati bazno ime, iza koga će slediti dve dvotačke i na kraju ime metoda. Na primer, Mammal: :Move().

Postoji mogućnost da se linija 28 iz listinga 12.6 ponovo napiše na sledeći način:

```
28:     fido.Mammal::Move(10);
```

Na ovaj način se Mammal metod poziva eksplicito. Ova ideja je ilustrovana u listingu 12.7 .

Listing 12.7: Pozivanje osnovne metode iz redefinisane metode.

```
//Listing 12.7 Pozivanje osnovne metode iz redefinisane metode.

#include <iostream.h>

class Mammal
```

nastavlja se

Listing 12.7: Pozivanje osnovne metode iz redefinisane metode. *nastavak*

```
6:    }
7:    public:
8:       void Move() const } cout « "Mammal move one step\n"; }
9:       void Move(int distance) const
10       }
11            cout « "Mammal move " « distance;
12            cout « " steps.\n";
13       }
14
15    protected:
16       int itsAge;
17       int itsWeight;
18;   };
19;
20    class Dog : public Mammal
21
22     ubl ic:
23       void Move()const;
24
25
26
27    void Dog::Move() const
28
29       cout « "In dog move...\n";
30       Mammal::Move(3);
31
32
33     nt main()
34
35       Mammal bigAnimal;
36       Dog fido;
37       bigAnimal.Move(2);
38       fido.Mammal::Move(6);
39     return 0;
40
```

```
Mammal move 2 steps.
Mammal move 6 steps.
```

U liniji 35 kreiranje `bigAnimal` i u liniji 36 `Dog fido`. Metoda pozvana u liniji 37 poziva `MoveQ` metoda `Mammal-a`, koji preuzima celobrojni argument.

Programer je želeo da pozove `Move (int)` u `Dog` objektu, ali je naišao na problem. `Dog` je izvršio override `MoveQ` metod, ali ne i overload, i nije obezbedio verziju koja preuzima celobrojni argument. Ovaj problem je rešen eksplicitnim pozivanjem `Move (int)` metoda bazne klase u liniji 33.

^| **PAZTTC** |£t Koristite proširenje funkcionainosti testiranih klasa, uz pomoć derivacije.

Koristite izmenu ponašanja odredenih funkcija izvedene klase, tako što ćete izvršiti override metoda bazne klase.

Nemojte sakrivati funkciju bazne klase promenom potpisa funkcije.

Virtuelne metode

U ovom poglavlju ćemo pažnju posvetiti činjenici da je `Dog` objekat `Mammal` objekat. Do sada je smatrano da `Dog` objekat nasleduje atribute (podatke) i mogućnosti (metode) od svoje bazne klase. Međutim, u C++ - u je relacija ide dublje od ovoga.

C++ proširuje svoj polimorfizam, da bi omogućio pokazivačima na baznu klasu da budu dodeljeni objektima izvedenih klasa. Stoga, Vi možete napisati

```
Mammal* pMammal = new Dog;
```

Na ovaj način se kreira novi `Dog` objekat na "steku", a pointer se vraća na taj objekat, koji se dodeljuje pointeru na `Mammal`. Ovo je u redu, zato što je pas sisar.

ШОМЕНАy Ovo je suština polimorfizma. Na primer, Vi možete kreirati mnogo različitih **tipova prozora,** uključujući i box-ove za dijalog, prozore za skrolovanje i list box-ove i svakom od njih dodeliti virtuelnu `draw()` metodu. Kreiranjem pointera za prozore i dodeljivanjem box-ova za dijalog i ostalih izvedenih tipova ovom pointeru, Vi možete pozvati `draw()`, bez obzira na aktuelan tip objekta na koji ukazujete. U svakom slučaju će biti pozvana ispravna `draw()` funkcija.

Ovaj pointer, zatim, možete koristiti za pozivanje bilo kog `Mammal` metoda. Ono što biste Vi želeli jeste da ovi metodi koji vrše override `Dog ()` pozivaju ispravne funkcije. Virtuelne funkcije Vam to omogućavaju. Listing 12.8 ilustruje kako ovo radi i šta se dešava sa ne-virtuelnim metodima.

Listing 12.8: Korišćenje virtualnih metoda.

```
      //Listing 12.8 Koi}čenje virtualnih metoda

      linclude <iostream.h>

      class Mammal
      }
      public:
         Mammal():itsAge(l) } cout « "Mammal constructor...\n"; }
9:       -Mammal() } cout « "Mammal destructor...\n"; }
10:      void Move() const } cout « "Mammal move one step\n"; }
11:      virtual void Speak() const } cout « "Mammal speak!\n"; }
12:   protected:
13:      int itsAge;
14:
15:
```

nastavlja se

* Listing 12.8: Korišćenje virtualnih metoda. *nastavak*

```
16:
17:    class Dog : public Mammal
18:    }
19:    public:
20:       Dog() } cout « "Dog Constructor...\n"; }
21:       ~Dog() } cout « "Dog destructor...\n"; }
22:       void WagTai1() } cout « "Wagging Tail...\n"; }
23:       void SpeakQconst } cout « "Woof!\n\ }
24:       void Move()const } cout « "Dog moves 5 steps...\n"; }
25:    };
26:
27:    int main()
28:    }
29:
30:       Mammal *pDog = new Dog;
31:       pDog->Move();
32:       pDog->Speak();
33:
34:    return 0;
35:  }
```

```
Mammal constructor...
Dog Constructor...
Mammal move one step
Woof!
```

U liniji 11 Mammal u je obezbedena virtuelna metoda - speak(). Dizajner ove klase zato dramatično signalizira da on očekuje da ova klasa eventualno bude bazni tip neke druge klase. Izvedena klasa će, verovatno, želeti da izvrši override ove funkcije.

U liniji 30 pokazivač za Mammal je kreiran (pDog), ali mu je dodeljena adresa novog Dog objekta. Postoje passisar, ovo je legalno dodeljivanje. Pointer je, zatim, upotrebljen za pozivanje Move() funkcije. Pošto kompajler zna da pDog može da bude samo Mammal, on će se obratiti Mammal objektu, da bi pronašao Move() metod.

U liniji 32 pokazivač poziva Speak() metod. Pošto je Speak() metod virtuelan, biće pozvan Speak() metod iz Doga, nad kojim je izvršen override.

Ovo se čini gotovo čarobno. Dokle god je poznata pozivajuća funkcija, ona poseduje Mammal pokazivač i biće pozvan Dog metod. U stvari, ako imate niz pointera na Mammal, od kojih je svaki usmeren na potklasu Mammal a, Vi ih možete pozivati redom, tako da će biti pozivane ispravne funkcije. Listing 12.9 ilustruje ovu ideju.

Listing 12.9: Vise virtualnih funkcija koje se pozivaju redom.

```
1:    //Listing 12.9 Vise virtualnih funkcija koje se pozivaju redom
2:
3:    #inc1ude <iostream.h>
4:
5:    class Mammal
6:    }
7:    public:
         Mammal ():itsAge(l) }  }
         -Mammal () } }
         virtual void Speak() const } cout « "Mammal speak!\n"; }
      protected:
         int itsAge;


      class Dog : public Mammal

      public:
         void SpeakQconst } cout « "Woof!\n"; }



      class Cat : public Mammal

      public:
         void SpeakQconst } cout « "Meow!\n"; }



      class Horse : public Mammal

      publi c:
         void SpeakQconst } cout « "Winnie!\n"; }


      class Pig : public Mammal

      public:
         void SpeakQconst } cout « "0ink!\n"; }


       nt mainQ

         Mammal * theArray[51 ;
         Mammal* ptr;
         int choice, i;
         for { i = 0; i<5; i++)
         }
            cout « "(l)dog (2)cat (3)horse (4)pig:
            cin » choice;
            switch (choice)
            }
               case 1: ptr = new Dog;
```

nastavlja se

Listing 12.9: Vise virtualnih funkcija koje se pozivaju redom. *nastavak*

```
53:              break;
54:              case 2: ptr · new Cat;
55:              break;
56:              case 3: ptr « new Horse;
57:              break;
58:              case 4: ptr » new Pig;
59:              break;
60:              default: ptr · new Mammal;
61:              break;
62:           }
63:           theArray[i] « ptr;
64:       }
65:       for (i=0;i<5;i++)
66:           theArray[i]->Speak();
67:    return 0;
68: }
```

```
(l)dog (2)cat (3)horse (4)pig: 1
(l)dog (2)cat (3)horse (4)pig: 2
(l)dog (Z)cat (3)horse (4)pig: 3
(l)dog (2)cat (3)horse (4)pig: 4
(l)dog (2)cat (3)horse (4)pig: 5
Woof!
Meow!
Winnie
Oink!
Mammal speak!
```

§M)MIMfcfr Ovaj program, koji pruža minimalnu funkcionalnost svakoj od klasa, ilustruje virtuelne funkcije u njihovoj najčistijoj formi. Deklarisane su četiri klase: Dog (pas), Cat (mačka), Horse (konj) i Pig (svinja) i sve su izvedene iz Mammal a.

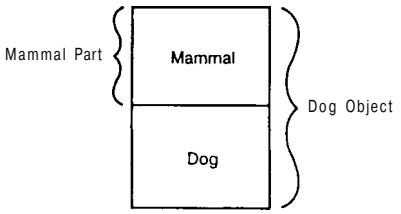
U liniji 10 Mammalova SpeakQ funkcija je deklarisana kao virtuelna. U linijama 18, 25, 32 i 38 četiri izvedene klase su izvršile override implementacije Speak().

Korisnik je upitan koje će objekte da kreira, a pointeri su dodati nizu linija 46-64.

yNAPOMItil^ U vreme kompajliranja nemoguće je znati koji će objekti biti kreirani, pa, prema tome, ni koja će od Speak () metoda biti pozvana. Pointer ptr će biti povezan sa odgovarajućim objektom u trenutku izvršavanja. Ovo se naziva dinamičko (po)vezivanje, ili povezivanje u fazi izvršavanja, suprotno od statičkog povezivanja, ili povezivanja u vreme kompajliranja.

Kako rade virtuelne funkcije?

Kada je kreiran izvedeni objekat, kao što je Dog objekat, prvo se poziva konstruktor za baznu, a, zatim, konstruktor za izvedenu klasu. Na slici 12.2 prikazano je kako izgleda Dog objekat pošto je kreiran. Primetićete da se Mammal deo objekta u memoriji graniči sa Dog delom.

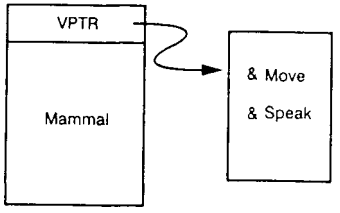


Slika 12.2.
Dog objekat posle kreiranja

Kada je u nekom objektu kreirana virtuelna funkcija, objekat mora da je sledi. Mnogi kompajleri prave tabele virtuelne funkcije, nazvane v-table. Za svaki tip se vodi po jedna tabela i svaki objekat ovog tipa vodi pointer na virtuelnu tabelu (vptr ili v-pointer), koji ukazuje na tabelu.

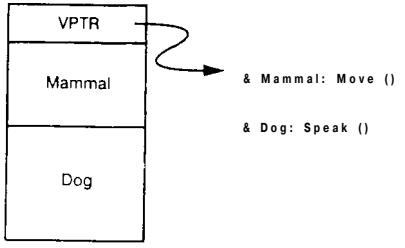
I dok implementacije variraju, svi kompajleri moraju postići isto, tako da nećete mnogo pogrešiti sa ovim opisom.

Svaki vptr, objekat ukazuje na v-table, on ima pokazivač koji ukazuje na svaku od virtuelnih funkcija (primetićete da će se o pokazivačima za funkcije detaljnije diskutovati u Danu 14, "Specijalne klase i funkcije"). Kada je Mammal deo psa kreiran, vptr je inicijalizovan da ukaže na ispravan deo v-table, kao što je prikazano na slici 12.3.



Slika 12.3.
v-table Mammal-a

Kada je pozvan Dog konstruktor i pridodat Dog deo ovog objekta, vptr je postavljen tako da bi mogao da ukaže na virtuelnu funkciju (ako postoji), koja vrši override u Dog objeku (videti sliku 12.4).



Slika 12.4.
v-table psa

Kada je upotrebljen pointer za Mammal, vptr nastavlja da ukazuje na ispravnu funkciju, u zavisnosti od "stvamog" tipa objekta. Stoga, kada je pozvan Speak () metod, pozvana je tačna (ispravna) funkcija.

Odavde ne možete tamo

Ako Dog objekat ima metod, WagTai 1 (), koji nije Mammal, Vi nećete mod da koristite pointer za Mammal da biste pristupili ovom metodu (osim ako ga ne proglasite da bude pointer za Dog). Pošto WagTai 1 nije virtuelna funkcija i zato što nije u Mammal objektu, tamo ne možete stići bez Dog objekta, ili bez Dog pointera.

Pošto ne možete da transformišete Mammal pointer u Dog pointer, postoje daleko bolji i sigurniji načini da se pozove WagTai l () metod. C + + ne odobrava eksplicitnu podelu uloga, zato što su osetljive na grešku. Ova tema će biti detaljnije obradena u odeljku "Višestruko nasledivanje" u sledećem Danu, i u templejtima, koji će biti obrašnjeni u Danu 20, "Izuzeci i obrada grešaka".

Slicing

Primetićete da magija virtuelne funkcije funkcioniše samo sa pointerima i referencama. Prosledivanje nekog objekta po vrednosti neće omogućiti pozivanje virtuelnih funkcija. U listingu 12.10 biće ilustrovan ovaj problem.

Listing 12.10: Odsecanje podataka prilikom predavanja po vrednosti.

```
//Listing 12.10 Odsecanje podataka sa predavanjem po vrednosti

linclude <iostream.h>

enum BOOL } FALSE, TRUE };
class Mammal
}
public:
   Mammal():itsAge(l) }   }
   -Mammal () } }
   virtual void Speak() const } cout « "Mammal speak!\n"; }
protected:
   int itsAge;
};

class Dog : public Mammal
}
publi c:
   void SpeakQconst } cout « "Woof!\n"; }
};

class Cat : public Manmal
public:
   void Speak()const } cout « "Meow!\n"; }
};

void ValueFunction (Mammal);
void PtrFunction    (Mammal*);
void RefFunction (Mammali);
int main()
}
   Mammal* ptr=0;
   int choice;
   while (1)
   }
      B00L fQuit = FALSE;
      cout « "(Ddog (2)cat (0)Quit:
      cin » choice;
      switch (choice)
      }
         case 0: fQuit = TRUE;
         break;
         case 1: ptr = new Dog;
         break;
         case 2: ptr = new Cat;
         break;
         default: ptr • new Mammal;
         break;
      }
       if (fQuit)
         break;
       PtrFunction(ptr);
       RefFunction(*ptr);
       ValueFunction(*ptr);
   }
    return 0;


oid ValueFunction (Mammal MammalValue)

  MammalValue.SpeakO;


oid PtrFunction (Mammal * pMammal)

  pMammal->Speak();


oid RefFunction (Mammal & rMammal)

  rMammal.Speak();
```

```
(1)dog (2)cat (0)Quit: 1
Woof
Woof
Mammal Speak!
(1)dog (2)cat (0)Quit: 2
Meow!
Meow!
Mammal Speak!
(1)dog (2)cat (0)Quit: 0
```

U linijama 6-26 deklarisane su verzije Mammal, Dog i Cat klase. Deklarisane su tri funkcije - PtrFunctionO, RefFunctionO i ValueFunction(). One uzimaju pointer na Mammal, Mammal referencu i Mammal objekat, respektivno. Sve tri funkcije, zatim, rade isto - pozivaju Speak() metod.

Korisnik je upitan da izabere Dog, ili Cat, i, u zavisnosti od izbora koji bude napravio, pointer za pravi tip će biti kreiran u linijama 44-49.

Pointer i reference pozivaju virtuelne funkcije i pozvana je Dog->() funkcija članstva. Ovo je prikazano u prve dve linije izlaza, posle izbora korisnika.

Medutim, pointer koji nije po referenci je prosleden po vrednosti. Funkcija očekuje Mammal objekat, tako da kompajler reže Dog objekat na Mammal deo. U ovom trenutku poziva se Mammal SpeakQ metod, kao što je prikazano u trećoj liniji.

Ovaj eksperiment se, zatim, ponavlja za Cat objekat, sa sličnim rezultatima.

Virtuelni destruktor!

Sasvim je ispravno i uobičajeno je da se pointer prosledi izvedenom objektu, kada se očekuje pointer na bazni objekat. Šta će se dogoditi kada se izbriše ovaj pointer za izvedeni subjekat? Ako je destruktor virtuelan, kao što i treba da bude, desiće se prava stvar - biće pozvan destruktor izvedene klase. Pošto će destruktor izvedene klase automatski pozvati destruktora bazne klase, ceo objekat će biti propisno uništen.

Pravilo je sledeće: ako je bilo koja funkcija u Vašoj klasi virtuelna, tada bi to trebalo da bude i destruktor.

Virtuelni konstruktori za kopiranje

Kao što je već rečeno, ni jedan konstruktor ne može biti virtuelan. Medutim, postoje situacije kada je Vašem programu potrebno da može da prosledi pointer baznom objektu i da ima kopiju pravilno izvedenog objekta koji je kreiran. Uobičajeno rešenje ovog problema je da se kreira Clone() metod u baznoj klasi kao virtuelan. Clone() metoda kreira novu kopiju objekta tekuće klase i vraća taj objekat.

Pošto svaka izvedena klasa vrši override Clone() metode, kreira se kopija izvedene klase. U listingu 12.11 ilustrovano je kako se ovo koristi.

Listing 12.11: Virtualni konstruktor kopije.

```
//Listing 12.11 Virtualni konstruktor kopije

linclude <iostream.h>

class Mammal
}
public:
   Mammal():itsAge(l) } cout « "Mammal constructor...\n"; }
   -Mammal() } cout « "Mammal destructor...\nM; }
   Mammal (const Mammal & rhs);
   virtual void SpeakQ const } cout « "Mammal speak!\n"; }
   virtual Mammal* CloneQ } return new Mammal (*this); }
   int GetAgeQconst } return itsAge; }
protected:
   int itsAge;
};

Mammal::Mammal (const Mammal & rhs):itsAge(rhs.GetAgeO)
}
   cout « "Mammal Copy Constructor...\nM;
}

class Dog : public Mammal
}
public:
   Dog() } cout « "Dog constructor...\n"; }
   -Dog() } cout « "Dog destructor...\n"; }
   Dog (const Dog & rhs);
   void SpeakQconst } cout « "Woof!\n"; }
   virtual Mammal* CloneQ } return new Dog(*this); }
};

Dog::Dog(const Dog & rhs):
Mammal(rhs)
}
   cout « "Dog copy constructor...\n";
>

class Cat : public Mammal
}
public:
   Cat() } cout « "Cat constructor...\n"; }
   -CatQ } cout « "Cat destructor...\n"; }
   Cat (const Cat &);
   void SpeakQconst } cout « "Meow!\n"; }
   virtual Mammal* CloneQ } return new Cat(*this); }
};
```

nastavlja se

Listing 12.11: Virtualni konstruktor kopije.

```
48
49      Cat::Cat(const Cat & rhs):
50      Mammal(rhs)
51      }
52         cout « "Cat copy constructor...\n";
53      }
54
55.     enum ANIMALS } MAMMAL. DOG, CAT};
56      const int NumAnimalTypes = 3;
57      int main()
58      }
59         Mammal *theArray[NumAnimalTypes];
60         Mammal* ptr;
61         int choice, i;
62         for ( i = 0; i<NumAnimalTypes; i++)
63         }
64            cout « "(l)dog (2)cat (3)Mammal: ";
65            cin » choice;
66            switch (choice)
67            }
68               case DOG: ptr = new Dog;
69               break;
70               case CAT: ptr = new Cat;
71               break;
72               default: ptr = new Mammal;
73               break;
74            }
75            theArray[i] = ptr;
76         }
77         Mammal *0therArray[NumAnimalTypes];
78         for (i=0;i<NumAnimalTypes;i++)
79         }
              theArray[i]->Speak();
              0therArray[i ] = theArray[i]->C1one();
           }
           for (i=0;i<NumAnimalTypes;i++)
              OtherArray[i]->Speak();
25       return 0;
         }
```

```
Ш Ш l >     (l)dog (2)cat (3)Mammal: 1
            Mammal constructor...
            Dog constructor...
            (l)dog (2)cat (3)Mammal: 2
            Mammal constructor...
            Cat constructor...
            (l)dog (2)cat (3)Mammal: 3
            Mammal constructor...
            Woof!
```



```
10: Mammal Copy Constructor...
11: Dog copy constructor...
12: Meow!
13: Mammal Copy Constructor...
14: Cat copy constructor...
15: Mammal speak!
16: Mammal Copy Constructor...
17: Woof!
18: Meow!
19: Mammal speak!
```

Ш П Ш ^ Listing 12.11 ima puno sličnosti sa prethodna dva listinga, osim što je Mammal klasi: Clone() pridodata novi virtuelni metod. Ovaj metod vraća pointer na novi Mammal objekat, tako što poziva kopiju konstruktora, prosleđujući mu sebe (*this) kao const referencu.

Dog i Cat vrše CloneO metode, inicijalizujući sopstvene podatke i prosleđujući sopstvene kopije konstruktoru. Pošto je CloneO virtuelna, na ovaj način će se efektivno kreirati virtuelna kopija konstruktora, kao što je prikazano u liniji 81.

Korisniku je ponudeno da izabere izmedu psa, mačke, ili nekog drugog sisara, koji će biti kreirani u linijama 62-74. Pointer za svaki pojedinčni izbor smešten je u nizu, u liniji 75.

Pošto program vrši iteraciju kroz niz, svaki objekat dobija svoje Speak() i svoje CloneO metode, jedan za drugim u linijama 80 i 81. Rezultat pozivanja CloneO je pointer za kopiju objekta, koji se tada smešta u drugi niz u liniji 81.

U liniji 1 na izlazu korisnik je na upit odgovorio sa 1, čime je izabrao da kreira psa. Pozivaju se Mammal i Dog konstruktori. Ovo se ponavlja za Cat (mačku) i Mammal (Sisar) u linija 4-8 konstruktora.

Linija 9 konstruktora predstavlja poziv metoda Speak () prvog Dog objekta. Pozvan je Speak() metod i korektna verzija Speak(). Zatim je pozvana CloneO funkcija, a, zatim, i Dog's Clone metod, pošto je takode virtuelan, što je prouzrokovalo pozivanje Mammal konstruktora i Dog konstruktora za kopiranje.

Isto ovo će se ponoviti i za Cat u linijama 12-14, zatim za Mammal u linijama 15 i 16. Na kraju, izvršena je iteracija novog niza i pozvani su svi Speak() metodi novih objekata.

Kolika je cena virtuelnih metoda

Pošto objekti sa virtuelnim metodima moraju da održavaju v-table, postoje neka ograničenja u njihovom korišćenju. Ako imate vrlo malu klasu, iz koje ne nameravate da izvodite druge klase, onda, možda, uopšte ne postoji razlog za posedovanje bilo kog virtuelnog metoda.

Ako bilo koji metod deklarišete kao virtuelan, Vi treba da platite cenu v-table (iako svaki ulaz koristi malo memorije). U torn trenutku, Vi ćete poželeti da destruktor bude virtuelan, polazeći **od** pretpostavke su i svi drugi metodi, verovatno, isto tako virtuelni. Dobro pogledajte sve ne-virtuelne metode i uverite se da li razumete zašto nisu virtuelni.

ИШП!· Koristite virtuelne metode kada očekujete izvodenja iz klase.

Koristite virtuelne destruktore, ako je bilo koji metod virtuelan.

Ne oznacavajte konstruktore kao virtuelne.

Rezime

Danas ste naučili kako se izvedene nasleduju iz baznih klasa. U ovom poglavlju bilo je reči o javnom nasleđivanju i virtuelnim funkcijama. Klase nasleduju sve javne i zaštićene podatke i funkcije od svojih baznih klasa.

Zaštićeni pristup je javan za izvedene klase i privatan za sve ostale objekte. Cak ni izvedene klase ne mogu da pristupe privatnim podacima, ili funkcijama u njihovim baznim klasama.

Konstruktori se mogu inicijalizovati pre "tela" konstruktora. To se dogada u trenutku kada su pozvani bazni konstruktori i kada parametri mogu biti prosledeni baznoj klasi.

Nad funkcijama u baznoj klasi može da se izvede override u izvedenoj klasi. Ako su funkcije bazne klase virtuelne i ako se objektu pristupa pointerom, ili referencom, funkcije izvedene klase će biti pozvane, na osnovu tipa u fazi izvršenja objekta, na koji pointer ukazuje.

Metodi u baznoj klasi mogu se pozvati eksplicitnim imenovanjem funkcije, sa prefiksom ispred imena bazne klase i dvema dvotačkama. Na primer, ako Dog nasleduje od `Mammal`, `Mammal`'s `walk()` metoda može biti pozvana sa `Mammal::walk()`.

U klasama sa virtuelnim metodama destruktor bi trebalo da, gotovo uvek, bude virtuelan. Virtuelan destruktor obezbeduje da izvedeni deo objekta bude osloboden kada se pozove delete pointera. Konstruktori ne mogu biti virtuelni. Virtuelni konstruktori za kopiranje mogu se efikasno kreirati, tako što ćete napraviti virtuelnu funkciju, koja poziva kopiju konstruktora.

Pitanja i odgovori

P Da li se nasleđeni članovi funkcije prosleduju duž sledeće generacije u nizu? **Ako** se dog izvodi iz `Mammal a` i `Mammal` izvodi iz `Animal a`, da li `Dog` nasleduje funkcije i podatke `Animal a`?

O **Da.** Izvodenje **se** nastavlja. Izvedene klase nasleduju zbir svih funkcija i podataka **iz** svojih baznih klasa.

Ako u prethodnom primeru `MAMMAL` vrši override funkcije u `ANIMALU`, šta će `DOG` dobiti, originalnu, ili override funkciju?

O Ako `Dog` nasleduje `Mammal`, on **će** dobiti funkciju u istom obliku kao što je ima i `Mammal`, **a to je override** funkcija.

P **Mogu li izvedene klase da javne bazne funkcije pretvore u privatne?**

O Da, i one ostaju privatne za sva sledeća izvođenja.

P **Zašto nije dobro da sve funkcije klase budu virtuelne?**

O Postoji određeno opterećenje sa prvom virtuelnom funkcijom u kreiranju v--table. Posle toga opterećenje je zanemarljivo. Mnogi C+ + programeri misle da bi, ako je jedna funkcija virtuelna, i ostale trebalo da budu takve. Drugi programeri se sa tim ne slažu i smatraju da treba uvek imati razlog za ono što radite.

Ako je funkcija (`NEKAFUNCQ`) virtuelna u baznoj klasi i ukoliko je nad njom izvršen overload, i uzima jedan celobrojni parametar, ili dva, i izvedena klasa izvrši override te funkcije, tako da uzima jedan celo**brojni parametar, šta će biti pozvano kada pointer na izvedeni objekat pozove formu sa dva celobrojna parametra?**

0 Override forme sa jednim celobrojnim parametrom sakriće celu funkciju bazne klase i stoga ćete dobiti grešku u kompilaciji, s obzirom da ta funkcija zahteva samo jedan celobrojni parametar.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje obrađene teme i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i proverite da li razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Sta je v-table?
2. Sta je virtuelni destruktor?
3. Kako ćete prikazati deklaraciju virtuelnog destruktora?
4. Kako se kreira kopija virtuelnog konstruktora?

5. Kako se poziva bazna funkcija iz izvedene klase, u kojoj ste uradili override te funkcije?
6. Kako se poziva bazna funkcija izvedene klase, u kojoj niste izvršili override te funkcije?
7. Ako je bazna klasa deklarisala funkciju kao virtuelnu i izvedena klasa nije koristila izraz virtuelna kada je vršila override te klase, da li je ona i dalje virtuelna kada se nasledi u klasi treće generacije?
8. Za šta se koristi ključna reč protected?

Vežbe

1. Prikažite deklaraciju virtuelne funkcije, koja preuzima celobrojni parametar, a vraćavoid.
2. Prikažite deklaraciju klase square, koja se izvodi iz klase rectangle izvedene iz klase shape.
3. Ako u vežbi 2 shape ne uzima parametre, rectangle uzima 2 (dužina i širina), ali square uzima samo jedan (dužina), prikažite inicijalizaciju konstruktora za square.
4. Napišite virtuelnu kopiju konstruktora za klasu square iz vežbe 3.
5. Lovci na greske: šta nije ispravno u sledećem kodu:

```
void SomeFunction (Shape);
Shape * pRect = new Rectangle;
SomeFunction(*pRect);
```

6. Lovci na greške: šta nije ispravno u sledećem kodu:

```
class ShapeO
}
publi c:
      ShapeO;
      virtual -Shape();
      virtual Shape(const Shape&);
};
```

Dan 13

Polimorfizam

U predhodnoj lekciji ste naučili kako se pišu virtuelne funkcije u izvedenim klasama. Ovo je fundament, koji služi za izgradnju polimorfizma: mogućnost povezivanja specifičnih izvedenih objekata klase sa pointerima bazne klase u fazi izvršenja. Danas ćete naučiti:

- šta je višestruko nasleđevanje i kako se koristi
- šta je virtuelno nasledivanje
- šta su apstraktni tipovi podataka
- šta su čiste virtuelne funkcije.

Problemi sa jednostrukim nasleđivanjem

Pretpostavimo da ste izvesno vreme radili sa klasama životinja i da ste ih hijerarhijski razvrstali na Birds (ptice) i Mammals (sisare). Klasa Birds uključuje funkciju člana Fly(). Klasa Mammal je podeljena na veći broj tipova Mammala, uključujući i Horse (konje). Klasa Hors obuhvata funkcije člana WhinnyO i GallopO.

Najednom ste shvatili da Vam je potreban objekat Pegasus (Pegaz): ukrštena rasa Horse (konja) i Bird (ptice). Pegasus može da leti (Fly), rže (Whinny) i galopira (Gallop). Sto se tiče jednostrukog nasledivanja, Vi ste baš u škripcu.

Pegasusa možete da pretvorite u Bird, ali on tada neće moći da Whinny () niti da Gallop(). Možete ga pretvoriti u konja, ali on neće moći da Fly().

Prva mogućnost koja Vam stoji na raspolaganju je da iskopirate **Fly()** metod u klasi **Pegasus i da** izvedete **Pegasusa iz Horsea.** Ovo će **dobro** da funkcioniše, **tako** što ćete **imati Fly() metode na** dva mesta **(Bird** i **Pegasus)**. Ako jedan promenite, ne smete zaboraviti **da** promenite **i** drugi. Naravno, **onaj** ko bude **došao** posle nekoliko mese-**ci, ili, čak, kroz** godinu **dana** da održava Vaš program moraće, takode, da zna da ispravku **treba da** unese **na oba** mesta.

Međutim, uskoro **ćete naići** na **novi** problem. Poželećete da kreirate listu **Horse** objekata, **kao i listu Bird** objekata. Poželećete da možete da dodate Vaš **Pegasus** objekat u obe liste, **ali ako je Pegasus konj,** nećete mod da ga uvrstite u listu ptica.

Imate par potencijalnih rešenja. Možete preimenovati **Horse** metod **GallopO u Move() i** da, zatim, izvršite override **Move()** metoda **u** Vašem **Pegasus** objektu, kako biste obavili posao **sa Fly().** Zatim ćete izvršiti override **Move()** metoda **u** ostalim konjima, kako biste obavili posao **sa GallopO.** Možda će **Pegasus** biti dovoljno pametan da galopira na kratkim rastojanjima i da leti na dugim.

```
Pegasus: :Move(long distance)
I
if (distance > veryFar)
fly(distance);
else
gallop(distance);
I
```

Ovo je pomalo ograničavajuće. Možda će **Pegasus** poželeti jednoga dana da leti na kratkim razdaljinama, ili da galopira na dugim. Vaša naredna solucija bi mogla da bude premeštanje **Fly()** u **Horse,** kao što je prikazano u listingu 13.1. Problem je u tome što konji ne mogu da lete, pa ćete, prema tome, morati da učinite da ova meto-da **ne radi** ništa, osim kada je u **Pegasusu.**

Listing 13.1: Kada bi konji mogli da lete...

```
// Listing 13.1. Kada bi konji mogli da lete...
// Proceđivanje funkcije Fly() u klasu Horse

linclude  <iostream.h>

class Horse
{
public:
   void Gallop(){ cout « "Galloping...\n"; }
   virtual void Fly() { cout « "Horses can't fly.\n" ; }
private:
   int itsAge;


class Pegasus : public Horse
{
public:
   virtual void Fly() { cout « "I can fly! I can fly! I can fly!\n"


const int NumberHorses = 5;
int main()
{
   Horse* Ranch[NumberHorses];
   Horse* pHorse;
   int choice,!;
   for (i=0; i<NumberHorses; i++)
  '{
      cout « "(l)Horse (2)Pegasus:
      cin » choice;
      if (choice == 2)
         pHorse = new Pegasus;
      else
         pHorse = new Horse;
      Ranch[i] = pHorse;
  }
   cout « "\n";
   for (i=0; i<NumberHorses; i++)
   {
      Ranch[i]->Fly();
      delete Ranch[i];
  }
return 0;
```

```
      Horse (2)Pegasus: 1
   (l)Horse (2)Pegasus: 2
   (l)Horse (2)Pegasus: 1
   (l)Horse (2)Pegasus: 2
   (l)Horse (2)Pegasus: 1

   Horses can't fly.
   I can fly! I can fly! I can fly!
   Horses can't fly.
   I can fly! I can fly! I can fly!
   Horses can't fly.
```

Ш Ш Е ^ $0^{ya}J$ Program će sigurno raditi, mada na račun toga što klasa `Horse` ima Fly() metod. U liniji 10 metod Fly() je obezbeđen `Horseu.` U realnom svetu klasa ovo bi bilo nesuvislo. U liniji 18 klasa `Pegasus` vrši override `Fly` () metod "kako bi uradila pravu stvar", što je ovde predstavljeno u formi vesele poruke.
U liniji 24 upotrebljen je niz `Horse` pointera, da bi demonstrirao da je pozvana ispra-vna Fly() metoda, koja se zasniva na povezivanju u fazi izvršavanja `Horse,` ili `Pegasus` objekata.

" ^ r Filtriranje nagore

Postavljanje zahtevane funkcije bliže vrhu na hijerarhijskoj lestvici klase je uobičajeni postupak za rešavanje ovog problema, koji rezultira nizom funkcija "filtriranih nagore" u baznoj klasi. U takvoj situaciji bazna klasa je u opasnosti da postane globalni naziv za sve funkcije koje može upotrebiti od strane bilo koja izvedena klasa. Ovo može ozbiljno da ugrozi tipiziranje C + + klase i da kreira veliku i kabastu baznu klasu.

Uopšteno govoreći, Vi želite da filtrirate deljenu funkcionalnost "nagore" na hijerarhijskoj lestvici, bez migriranja interfejsa za svaku klasu. To znači da ako dve klase dele zajedničku (istu) baznu klasu (na primer, i konj i ptica dele klasu životinja), Vi ćete poželeti da pomerite ovu funkcionalnost nagore u baznu klasu i da kreirate virtuelnu funkciju.

Ono što ćete, ipak, želeti da izbegnete je filtiranje interfejsa (kao što je, na primer, podizanje Fly funkcije tamo gde ona ne pripada), tako da tu funkciju možete pozvati samo u nekim izvedenim klasama.

Podela uloga

Alternativa ovakvom pristupu, koja se još uvek nalazi u okvirima jednostrukog nasledivanja, je da se Fly () metod zadrži u okviru Pegasus a i da se poziva samo ako pointer zaista ukaže na Pegasus objekat. Da bi ovo proradilo, biće potrebno da Vi možete da upitate Vaš pointer na koji tip on zaista ukazuje. Ovaj postupak je poznat pod imenom Run Time Type Identification (RTTI). Korišćenje RTTI je tek nedavno postalo zvanični deo C + + .

Ako Vaš kompajler ne podržava RTTI, možete ga imitirati, tako što ćete postaviti metodu koja vraća tip svake klase, zatim, možete testirati taj tip u fazi izvršenja i pozvati metod Fly() ako, je on vratio Pegasus.

^iAPOMIMA^ Budite pailjivi kada **dodajete** RTTI svojim klasama. Njegovo korišćenje može biti **indikacija** lošeg dizajna. Razmislite o korišćenju virtuelnih funkcija, templejta, ili višestrukog nasledivanja, umesto RTTI-a.

Medutim, da biste pozvali Fly(), potrebno je da uvedete pointer, koji će red da objekat ukazuje na objekat Pegasus, a ne na Horse. Ovo se naziva *"podela nadole",* s obzirom da ćete objektu Horse dodeliti *nizu* funkciju u izvedenom tipu.

C+ + sada zvanično nudi ovu podelu "nadole", uz pomoć novog Dynamic_Cast operatora. Pogledajte kako to funkcioniše.

Ako imate pointer na baznu klasu, kao što je Horse, i ako ste joj dodelili pointer na izvedenu klasu, kao što je Pegasus, tada možete Horse pointer koristiti polimorfno. Ako zatim, želite da dobijete objekat Pegasus, kreiraćete Pegasus pointer i iskoristićete Dynamic_Cast operator za izvršenje konverzije.

U fazi izvršavanja bazni pointer će biti ispitan. Ako je konverzija odgovarajuća, Vaš novi **Pegasus** pointer će biti ispravan. Ako konverzija nije odgovarajuća, tj. ako Vi, u stvari, niste ni imali **Pegasus** objekat, tada će Vaš novi pointer biti Null. U listingu 13.2 ilustrovana je ova situacija.

Listing 13.2: Konvertovanje.

```
// Listing 13.2 Korišćenje dynamic_cast.
// Korišćenje rtti

linclude <iostream.h>
enum TYPE { HORSE, PEGASUS };

class Horse
{
public:
   virtual void Gallop(){ cout « "Galloping...\n"; }

private:
   int itsAge;
};

class Pegasus : public Horse
(
public:

   virtual void Fly() { cout « "I can fly! I can fly! I can fly!\n"


const int NumberHorses = 5;
int main()
{
   Horse* Ranch[NumberHorses];
   Horse* pHorse;
   int choice,!;
   for (i=0; i<NumberHorses; i++)
   {
      cout « "(l)Horse (2)Pegasus: ";
      cin » choice;
      if (choice == 2)
         pHorse = new Pegasus;
      else
         pHorse = new Horse;
      Ranch[i] = pHorse;
   }
   COUt « "\n";
   for (i=0; i<NumberHorses; i++)
   {
      Pegasus *pPeg = dynamic_cast< Pegasus *> (Ranch[i]);
```

nastavlja se

Listing 13.2: Konvertovanje.

```
            if (pPeg)
               pPeg->Fly();
            else
               cout « "Just a horse\n"

            delete Ranch[i];
         }
      return 0;
```

```
(I)Horse (2)Pegasus: 1
(I)Horse (2)Pegasus: 2
(I)Horse (2)Pegasus: 1
(I)Horse (2)Pegasus: 2
(I)Horse (2)Pegasus: 1

Just a horse
I can fly! I can fly! I can fly!
Just a horse
I can fly! I can fly! I can fly!
Just a horse
```

ANALIZA: Ovo rešenje će, takođe, raditi. `Fly ()` će biti izvan `Horse`-a i neće biti pozvan unutar `Horse` objekata. Kada bude pozvan unutar `Pegasus` objekta, moraće da mu se dodeli eksplicitna uloga; `Horse` objekti ne poseduju metod `Fly()`, tako da pointeru mora biti rečeno da on ukazuje na objekat `Pegasus`, pre nego što bude upotrebljen.

Potreba za dodelom uloge `Pegasus` objektu je upozorenje da nešto može biti neispravno u Vašem dizajnu. Ovaj program efektivno slabi virtuelnu funkciju polimorfizma, s obzirom da on zavisi od uloge objekta prema njegovom realnom tipu u fazi izvršenja.

Dodavanje u dve liste

Drugi problem sa ovim rešenjem je što ste deklarisali da je `Pegasus` jedan tip `Horsea`, tako da objekat `Pegasus` ne možete dodati u listu `Birds`. Platili ste visoku cenu, zato što ste ili prebacili `Fly()` u `Horse`, ili ste dodelili ulogu pointeru, ali još uvek nemate neophodnu i potpunu funkcionalnost.

Jedno rešenje sa jednostrukim nasleđivanjem predstaviće samo sebe. Možete prebaciti `Fly()`, `WhinnyO` i `GallopO` u zajedničku baznu klasu za `Bird` i `Horse: Animal`. Sada, umesto da imate listu za `Birds`, ili za `Horses`, možete imati jedinstvenu listu za `Animals`. Ovo će funkcionisati, ali izgubićete na funkcionalnosti na baznoj klasi.

Alternativno, možete ostaviti ove metode tamo gde jesu i dodeliti uloge za `Horses`, `Birds` i `Pegasus` objekte, ali ovim će se situacija još vise pogoršati.

PAZITE **Koristite pomeranje funkcionalnosti nagore na hijerarhijskoj lestvici naslectivanja.**

Nemojte pomerati interfejs nagore na hijerarhijskoj lestvici naslectivanja.

Izbegnite izmenu tipa objekta u fazi izvršavanja - koristite virtulne metode, templejte i visestruko nasledivanje.

Nemojte dodeliti uloge baznim objektima za izvedene objekte.

Visestruko nasledivanje

Moguće je izvesti novu klasu iz vise različitih baznih klasa. Ovo se naziva *visestruko nasledivanje*. Da biste izvršili izvodenje iz vise baznih klasa, potrebno je da odvojite sve bazne klase zapetama. U listingu 13.3 prikazano je kako se deklariše `Pegasus`, koji se izvodi iz klasa `Horses` i `Birds`. Program, zatim, dodaje objekat `Pegasus` u liste oba tipa.

Listing 13.3: Visestruko nasledivanje.

```
1    // Listing 13.3. Visestruko nasledivanje.
2    // Visestruko nasledivaje
3
4    #include <iostream.h>
5
6    class Horse
7    {
8    public:
9       Horse() { cout « "Horse constructor... "; }
10      virtual -Horse() { cout « "Horse destructor... "; }
11      virtual void WhinnyO const { cout « "Whinny!, . "; }
12   private:
13      int itsAge;
14
15
16   class Bird
17   {
18   public:
19      Bird() { cout « "Bird constructor... "; }
20      virtual -Bird() { cout « "Bird destructor...
21      virtual void Chirp() const { cout « "Chirp.,
22      virtual void F1y() const
23:     I
24         cout « "I can fly! I can fly! I can fly!
25      }
26   private:
27      int itsWeight;
28
29
30   class Pegasus : public Horse, public Bird
```

nastavlja se

SEpp **Usting 13.3: Visestruko naslealvgnje.**

nastavak

```
31  {
32  publi c:
33      void ChirpO const { WhinnyO; }
34      PegasusO  { cout « "Pegasus constructor.
35      -PegasusO  { cout « "Pegasus destructor.     "; }
36
37
38  const int MagicNumber = 2;
39  int  main()
40  {
41      Horse*  Ranch[MagicNumber];
42      Bird* Aviary [MagicNumber];
43      Horse * pHorse;
44      Bird * pBird;
45      int  choice,!';
46      for (i=0; i<MagicNumber; i++)
47      {
48          cout « "\n(l)Horse (2)Pegasus: ";
49          cin » choice;
50          if (choice == 2)
51              pHorse = new Pegasus;
52          else
53              pHorse = new Horse;
54          Ranch[i] = pHorse;
55      }
56      for (i=0; i<MagicNumber; i++)
57      {
58          cout « "\n(l)Bird (2)Pegasus: ";
59          cin » choice;
60          if (choice == 2)
61              pBird = new Pegasus;
62          else
63              pBird = new Bird;
64          Aviary[i] = pBird;
65
66
«7      cout « "\n";
68      for (i=0; i<MagicNumber; i++)
69      {
70          cout « "\nRanch[" « i « "]: "
71          Ranch[i]->Whinny();
72          delete Ranch[i];
73      }
74
75      for (i=0; i<MagicNumber; i++)
76      {
77          cout « "\nAviary[" « i « "]:
78          Aviary[i]->Chirp();
79          Aviary[i]->Fly();
80          delete  Aviary[i];
81:
82:     return 0;
83:  }
```

```
(l)Horse (2)Pegasus: 1
Horse constructor...
(l)Horse (2)Pegasus: 2
Horse constructor... Bird constructor... Pegasus constructor...
(l)Bird (2)Pegasus:
Bird constructor...
(l)Bird (2)Pegasus:
Horse constructor... Bird constructor... Pegasus constructor...

Ranch[0]: Whinny!... Horse destructor...
Ranch[l]: Whinny!... Pegasus destructor...  Bird destructor... Horse
destructor...
Aviary[0]: Chirp... I can fly! I can fly! I can fly! Bird destructor...
Aviary[l]: Whinny!... I can fly! I can fly! I can fly!
Pegasus destructor... Bird destructor... Horse destructor...
Aviary[0]: Chirp... I can fly!
I can fly! I can fly! Bird destructor...
Aviary[l]: Whinny!... I can fly! I can fly! I can fly!
Pegasus destructor.. Bird destructor... Horse destructor...
```

*?xn\ti^a*** U linijama 6-14 deklarisana je klasa Horse. Konstruktor i destruktor su prikazali poruku, dok je metoda Whi nny () odštampala reč Whi nny!

U linijama 16-25 deklarisana je klasa Bird. Zajedno sa svojim konstruktorom i destruktorom, ova klasa poseduje dva metoda, ChirpO i Fly(), koji svaki za sebe, prikazuju identifikacione poruke. U realnim programima, koje ćete raditi, aktivirajte zvučnik, ili generišite animirane slike.

Na kraju, u linijama 30-36 deklarisana je klasa Pegasus. Ona je izvedena i iz klase Horse i iz klase Bird. Klasa Pegasus vrši override ChirpO metoda, kako bi pozvala Whinny () metod koji je nasleden iz klase Horse.

Biće kreirane dve liste: Ranch, sa pointerom na Horse u liniji 41, i Aviary, sa pointerom na klasu Bird u liniji 42. U linijama 46-55 objekti Horse i Pegasus su dodati Ranchu. U linijama 56-65 objekti Bird i Pegasus su dodati Aviaryu.

Pozivanjem virtuelnih metoda Bird pointerom i Horse pointerom Pegasus, objekti će ispravno funkcionisati. Na primer, u liniji 78 članovi Aviary niza su iskorišćeni za poziv ChirpO u objektima na koje ukazuju. Klasa Bird je ovo deklarisala kao virtuelnu metodu, pa će, stoga, za svaki objekat biti pozvana odgovarajuća funkcija.

Primetićete da će svaki put kada se kreira objekat Pegasus izlaz ukazivati na to da su kreirani i Bird i Horse delovi Pegasus objekta. Kada se uništi objekat Pegasus, Bird i Horse delovi će takode biti uništeni, zato što su destruktori virtuelni.

Deklaracija višestrukog nasleđivanja

Deklarišite objekat koji će biti nasleđen iz vise klasa, tako što ćete izlistati bazne klase, iza čijeg imena će slediti dve tačke. Odvojite ove bazne klase zapetama.

Primer **1:**

```
class Pegasus : public Horse, public Bird
```

Primer 2:

```
class Schnoodle : public Schnauzer, public Poodle
```

Delovi visestruko nasleđenih objekata

Kada je objekat `Pegasus` kreiran u memoriji, obe bazne klase, koje su deo `Pegasus` objekta, nastaće na način kako je ilustrovanao na slici 13.1.

Slika 13.1.
Visestruko **nasledeni** objekti

Različite verzije narastaju sa objektima koji imaju višestruke bazne klase. Na primer, šta će se desiti ako dve bazne klase, koje slučajno imaju isto ime, imaju virtuelne funkcije, ili podatke? Kako će se višestruki konstruktori bazne klase inicijalizovati? Šta će se desiti ako su višestruke bazne klase izvedene iz iste klase? U sledećem odeljku biće dati odgovori na ova pitanja i istražićemo kako se visestruko nasledivanje može primeniti.

Konstruktori u visestruko nasleđenim objektima

Ako je `Pegasus` izveden iz `Horse` i `Bird` i svaka od baznih klasa ima konstruktore koji uzimaju neke parametre, `Pegasus` klasa će inicijalizovati ove konstruktore. U listingu 13.4 je ilustrovano kako se to može uraditi.

Listing 13.4: Pozivanje vise konstruktora.

```
1:    // Listing 13.4
2:    // Pozivanje vise konstruktora
3:    linclude <iostream.h>
4:    typedef int HANDS;
5:    enum COLOR { Red, Green, Blue, Yellow, White, Black, Brown };
```



```
enum BOOL { FALSE, TRUE };

class Horse
{
public:
   Horse(C0L0R color, HANDS height);
   virtual -Horse() { cout « "Horse destructor...\n"; }
   virtual void Whinny()const { cout « "Whinny!... ";}
   virtual HANDS GetHeightQ const { return itsHeight; }
   virtual COLOR GetColorQ const { return itsColor; }
private:
   HANDS itsHeight;
   COLOR itsColor;


Horse::Horse(C0L0R color, HANDS height):
   itsColor(col or).itsHeight(height)
{
   cout « "Horse constructor...\n";


class Bird
{
public:
   Bird(COLOR color, BOOL migrates);
   virtual -Bird() {cout « "Bird destructor...\n";  }
   virtual void Chirp()const { cout « "Chirp... ";  }
   virtual void FlyQconst
   {
      cout « "I can fly! I can fly! I can fly! ";
   }
   virtual COLOR GetColorQconst { return itsColor; }
   virtual BOOL GetMigration() const { return itsMigration; }

private:
   COLOR itsColor;
   BOOL itsMigration;


Bird::Bird(COLOR color, BOOL migrates):
   itsColor(color), itsMigration(migrates)
{
   cout « "Bird constructor...\n";
}

class Pegasus : public Horse, public Bird
{
public:
   void Chirp()const { WhinnyO; }
```

nastavlja se

Listing 13.4: Pozivanje vise konstruktora.

```
55        Pegasus(COLOR, HANDS, BOOL.long);
56        -PegasusO {cout « "Pegasus destructor.. An";}
57        virtual long GetNumberBelieversO const
58         {
59            return   itsNumberBelievers;
60
61
62     private:
63        long itsNumberBelievers;
64     };
65
66     Pegasus::Pegasus(
67         COLOR aColor,
68         HANDS height,
69         BOOL migrates,
70         long NumBelieve):
71     Horse(aColor, height),
72     Bird(aColor, migrates),
73      tsNumberBeli evers(NumBeli eve)
74
75        cout « "Pegasus constructor..An"
76
77
78      nt main()
79
80        Pegasus *pPeg = new Pegasus(Red, 5, TRUE, 10);
81        pPeg->Fly();
82        pPeg->Whinny();
83        cout « "\nYour Pegasus is " « pPeg->GetHeight();
84        cout « " hands tall and ";
85        if (pPeg->GetMigration())
86           cout « "it does migrate.";
87        el se
88           cout « "it does not migrate.";
89        cout « "\nA total of " « pPeg->GetNumberBelievers();
90        cout « " people believe it exists.\n";
91        delete pPeg;
92      return 0;
93
```

```
Horse constructor...
Bird constructor...
Pegasus constructor...
I can fly! I can fly! I can fly! Whinny!...
Your Pegasus is 5 hands tall and it does migrate.
A total of 10 people believe it exists.
Pegasus destructor...
Bird destructor...
Horse destructor...
```


nastavak

U linijama 8-19 deklarisana je klasa `Horse`. Konstruktor je preuzeo dva parametra, koja su deklarisana u linijama 5 i 6. Implementacija konstruktora u linijama 21-25, jednostavno, inicijalizuje promenljive i prikazuje poruku.

U linijama 27-43 deklarisana je klasa `Bird` i implementacija njenog konstuktora je u linijama 45-49. `Bird` klasa, takode, uzima dva parametra. Interesantno je da konstuktor za `Horse` preuzima boju dlake (tako da možete razlikovati konje različitih boja), a da konstruktor za `Bird` preuzima boju perja (tako da se ptice sa istobojnim perjem mogu zajedno svrstati). Ovo nas vodi u problem kada poželimo da upitamo `Pegasusa` za njegovu boju, a što ćete videti u sledećem primeru.

Sama klasa `Pegasus` je deklarisana u linijama 51-64, a njeni konstruktori su u linijama 66-72. Inicijalizacija `Pegasus` objekta uključuje tri naredbe. Najpre, inicijalizuje se `Horse` konstruktor sa bojom i visinom. Zatim se inicijalizuje `Bird` konstruktor sa bojom i `Bool ean-om`. Na kraju se inicijalizuje `Pegasus` promenljiva `i tsNumberBel i evers`. Kada je ovaj posao završen, biće pozvano telo `Pegasus` konstruktora.

U `Mine()` funkciji kreiran je i iskorišćen `Pegasus` pointer za pristup funkcijama, koje su članovi baznih objekata.

Dvoznačna rezolucija

U listingu 13.4 klase `Horse` i `Bird`, imaju metod `GetColor()`. Možete poželeti da upitate objekat `Pegasus` za njegovu boju, ali tada će nastati problem: `Pegasus` klasa je nasledena iz klasa `Bird` i `Horse`. Obe ove klase imaju boju i svoje metode za preuzmanje te boje, imaju isto ime i potpis. Ovim se kreira dvoznačnost za kompajler, koju morate da rešite.

Ako napišete

```
COLOR currentColor = pPeg->Horse::GetColor();
```

dobićete grešku u kompilaciji

```
Member is ambiguous: 'Horse::GetColor' and 'Bird::GetColor'
```

Dvoznačnost se može rešiti eksplicitnim pozivom željene funkcije:

```
COLOR currentColor =pPeg ->Hors: :GetColor();
```

Svaki put kada poželite da rešite koja funkcija klase, ili podatak su nasleđeni, možete potpuno odrediti poziv, tako što ćete dodati ime klase baznoj klasi podataka, ili funkciji.

Primetite da ako `Pegasus` izvrši override ove funkcije, problem će takođe biti pomeren, kao što bi trebalo da bude, u Pegasus funkciji:

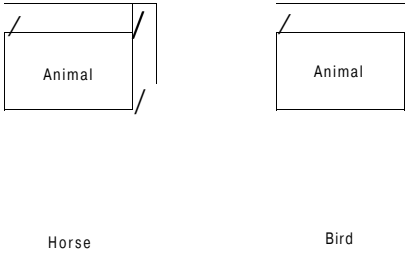
```
virtual COLOR GetColor()const { return Horse::itsColor; }
```

Ovim ste sakrili problem od klijenata Pegasus klase i enkapsulirali ste u Pegasus "znanje" od koje bazne klase on želi da nasledi boju. Klijent je i dalje u mogućnosti da isforsira sledeće:

```
COLOR currentColor = pPeg->Bird::GetColor();
```

Nasledivanje iz deljenih baznih klasa

Šta će se desiti ako su i Bi rd i Horse nasledeni iz zajedničke bazne klase, kao što je Animal ? Na slici 13.2 prikazan je ovaj primer.



Slika 13.2.
Zajedničke bazne klase

Kao što se vidi na slici 13.2, postoje dve bazne klase objekta. Kada su funkcija ili, podatak pozvani iz deljene bazne klase pojavljuje se nova dvoznačnost. Na primer, ako Animal deklariše itsAge kao člana promenljivu i GetAge () kao funkciju, a Vi pozovete pPeg->GetAge(), da li ste mislili da pozovete GetAgeO funkciju, koja je nasledena iz Animal a, na strani Horsea, ili na strani Birda? Ova dvoznačnost se mora rešiti kao što je ilustrovano u listingu 13.5.

Listing 13.5: Opšte osnovne klase.

```
1       // Listing 13.5
2       // Opšte osnovne klase
3       linclude <iostream.h>
4
5       typedef int HANDS;
6       enum COLOR { Red, Green, Blue, Yellow, White, Black, Brown}
7       enum BOOL { FALSE. TRUE };
8
9       class Animal          // zajednička baza za konja i pticu
10:     {
11:     public:
```



```
    Animal(int);
    virtual -Animal() { cout « "Animal destructor...\n"
    virtual int GetAgeO const { return itsAge; }
    virtual void SetAge(int age){ itsAge = age; }
private:
    int itsAge;
};

Animal::Animal(int age):
itsAge(age)
{
    cout « "Animal constructor.. An";


class Horse : public Animal
{
public:
    Horse(C0L0R color, HANDS height, int age);
    virtual -Horse() { cout « "Horse destructor.. An"; )
    virtual void Whinny()const{ cout « "Whinny!... "; }
    virtual HANDS GetHeight() const { return itsHeight; }
    virtual COLOR GetColor() const { return itsColor; }
protected:
    HANDS itsHeight;
    COLOR itsColor;
};

Horse::Horse(C0L0R color, HANDS height, int age):
    Animal(age),
    itsColor(col or).itsHeight(height)

    cout « "Horse constructor..An";


class Bird : public Animal
{
public:
    Bird(C0L0R color, BOOL migrates, int age);
    virtual -Bird() {cout « "Bird destructor..An";   }
    virtual void Chirp()const {cout « "Chirp... ";   }
    virtual void Fly()const
        { cout « "I can fly! I can fly! I can fly! "; }
    virtual COLOR GetColor()const { return itsColor; }
    virtual BOOL GetMigration() const { return itsMigration;
protected:
    COLOR itsColor;
    BOOL itsMigration;
```

nastavlja se

Listing 13.5: Opšte osnovne klase. *nastavak*

```
61:    Bird::Bird(COLOR color, BOOL migrates, int age):
62:        Animal(age),
63:        itsColor(color),   itsMigration(migrates)
64:    {
65:        cout « "Bird constructor...\n";
66:    }
67:
68:    class Pegasus : public Horse, public Bird
69:    {
70:     public:
71:        void Chirp()const{ WhinnyO; }
72:        Pegasus(COLOR, HANDS, BOOL, long, int);
73:        -PegasusO  {cout « "Pegasus destructor...\n";}
74:        virtual long GetNumberBelievers()  const
75:           { return  itsNumberBelievers; }
76:        virtual COLOR GetColor()const { return Horse::itsColor; }
77:        virtual int GetAgeO const { return Horse: :GetAge(); }
78:    private:
79:        long itsNumberBelievers;
80:    };
81:
82:    Pegasus::Pegasus(
83:       COLOR aColor,
84:       HANDS height,
85:       BOOL migrates,
86:       long NumBelieve,
87:       int age):
88:    Horse(aColor, height,age),
89:    Bird(aColor, migrates,age),
90:    itsNumberBelievers(NumBelieve)
91:    {
92:       cout « "Pegasus constructor...\n";
93:    }
94:
95:    int main()
96:    {
97:       Pegasus *pPeg = new Pegasus(Red, 5, TRUE, 10, 2);
98:       int age = pPeg->GetAge();
99:       cout « "This pegasus is " « age « " years old.\n";
100:      delete pPeg;
101:   return 0;
102:
```

```
Animal  constructor...
Horse  constructor...
Animal  constructor...
Bi rd  constructor...
Pegasus  constructor...
This pegasus is 2 years old.
Pegasus destructor.
Bird  destructor...
Animal  destructor..
Horse destructor...
Animal  destructor..
```


Д-ЦМЈј^. Postoji nekoliko interesantnih detalja u ovom listingu. Klasa Animal je deklarisana u linijama 9-18. Klasa Animal je dodala promenljivu itsAge i funkciju SetAgeO.

U liniji 26 deklarisana je klasa Horse, izvedena iz klase Animal. Sada Horse konstruktor ima i treći parametar Age, koji prosleduje baznoj klasi Animal. Primetite da klasa Horse *ne* vrši override funkcije GetAgeO; ona je, jednostavno, nasleduje.

U liniji 46 deklarisana je klasa Bird, koja je izvedena iz klase Animal. Njen konstruktor preuzima parametar Age i koristi ga za inicijalizaciju svoje bazne klase Animal. On, takode, nasleduje funkciju GetAgeO , bez overriding-a.

Pegasus se nasleduje iz Bird i iz Animal i ima dve Animal klase u svom lancu nasledivanja. Ako poželite da pozovete GetAgeO Pegasus objekta, potrebno je da jednoznačno odredite, ili potpuno kvalifikujete metod koji Vam je potreban, ako Pegasus nije izvršio override metoda.

Ovo je rešeno u liniji 76, u kojoj je objekat Pegasus izvršio override GetAgeO, tako da ne uradi ništa vise, osim *da izvrši olančavanje,* kako bi pozvao isti metod bazne klase.

Olančavanje se vrši zbog, dva razloga: ili da bi se izvršilo jednoznačno određivanje bazne klase koja će biti pozvana, kao što je to ovde slučaj, ili da bi se obavio odredeni posao i zatim dozvolilo funkciji bazne klase da ga završi. U nekom trenutku možete poželeti da radite i zatim izvršite olančavanje, ili da prvo izvršite olančavanje, pa, zatim, da radite, dok se ne vratite iz funkcije bazne klase.

Pegasus konstruktor zahteva pet parametara: boju stvorenja, visinu, da li migrira, ili ne, koliko je onih koji veruju u njega, i njegove godine. Ovaj konstruktor vrši inicijalizaciju Horse dela Pegasusa sa bojom, visinom i godinama u liniji 88. On inicijalizuje Bird deo sa bojom, podatkom o tome da li ona migrira i godinama u liniji 89. Na kraju se inicijalizuje i tsNumberBel ievers u liniji 90.

Poziv Horse konstruktora u liniji 88 uvodi implementaciju, koja je prikazana u liniji 39. Konstruktor Horse koristi parametar Age za inicijalizaciju Animal dela Horse dela Pegasusa. On, zatim, prelazi na inicijalizaciju dve Horse promenljive: itsColor i itsAge.

Konstruktor Bird u liniji 89 poziva implementaciju, prikazanu u liniji 46. I ovde se parametar Age koristi za inicijalizaciju Animal dela Birda.

Primetićete da se parametar Color Pegasusa koristi za inicijalizaciju promenljivih i u Birdu i u Horseu. Takođe ćete primetiti da se Age koristi za inicijalizaciju itsAge i u Horseovoj baznoj klasi Animal i u Birdovoj baznoj klasi Animal.

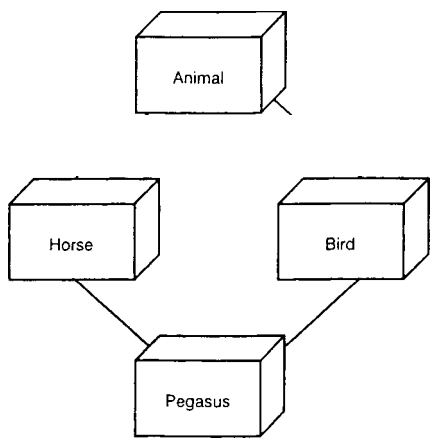
Virtuelno nasledivanje

U listingu 13.5 klasa Pegasus jednoznačno određuje koju će od baznih klasa Animal pozvati. U većini slučajeva odluka koja će se klasa koristiti je diskutabilna - na kraju krajeva, i Horse i Bird imaju potpuno istu baznu klasu.

C++-u se može reći da ne želite dve kopije zajedničke bazne klase, kao što je prikazano na slici 13.2, nego da želite jednu deljenu baznu klasu, kao što je prikazano na slici 13.3.

Ovo možete izvršiti tako što ćete Animal proglasiti za virtuelnu baznu klasu i za Horse i za Bi rd. Klasa Animal se, uopšte, neće menjati. Klase Horse i Bi rd će se menjati samo u delu u kojem će se koristiti izraz Virtual u njihovim deklaracijama. S druge strane, Pegasus će se prilično izmeniti.

Uobičajeno je da konstruktor klase inicijalizuje samo sopstvene promenljive i svoju baznu klasu. Virtuelno nasledene bazne klase su, medutim, izuzetak od ovog pravila. One se inicijalizuju iz svoje *najviše izvedene* klase. Stoga, Animal će biti inicijalizovana ne iz Horsea ili Birda, već iz Pegasusa. Horse i Bird bi trebalo da inicijalizuju Animal u svojim konstruktorima, ali ove inicijalizacije će biti ignorisane kada se kreira objekat Pegasus.



Slika 13.3.
Romboidno
nasledivanje

Listing 13.6 vrši izmenu listinga 13.5, da biste ostvarili prednosti virtuelnog izvodenja.

Listing 13.6: Ilustracija upotrebe virtualnog nasleđivanja.

```
// Listing 13.6
// Virtualno nasledivanje
#include <iostream.h>

typedef int HANDS;
```



```
enum COLOR { Red, Green, Blue, Yellow, White, Black, Brown
enum BOOL { FALSE, TRUE };

class Animal          // opšta baza za konja i pticu
{
public:
   Animal(int);
   virtual CAnimal(){ cout « "Animal destructor...\n"; }
   virtual int GetAgeO const { return itsAge; }
   virtual void SetAge(int age) { itsAge = age; }
private:
   int itsAge;
};

Animal::Animal(int age):
itsAge(age)
{
   cout « "Animal constructor...\n";


class Horse : virtual public Animal
{
public:
   Horse(C0L0R color, HANDS height, int age);
   virtual čHorse(){ cout « "Horse destructor...\n"; }
   virtual void Whinny()const { cout « "Whinny!... ";
   virtual HANDS GetHeight() const { return itsHeight;
   virtual COLOR GetColor() const { return itsColor; }
protected:
   HANDS itsHeight;
   COLOR itsColor;
};

Horse::Horse(C0L0R color, HANDS height, int age):
   Animal(age),
   itsColor(color),itsHeight(height)
{
   cout « "Horse constructor...\n";


class Bird : virtual public Animal
{
public:
   Bird(COLOR color, BOOL migrates, int age);
   virtual -Bird() (cout « "Bird destructor..An";   }
   virtual void Chirp()const { cout « "Chirp... ";   }
   virtual void Fly()const
      { cout « "I can fly! I can fly! I can fly! "; }
   virtual COLOR GetColor()const { return itsColor; }
```

nastavlja se

Listing 13.6: Ilustracija upotrebe virtualnog nasledivanja. *nastavak*

```
55      virtual BOOL GetMigrationQ const { return itsMigration;}
56    protected:
57      COLOR itsColor;
58      B00L itsMigration;
59
60
61    Bird::Bird(COLOR color, B00L migrates, int age):
62      Animal(age),
63      itsColor(color), itsMigration(migrates)
64    I
65      cout « "Bird constructor...\n";
66
67
68    class Pegasus : public Horse, public Bird
69    {
70    publi c:
71      void Chirp()const { WhinnyO; }
72      Pegasus(COLOR, HANDS, B00L, long, int);
73      -PegasusO {cout « "Pegasus destructor..An";}
74      virtual long GetNumberBelievers() const
75        { return itsNumberBelievers; }
76      virtual COLOR GetColor()const { return Horse::itsColor;
77    private:
78      long itsNumberBelievers;
79
80
81    Pegasus::Pegasus(
82      COLOR aColor,
83      HANDS height,
84      B00L migrates,
85      long NumBelieve,
86      int age):
87    Horse(aColor, height,age),
88    Bird(aColor, migrates,age),
89    Animal(age*2),
90    i tsNumberBeli evers(NumBeli eve)
91    {
92      cout « "Pegasus constructor.. An"
93
94:
95;   int main()
96    {
97      Pegasus *pPeg = new Pegasus(Red, 5, TRUE, 10, 2);
98      int age = pPeg->GetAge();
99      cout « "This pegasus is " « age « " years old.\n"
100     delete pPeg;
101   return 0;
102
```


IZLAZ

```
Animal constructor...
Horse constructor...
Bird constructor...
Pegasus constructor...
This pegasus is 4 years old.
Pegasus destructor...
Bird destructor...
Horse destructor...
Animal destructor...
```

t-WjI^fc U liniji 26 `Horse` se deklariše kao da je *virtuelno* nasleden iz `Animal a`. U liniji 46 napravljena je ista deklaracija za `Bird`. Primetićete da konstruktori za `Bird` i `Animal` i dalje vrše inicijalizaciju `Animal` objekta.

`Pegasus` se nasleduje iz `Birda` i iz `Animal a`. I kao što je, uglavnom, slučaj sa izvedenim objektima iz `Animal a`, on takođe vrši inicijalizaciju `Animal a`. Medutim, ovo je inicijalizacija `Pegasusa`, tako da će pozivi konstruktora za `Animal` iz `Birda` i `Horsea` biti ignorisani. Ovo možete videti, s obzirom da je prosledena vrednost 2 od strane `Horsea` i `Bi rda`, dok ju je `Pegasus` duplirao. Rezultat, broj 4, prikazan je u izlazu iz linije 99.

`Pegasus vise ne mora da rešava` višeznačnost `poziva GetAgeO`, pa `je, stoga, Slobodan` da, jednostavno, nasledi ovu funkciju `iz Animal a`. Primetićete da ovo nije slučaj sa `GetCol or` (), s obzirom da se ova funkcija nalazi u obe bazne klase, a ne i u `Animal u`.

Deklarisanje klasa za virtuelno nasledivanje

Da biste obezbedili da izvedena klasa ima samo jednu verziju zajedničke bazne klase, deklarišite intermediate klase, koje će se virtuelno nasledivati iz bazne klase.

Primer 1:

```
class Horse : virtual public Animal
class Bird : virtual public Animal
class Pegasus : public Horse, public Bird
```

Primer2:

```
class Schnauzer : virtual public Dog
class Poodle : virtual public Dog
class Schnoodle : public Schnauzer, public Poodle
```

Problemi sa višestrukim nasleđivanjem

Iako visestruko nasledivanje nudi veliki broj prednosti u odnosu na jednostuko nasledivanje, vecina C++ programera ga nerado koristi. Problemi sa kojima se oni susreću i koje često navode su da mnogi kompajleri još uvek ne podržavaju višetruko nasledivanje, pronalaženje grešaka je otežano i da sve što može biti urađeno sa višestrukim nasledivanjem uglavnom može biti urađeno i bez njega.

Sve ovo je sasvim ispravno i Vi ćete, verovatno, poželeti da zaštitite Vaše programe od nepotrebne kompleksnosti. Neki debageri imaju dosta problema sa višestrukim nasleđivanjem, dok, s druge strane, postoje projektanti koji nepotrebno pojačavaju složenost programa korišćenjem višestrukog nasledivanja kada ono nije neophodno.

<j| **PASTE j£** Koristite visestruko nasledivanje kada nova klasa zahteva funkcije i osobine iz vise od jedne bazne klase.

Koristite virtuelno nasledivanje kada vedna izvedenih klasa mora da ima tačno jednu kopiju **deljene** bazne klase.

Koristite inicijalizaciju deljene bazne klase iz najvise izvedene klase kada koristite virtuelne bazne klase.

Nemojte koristiti visestruko nasledivanje kada jednostruko nasledivanje može da Vam "završi posao."

Mixins i Capabilities klase

Jedan od načina da iznadete srednje rešenje između jednostrukog i višestrukog nasledivanja je korišćenje onoga što se naziva mixins. Ovde možete imati Vašu Horse klasu, izvedenu iz Animal i Displayable. Di si playable će dodati nekoliko metoda za prikazivanje bilo kojeg metoda na ekranu.

^BŽ^'frU *Mixin,* ili Capability klasa je klasa koja dodaje funkcionalnost, bez dodavanja puno podataka, ili, uopšte, bez dodavanja podataka.

Capability klase su izmešane u izvedene klase, kao i bilo koja druga klasa. Deklarisanjem izvedene klase, nasledićete njen javni deo. Jedina razlika izmedu Capabilty i bilo koje druge klase je da Capabilty nema podatke, ili ih ima sasvim malo. Ovo je prečica koja omogućava da sve osobine budu uvek na raspolaganju, bez potrebe za komplikovanjem izvedenih klasa.

Ovim ćete nekim debagerima olakšati posao u radu sa Mixinima, što nebi bio slučaj kada biste radili sa visestruko nasledenim objektima. Takode, postoji manja verovatnoća za pojavljivanje višeznačnosti u pristupu podacima iz drugih glavnih baznih klasa.

Na primer, akoje Horse izveden iz Animal i iz Displayable, Displayable ne bi trebalo da ima podatke. Animal bi trebalo da bude baš ono što je uvek bio, tako da se svi podaci za Horse izvode iz Animal, ali se funkcije za Horse izvode iz obe klase.

Izraz Mixin nam dolazi iz poslastičarnice u Samervilu,(Masačusets), gde se bombone i kolači mešaju sa osnovnim ukusima sladoleda. Ovo se čini kao dobra metafora nekim objektno-orijentisanim programerima, koji su "iskoristili" ceo godišnji odmor za rad sa objektno-orijentisanim programskim jezikom SCOOPS.

Apstraktni tipovi podataka

Često ćete imati potrebu da kreirate hijerarhiju klasa. Na primer, možete kreirati klasu Shape i iz nje izvesti klase Rectangle i Circle. Iz Rectangle možete izvesti Square kao specijalan slučaj za Rectangle.

Svaka od izvedenih klasa će izvršiti override Draw() metoda, GetArea () metoda i tako dalje. U listingu 13.7 prikazana je implementacija Shape klase i njenih izvedenih Ci rcle i Rectangle klasa.

Listing 13.7: Shape klase.

```
//Listing 13.7. Shape klase.

#include <iostream.h>

enum BOOL{ FALSE, TRUE };

class Shape
{
public:
10    ShapeOO
ii    -ShapeOO
12    virtual long GetArea() { return -1; } // gre ška
13    virtual long GetPerim() { return -1; }
14    virtual void Draw() {}
15 private:
16 };
17
18 class Circle : public Shape
19 {
20 publi c:
21       Circle(int radius):itsRadius(radius){}
22       ~Circle(){}
23       long GetArea() { return 3 * itsRadius * itsRadius;
24       long GetPerim() { return 9 * itsRadius; }
25       void Draw();
26 private:
27    int itsRadius;
28    int itsCircumference;
29 };
30
31; void Circle::Draw()
32 {
33    cout « "Circle drawing routine here!\n";
34
35
36
37 class Rectangle : public Shape
```

nastavlja se

Listing 13.7: Shape klase. *nastavak*

```
{
public:
      Rectangle(int len, int width):
         itsLength(len), itsWidth(width){}
      -Rectangle(){}
      virtual long GetAreaO { return itsLength * itsWidth; }
      virtual long GetPerimQ {return 2*itsLength + 2*itsWidth; }
      virtual int GetLengthQ { return itsLength; }
      virtual int GetWidthQ { return itsWidth; }
      virtual void Draw();
private:
   int itsWidth;
   int itsLength;


void Rectangle::Draw()
{
   for (int i = 0; i<itsLength; i++)
   {
      for (int j = 0; j<itsWidth; j++)
         cout « "x ";

      cout « "\n";



class Square : public Rectangle
{
public:
      Square(int len);
      Square(int len, int width);
      -Square(){}
      long GetPerim() {return 4 * GetLength();}
};

Square::Square(int 1en):
   Rectangle(len.len)
{}

Square::Square(int len, int width):
   Rectangle(len,width)


   if (GetLengthQ != GetWidth())
      cout « "Error, not a square... a Rectangle??\n"


int main()
```



```
86:
87:       int choice;
88:       B00L fQuit = FALSE;
89:       Shape * sp;
90:
91:       while (1)
92:       {
93:          cout « "(l)Circle (2)Rectangle (3)Square (0)Quit:
94:          cin » choice;
95:
96:          switch (choice)
97:          {
98:             case 1: sp = new Circle(5);
99:             break;
100:            case 2: sp = new Rectangle(4,6);
101:            break;
102:            case 3: sp = new Square(5);
103:            break;
104:            default: fQuit = TRUE;
105:            break;
106:         }
107:         if (fQuit)
108:            break;
109:
110:         sp->Draw();
111:         cout « "\n";
112:      }
113:   return 0;
114:
```

```
MULI^*  (l)Circle (2)Rectangle (3)Square (0)Quit: 2
        x x x x x x
        x x x x x x
        x x x x x x
        x x x x x x

        (l)Circle (2)Rectangle (3)Square (0)Quit:3
        x x x x x
        x x x x x
        x x x x x
        x x x x x
        x x x x x

        (l)Circle (2)Rectangle (3)Square (0)Quit:0
```

^ linijama 7-16 deklarisana je klasa Shape. Metodi GetAreaO i GetPerimO vraćaju poruku o grešci, dok Draw() metoda ne radiništa.Na kraju krajeva, šta, uopšte, znači nacrtati oblik? Jedino tipovi oblika (krugovi, pravougaonici, itd) mogu biti nacrtani, dok oblik, kao njihova apstrakcija, ne može.

Klasa **Circle** se izvodi **iz** klase **Shape i** vrši override tri virtuelna metoda. Primetite **da** nema potrebe **za** dodavanjem reči "virtual", pošto je ona deo njenih naslednika. Međutim, neće **biti** pogrešno **ako to** učinimo, kao **što** je prikazano u klasi **Rectangle,** u linijama 43, 44 **i** 47. Pametno je izraz "virtual" uključuti kao podsetnik, dokumentovanja programa.

Square se izvodi iz **Rectangle i** on, takode, vrši override GetPerimQ metoda, nasledujući, pritom, preostale metode koji su definisani u Rectangle.

Postoji problem ako korisnik pokuša da skrati **Shape** oblik, **a** Vi poželite da mu to zabranite. **Shape** klasa postoji samo da bi obezbedila interface klasama koje su izvedene **iz** nje i stoga se ona naziva Abstact Data Type (ADT).

> *Abstract Data Type* predstavlja koncept (kao što je oblik), pre nego objekat (kao što je krug). U C++ **-U** ADT je uvek bazna klasa drugim klasama.

v

Ciste virtuelne funkcije

C++ Vam omogućava kreiranje apstraktnih tipova podataka uz pomoć čistih virtuelnih funkcija. Virtuelna funkcija će biti čista, ako je inicijalizovana nulom, kao u sledećm primeru:

```
virtual void Draw() = 0;
```

Svaka klasa sa jednom, ili vise virtuelnih funkcija je ADT i nije dozvoljeno da instantizujemo objekat klase koja je ADT. Pokušaj da se ovo učini dovešće do greške u trenutku kompilacije. Smeštanjem čistih virtuelnih funkcija u Vašu klasu, signaliziraćete klijentima Vaše klase:

- Nemojte praviti objekte ove klase, izvedite ih iz nje.
- Proverite da li ste izvršili override čiste virtuelne funkcije.

Svaka klasa koja se izvodi iz ADT nasleduje čistu virtuelnu funkciju kao čistu i, stoga, mora da izvrši override svih čistih virtuelnih funkcija, ako želi da instantizuje objekte. Zbog toga, ako se Rectangle nasleduje iz Shape, i Shape ima tri čiste virtuelne funkcije. Rectangle mora da izvrši override sve tri funkcije, inače će i on postati ADT. U listingu 13.8 ponovo je napisana Shape klasa, koja je ADT. Da bismo sačuvali prostor, preostali deo listinga 13.7 ovde nije prikazan. Zamenite deklaraciju Shape klase u listingu 13.7, linija 7-16, deklaracijom klase Shape iz listinga 13.8 i ponovo startujte program.

Listing 13.8: Abstraktni tipovi podataka.

```
1:  class Shape
2:  {
3:  public:
4:        Shape(){}
5:        ~Shape(){}
6:        virtual long GetAreaO = 0; // greška
7:        virtual long GetPerim() = 0;
8:        virtual void Draw() = 0;
9:  private:
10: };
```

```
ТЦЦЛ^ OKircle (2)Rectangle (3)Square (0)Quit: 2
          x x x x x x
          x x x x x x
          x x x x x x
          x x x x x x

          (1)Circle (2)Rectangle (3)Square (0)Quit: 3
          x x x x x
          x x x x x
          x x x x x
          x x x x x
          x x x x x

          (1)Circle (2)Rectangle (3)Square (0)Quit: 0
```

|r:m|^jp^ Kao što možete videti, rad ovog programa je potpuno nepromenjen. Jedina razlika je što sada nije moguće napraviti objekat klase `Shape`.

Apstraktni tipovi podataka

Deklarišite klasu kao apstraktni tip podataka, tako što ćete u deklaraciju klase ukljuciti jednu, ili vise čistih virtuelnih funkcija. Deklarišite jednu virtuelnu funkciju, tako što ćete napisati = 0 posle deklaracije funkcije.
Primer:

```
class Shape
{
virtual void Draw() = 0;      // čista virtuelna
```

Implementacija čistih virtuelnih funkcija

Tipično čiste virtuelne funkcije u apstraktnim baznim klasama se nikada ne implementiraju. Pošto za taj tip klasa nikada neće biti kreiran ni jedan objekat, ne postoji razlog da se uvede implementacija i ADT će raditi čisto kao definicija interfejsa za objekte koji se iz nje izvode.

Međutim, moguće je obezbediti implementaciju čiste virtuelne funkcije. Funkcija bi tada mogla biti pozvana iz objekata koji su izvedani iz ADT; recimo, mogla bi da obezbedi zajedničku funkcionalnost za sve funkcije nad kojima je izvršen override. Listing 13.9 je sličan listingu 13.7 i opisana je klasa Shape kao ADT, a ovaj put je implementirana čista virtuelna funkcija Draw(). Klasa Circle vrši override funkcije

DrawQ, kao što je o neophodno, ali se ona, zatim, povezuje sa funkcijom bazne klase zarad dodatne funkcionalnosti.

U ovom primeru dodatna funkcionalnost se ogleda u jednostavnoj dodatnoj poruci koja će biti odštampana, ali će nam ona dočarati da bazna klasa ovezbeduje deljeni mehanizam za crtanje bi, recimo, mogao da bude i crtanje prozora, koji bi sve izvedene klase mogle da koriste.

Listing 13.9: Implementiranje dstih virtuelnih funkcija.

```
1:    //Implementiranje čistih virtuelnih funkcija
2:
3:    linclude <iostream.h>
4:
5:    enum BOOL { FALSE, TRUE };
6:
7:    class Shape
8:    {
9:    public:
10:      Shape(){}
11:      -ShapeOO
12:      virtual long GetAreaO = 0; // greška
13:      virtual long GetPerim()= 0;
14:      virtual void Draw() = 0;
15:   private:
16:   };
17:
18:    void Shape::Draw()
19:   {
20:      cout « "Abstract drawing mechanism!\n";
21:   }
22:
23:   class Circle : public Shape
24:   {
25:   public:
26:         Circle(intradius):itsRadius(radius){}
27:         -CircleOO
28:         long GetAreaO { return 3 * itsRadius * itsRadius; }
29:         long GetPerimQ { return 9 * itsRadius; }
30:         void Draw();
31:   private:
32:      int itsRadius;
33:      int itsCircumference;
34:   };
35:
36:   void Circle::Draw()
37:   {
38:      cout « "Circle drawing routine here!\n";
39:      Shape::Draw();
40:   }
```



```
class Rectangle : public Shape
{
public:
      Rectangle(int len, int width):
         itsLength(len), itsWidth(width){}
      -Rectangle()0
      long GetAreaO { return itsLength * itsWidth; }
      long GetPerimO {return 2*itsLength + 2*itsWidth; }
      virtual int GetLength() { return itsLength; }
      virtual int GetWidth() { return itsWidth; }
      void Draw();
private:
   int itsWidth;
   int itsLength;


void Rectangle::Draw()
{
   for (int i = 0; i<itsLength; i++)
   {
      for (int j = 0; j<itsWidth; j++)
         cout « "x ";

      cout « "\n";
   }
   Shape::Draw();



class Square : public Rectangle
{
public:
      Square(int len);
      Square(int len, int width);
      -Square()0
      long GetPerimO (return 4 * GetLength();


Square::Square(int len):
   Rectangle(len,len)
{}

Square::Square(int len, int width):
   Rectangle(len,width)


   if (GetLengthO != GetWidthO)
      cout « "Error, not a square... a Rectangle??\n";
```

nastavlja se

Listing 13.9: Implementiranje cistih virtuelnih funkcija.

```
91    }
92
93    int main()
94    {
95       int choice;
96       BOOL fQuit = FALSE;
97       Shape * sp;
98
99       while (1)
100      {
101         cout « "(I)Circle (2)Rectangle (3)Square (O)Quit:
102         cin » choice;
103
104         switch (choice)
105         {
106            case 1: sp = new Circle(5);
107            break;
108            case 2: sp = new Rectangle(4,6);
109            break;
110            case 3: sp = new Square (5);
111            break;
112            default: fQuit = TRUE;
113            break;
114         }
115         if (fQuit)
116            break;
117
118         sp->Draw();
119         cout « "\n";
120      }
121   return 0;
122
```

```
(I)Circle (2)Rectangle (3)Square (O)Quit: 2
x x x x x x
x x x x x x
x x x x x x
x x x x x x
Abstract drawing mechanism!

(I)Circle (2)Rectangle (3)Square (O)Quit: 3
x x x x x
x x x x x
x x x x x
x x x x x
x x x x x
Abstract drawing mechanism!

(I)Circle (2)Rectangle (3)Square (O)Quit: 0
```


nastavak

U linijama 7-16 deklarisan je apstraktni tip podataka Shape. Pomoću sva ^ш^ш^ш^ met0(ja^ k0ji su deklarisani kao čisto virtuelni, primetićete da ovo nije neophodno. Ako je bilo koji od ovih metoda deklarisan kao čisto virtuelan, klasa bi, u svakom slučaju, postala ADT.

Metodi GetAreaO i GetPerimO nisu implementirani, dok metod Draw() jeste. Klase Circle i Rectange vrše override metoda Draw() i obe se povezuju na bazni metod, dobijajući, time, prednost deljene funkcionalnosti u baznoj klasi.

Složena hijerarhija i apstrakcija

Vremenom, Vi ćete izvoditi ADT-e iz drugih ADT-a. Ovo se može dogoditi kada poželite da neke od izvedenih čistih virtuelnih funkcija ne budu čiste, a da preostale budu.

Ako kreirate klasu Animal, možete napraviti Čiste virtuelne funkcije Eat(), SleepO, Move() i Reproduce(). Pretpostavimo da iz klase Animal dalje izvodite klase Mammal i Fish.

U daljem ispitivanju odlučili ste da se svi Mammal i reprodukuju na isti način i stoga ste uČinili da funkcija Mammal::Reproduce() vise nije čista, dok su funkcije Eat(), SleepO i Move() i dalje čiste virtuelne funkcije.

Iz Mammal a ste, zatim, izveli Dog. I Dog mora da izvrši override i implementaciju preostale tri virtuelne funkcije, kako biste mogli da napravite objekte tipa Dog.

Šta biste rekli kao dizajner klase kada ne postoje Animal i, ili Mammal i, koji mogu biti kreirani, a da svi Mammal i mogu da naslede obezbeden Reproduce() metod bez njegovog override-a?

Listing 13.10 prikazuje ovu tehniku.

Listing 13.10: Izvodenje ADT-a iz drugih ADT-a.

```
// Listing 13.10
// Izvodenje ADT-a iz drugih ADT-a
linclude <iostream.h>

enum COLOR { Red, Green, Blue, Yellow, White, Black, Brown
enum B00L { FALSE, TRUE };

class Animal          // zajednička baza za konja i pticu

public:
   Animal(int);
   virtual  -Animal() { cout « "Animal destructor...\n"
   virtual  int GetAgeO const { return itsAge; }
   virtual  void SetAge(int age) { itsAge = age; }
   virtual  void SleepO const = 0;
   virtual  void Eat() const = 0;
```

nastavlja se

Listing 13.10: Izvodenje ADT-a iz drugih ADT-a.

nastavak

```
   virtual void ReproduceO const
   virtual void Move() const = 0;
   virtual void SpeakQ const = 0;
private:
   int itsAge;


Animal::Animal(int age):
itsAge(age)
{
   cout « "Animal constructor.. An";


class Mammal : public Animal
{
publi c:
   Mammal(int age):Animal(age)
      { cout « "Mammal constructor...\n";}
   -Mammal() { cout « "Mammal destructor...\n";}
   virtual void ReproduceO const
       { cout « "Mammal reproduction depicted.. An";


class Fish : public Animal
{
public:
   Fish(int age):Animal(age)
      { cout « "Fish constructor..An";}
   virtual -Fish() {cout « "Fish destructor...\n";  }
   virtual void SleepO const { cout « "fish snoring...\n"
   virtual void Eat() const { cout « "fish feeding.. An";
   virtual void ReproduceO const
      { cout « "fish laying eggs...\n"; }
   virtual void Move() const
       { cout « "fish swimming..An";   }
   virtual void Speak() const{ }


class Horse : public Mammal
{
publi c:
   Horse(int age, COLOR color ):
   Mammal(age), itsColor(color)
         { cout « "Horse constructor..An"; }
   virtual -Horse() { cout « "Horse destructor.. An";
   virtual void Speak()const { cout « "Whinny!... \n";
   virtual COLOR GetItsColorO const { return itsColor;
   virtual void SleepO const
```



```
65:          { cout « "Horse snoring..An"; }
66:     virtual void Eat() const { cout « "Horse feeding.. An";
67:     virtual void Move() const { cout « "Horse running.. An"
68:
69:  protected:
70:     COLOR itsColor;
71:
72:
73:  class Dog : public Mammal
74:  {
75:  public:
76:     Dog(int age, COLOR color ):
77:        Mammal(age), itsColor(color)
78:           { cout « "Dog constructor..An"; }
79:     virtual -Dog() { cout « "Dog destructor..An"; }
80:     virtual void Speak()const { cout « "Whoof!... \n"; }
81:     virtual void SleepO const {cout « "Dog snoring.. An"; }
82:     virtual void Eat() const { cout « "Dog eating.. An"; }
83:     virtual void Move() const { cout « "Dog running.. An"; }
84:     virtual void ReproduceO const
85:         { cout « "Dogs reproducing..An"; }
86:
87:  protected:
88:     COLOR itsColor;
89:
90:
91:  int main()
92:  {
93:     Animal *pAnimal=0;
94:     int choice;
95:     B00L fQuit = FALSE;
96:
97:     while (1)
98:     {
99:        cout « "(l)Dog (2)Horse (3)Fish (0)Quit:
100:        cin » choice;
101:
102:        switch (choice)
103:        {
104:           case 1: pAnimal = new Dog(5,Brown);
105:           break;
106:           case 2: pAnimal = new Horse(4,Black);
107:           break;
108:           case 3: pAnimal = new Fish (5);
109:           break;
110:           default: fQuit = TRUE;
111:           break;
112:        }
113:        if (fQuit)
```

nastavlja se

nastavak

Listing 13.10: Izvodenje ADT-a ii drugih ADT-a.

```
114             break;
115
116          pAnimal->Speak();
117          pAnimal->Eat();
118          pAnimal->Reproduce();
119          pAnimal->Move();
120          pAnimal->Sleep();
121          delete pAnimal;
122          cout « "\n";
123
124       return 0;
125
```

```
(I)Dog (2)Horse (3)Bird (O)Quit: 1
Animal constructor...
Mammal constructor...
Dog constructor...
Whoof!...
Dog eating...
Dog reproducing——
Dog running...
Dog snoring...
Dog destructor...
Mammal destructor...
Animal destructor...

(I)Dog (2)Horse (3)Bird (O)Quit: 0
```

. U linijama 8-22 deklarisan je apstraktni tip podataka Animal. Animal poseduje virtuelnu funkciju koja nije čista ItsAge(), podeljenu izmedu svih Animal objekata. Ona, takode, ima pet cistih virtuelnih funkcija: SleepO, Eat(), ReproduceO, Move() i Speak().

Mammal je izveden iz Animal a, i deklarisana je u linijama 30-38 i nije dodala nikakve nove podatke. Medutim, ona je izvršila override funkcije Reproduce (), obezbeđujući, time, zajednički način reprodukcije za sve sisare. Fish, takođe, mora da izvrši overide funkcije ReproduceO, s obzirom da se Fish izvodi direktno iz Animal a i ne može preuzeti prednosti reprodukcije sisara (što je inače dobra stvar).

Klase Mammal nemaju vise potrebu za daljim override-om funkcije ReproduceO, ali su slobodne da to urade ako žele, kao što je to uradila klasa Dog u liniji 84. Klase Fi sh, Horse i Dog vrše override čistih virtuelnih funkcija, tako da njihovi objekti mogu biti kreirani.

U telu programa jedan pointer Animal se koristi da bi ukazao na različite izvedene objekte, jedan za drugim. Pozvane su virtuelne metode i baziraju se na povezivanju pointera u trenutku izvršavanja i tada će biti pozvana ispravna metoda izvedene klase.

Doći će do provere greške u kompilaciji, ako pokušate da kreirate Animal, ili Mammal objekat, s obzirom da su obe klase apstraktni tipovi podataka.

Koji tipovi su apstraktni?

U nekom programu klasa Animal je apstraktna, u drugom nije. Šta odreduje kada klasa treba da bude apstraktna, a kada ne? Odgovor na ovo pitanje nije odreden nikakvim faktorom iz realnog sveta, već iz smisla u Vašem programu. Ako pišete program koji opisuje farmu, ili zoo-vrt, možete poželeti da Animal bude apstraktni tip podataka, ali da klasa Dog bude klasa iz koje ćete kreirati objekte. S druge strane, ako pravite neku animaciju, možete poželeti da klasa dog bude apstraktni tip podataka, a da tipovi pasa, kao što su retrajveri, terijeri i drugi, to ne budu. Nivoi apstrakcije su funkcija, u od koje će zavisiti kako finalno kreirate Vaše tipove.

PAOTI §• Koristite apstraktne tipove podataka, da biste obezbedili zajedničku funkcionalnost za ved broj klasa koje su u relaciji. Izvršite override svih cistih virtuelnih funkcija. Sve funkcije za koje morate da izvršite override neka budu takode virtuelne. Nemojte kreirati objekat u klasi koja je apstraktnog tipa podataka.

Seme posmatranja

Veoma popularan trend u C + + -u je kreiranje *šema dizajniranja*. Ovo je dobro dokumentovano rešenje opštih problema, koje su uradili C + + programed. Na primer, šema posmatranja rešava opšte probleme u nasledivanju.

Pretpostavimo da razvijate tajmer klasu koja zna da broji protekle sekunde. Pošto klasa može da ima svoj clan ItsSeconds koji je celobrojan, ona takode mora da ima metode da ga postavljaju, preuzimaju i uvećavaju.

Hajde da sada pretpostavimo da Vaš program želi da bude informisan svaki put kada je tajmer ItsSeconds uvećan. Jedno očigledno rešenje je da ugradite metod za obaveštavanje u tajmer. Međutim, obaveštavanje nije deo tajminga i kompleksan kod za registrovanje tih klasa, koje žele da budu informisane o promeni vremena, ne pripada Vašoj tajmer klasi.

Što je još važnije, kada jednom rešite logiku registracije svih onih koji su zainteresovani za ove promene i kada ih obavestite, poželećete da apstahujete sve to u posebnu klasu kako biste mogli da je ponovo iskoristite sa drugim klasama, koje mogu biti posmatrači sa ove tačke gledišta.

Stoga je bolje rešenje kreirati klasu posmatrača. Neka ovaj posmatrač bude ADT sa čisto virtuelnom funkcijom Update().

Sada kreirajte drugi apstaktni tip podataka pod nazivom Subject. On će sadržati niz Observer objekata i takođe će obezbediti dva metoda: Register(), koji dodaje posmatrača u listu, i Noti fy (), koji će biti pozvan kada ima šta da se prijavi. Ove klase, koje žele da budu obaveštene o tajmeru, promeniće naslede iz Observera. Sam tajmer će biti nasleden iz Subjecta. Observer klasa će registrovati samu sebe u Subject klasi. Klasa Subject će pozvati Noti fy kada dode do izmene u trenutku kada se ažurira tajmer.

3 > *

Na kraju, primetićete da neće svaki klijent želeti da bude posmatrač i stoga smo kreirali novu klasu, pod nazivom ObservedTimer, koja nasleduje i iz tajmera i iz Subject-a. Ovim ste ObservedTimer-u da, li karakteristike tajmera i mogućnost da bude posmatran.

Još nešto o višestrukom nasleđivanju, apstraktnim tipovima podataka i Javai

Većina C + + programera misle da je Java bazirana većim delom na C + + da su kreatori Javae nameravali da izostave visestruko nasledivanje. Njihovo mišljenje je bilo da visestruko nasledivanje dovodi do kompleksnosti, koja je u suprotnosti sa lakim i jednostavnim korišćenjem Javae. Oni su osetili da mogu da zadovolje 90 odsto funkcionalnosti višestrukog nasledivanja, korišćenjem nečega što su nazvali interfejsi.

:ш∧&IIIJ *Interfejs* je veoma sličan apstraktnom tipu podataka time što se definiše kao set funkcija, koje se isključivo mogu implementirati u izvedenim klasama. Medutim, sa interfejsovima Vi ne vršite direktno izvodenje iz interfejsa. Vi to izvodenje vršite iz druge klase, a interfejs implementirate, kao i u višestrukom nasledivanju. Stoga, ovaj "brak" apstraktnih tipova podataka i višestrukog nasledivanja Vam pruža nešto slično Capabilty klasi, bez složenosti višestrukog nasledivanja. S obzirom da interfejsovi ne mogu imati implementacije, kao ni članove podataka, eliminisana je potreba za virtuelnim nasledivanjem.

Ovo može biti bag, ili osobina, u zavisnosti od posmatrača. U svakom slučaju, ako ste razumeli visestruko nasledivanje i apstraktne tipove podataka u C + + -u bićete u poziciji da koristite naprednije funkcije Javae, ako odlučite da naučite taj jezik.

Rezime

Danas ste naučili kako da prevazidete neka ograničenja u oblasti jednostrukog nasledivanja. Naučili ste koliko je opasno vršiti filtriranje interfejsa "nagore" na hijerarhijskoj lestvici nasledivanja i koliki je rizik od podele uloga "nadole" na istoj toj lestvici. Dalje, naučili ste kako da koristite princip višestrukog nasledivanja, do kojih problema može da dovede visestruko nasledivanje i kako te probleme rešiti uz pomoć virtuelnog nasledivanja.

Upoznali ste i apstraktn tipove podataka i način na koji se kreiraju apstraktne klase, uz pomoć čistih virtuelnih funkcija. Naučili ste kako da implementirate čiste virtuelne funkcije i kada to možete da učinite. Na kraju, videli ste kako se implementira šema posmatranja, uz pomoć principa višestrukog nasledivanja i apstraktnih tipova podataka.

Pitanja i odgovori

P Šta znači filtriranje funkcionalnosti "nagore"?

O Ovo se odnosi na ideju o pomeranju deljene funkcionalnosti "nagore" u zajedničku baznu klasu. Ukoliko vise od jedne klase deli funkciju, poželjno je pronaći zajedničku baznu klasu, u koju ta funkcija može da se smesti.

P **Da** li je filtriranje "nagore" uvek dobro rešenje?

0 Jeste, ukoliko filtrirate *deljenu* funkcionalnost "nagore". Ali nije, ukoliko je predmet tog pomeranja *interfejs*. Ako sve izvedene klase nisu u mogućnosti da koriste metodu, onda je takvo pomeranje na gore u zajedničku baznu klasu pogrešan izbor. Medutim, ako se za njega opredelite, moraćete da promenite tip objekta u fazi izvršenja, pre nego što odlučite da li ćete moći da pozovete funkciju.

P Zbog čega je promena tipa objekta u fazi izvršenja lose rešenje?

0 U velikim programima, switch komande postaju glomazne i teske za održavanje. Svrha virtuelnih funkcija je da Vam omoguće da u virtuelnoj tabeli odredite tip objekta u fazi izvršavanja, radije nego da to učini programer.

P Zbog čega deljenje uloga nije srećno rešenje?

0 Deljenje uloga nije loša solucija, ako se izvede na siguran način. Ako je pozvana funkcija koja zna da objekat mora da bude odredenog tipa, deljenje uloga za ovaj tip je sasvim u redu. Deljenje uloga može da se upotrebi, da bi oslabio jaki tip čekiranja u C + + -u i onda je to ono što želimo da izbegnemo. Ako vršite izmenu tipa objekta u fazi izvršavanja i, zatim, delite uloge pointera, to može biti znak upozorenja da nešto nije u redu sa Vašim dizajnom.

P Zašto ne bi trebalo **da** sve funkcije budu virtuelne?

O Virtuelne funkcije podržava tabela virtuelnih funkcija, koja uvodi overhead u fazi izvršenja, kako u pogledu veličine programa, tako i u pogledu njegovih performansi. Ako imate sasvim male klase i ne očekujete da one imaju potklase, nećete poželeti da bilo koju od funkcija učinite virtuelnom.

P **Kada** bi destruktor trebalo **da** postane virtuelan?

0 Uvek kada mislite da će klasa biti razdeljena na potklase, a pointer za baznu klasu bude upotrebljen za pristup nekom objektu potklase. I, po pravilu, ako ste bilo koju funkciju u Vašoj klasi učinili virtuelnom, u svakom slučaju, uradite isto to i sa destruktorom.

P Zašto se zamarati kreiranjem apstraktnih tipova podataka - zašto ih, jednostavno, ne napraviti ne-apstaktnim i izbeći kreiranje bilo kakvog objekta tog tipa?

O Svrha većine konvencija u C++-u je da omogući kompajleru da pronade bagove, kako biste ih izbegli u fazi izvršavanja, u kodu koji ćete predati na korišćenje korisnicima. Ako klasu učinite apstraktnom,odnosno, ako jojdodelite čiste virtuelne funkcije, kompajler će označiti svaki objekat koji je kreiran iz tog apstaktnog tipa kao grešku.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Šta je podela uloga "nadole"?
2. Štajev-ptr?
3. Ako zaobljeni pravougaonik ima prave ivice (stanice) i oble uglove i Vaša `RoundRect` klasa nasledi obe osobine i od `Rectangle` i od `Circle`, a zatim i od `Shape`, koliko će `Shapeova` biti kreirano, kada kreirate `RoundRect`?
4. Ako se `Horse` i `Bird` nasleduju od `Animal a`, koristeći public virtuelno nasledivanje, da li njihovi konstruktori inicijalizuju `Animal` konstruktor? Ako se `Pegasus` nasleduje i iz `Horse` i iz `Bird`, kako on inicijalizuje `Animalov` konstruktor?
5. Deklarišite klasu `Vechi cl e` i napravite je da bude apstraktni tip podataka.
6. Ako je bazna klasa neki ADT i ima tri čiste virtuelne funkcije, nad koliko od njih mora da se izvrši override u njenim izvedenim klasama?

Vežbe

1. Prikažite deklaraciju za klasu `JetPlane, koja` nasleduje `od Rocket i Airpl ane`.
2. Prikažite deklaraciju za 747, `koja` nasleduje `od JetPl ane` klase, opisane u Vežbi
3. Napišite program `koji izvodi Car i Bus` iz `klase Vechicle`. Neka `Vechicle` bude ADT sa dve `čiste` virtuelne funkcije,a neka `Car i Bus` ne `budu ADT`.
4. Izmenite program `iz` Vežbe 3, `tako` da `Car` bude ADT `i izvedite SportCar, Wagon` i `Coupe iz Car`. U klasi `Car` obezbedite implementaciju `za` jednu `čistu` virtuelnu funkciju u `Vechi cle,` koja neće `biti čista` (virtuelna funkcija).

Specijalne klase i funkcije

C++ nudi mnogobrojne načine za ograničavanje opsega delovanja i uticaja promenljivih i pointera. U daljem tekstu videćete kako se kreiraju globalne promenljive, promenljive lokalnih funkcija, pointeri za promenljive i promenljive članova klase. Danas ćete nauciti:

- šta su statičke promenljive članovi i statičke funkcije članovi
- kako se koriste statičke promenljive članovi i statičke funkcije članovi
- kako se kreiraju i kako se manipuliše pointerima na funkcije i pointerima na funkcije članove
- kako se radi sa nizovima pointera na funkcije.

Statički podaci članovi

Do sada ste, verovatno, smatrali da su podaci jedinstveni za jedan objekat i da su nedeljivi izmedu objekata u okviru klase. Na primer, ako imate pet `Cat` objekata, gde je svaka mačka stara odredeni broj godina, ima svoju težinu i ostale podatke. Broj godina jedne ne utiče na godine starosti druge.

Medutim, dešava se da želite da pratite neki skup podataka. Na primer, možda ćete poželeti da znate koliko je objekata specifične klase kreirano u Vašem programu i koliko njih još uvek postoji. Statičke promenljive članovi su deljive između svih kopija te klase. One su kompromisno rešenje izmedu globalnih podataka, koji stoje na raspolaganju svim delovima Vašeg programa, i podataka članova, koji su, obično, na raspolaganju samo pojedinačnim objektima.

Za statičke članove možete misliti da pre pripadaju klasi, nego objektu. Za svaki pojedinačni objekat postoji jedan podatak clan, dok na nivou klase postoji samo jedan statički član. Listing 14.1 deklariše Cat objekat sa statičkim podatkom članom HowManyCats. Ova promenljiva prati koliko je Cat objekata kreirano. Ovo se postiže uvećavanjem statičke promenljive, HowManyCats, određenom konstrukcijom i njenim umanjivanjem, uz pomoć destrukcije.

Listing 14.1: Statički podaci clanovi.

```
1   //Listing 14.1 statički podaci članovi
2
3   linclude <iostream.h>
4
5   class Cat
6   {
7   public:
8      Cat(int age):itsAge(age)}HowManyCats++; }
9      virtual -Cat() } HowManyCats-;  }
10     virtual int GetAgeO { return itsAge; }
11     virtual void SetAge(int age) { itsAge = age; }
12     static int HowManyCats;
13
14  private:
15     int itsAge;
16
17  };
18
19  int Cat::HowManyCats = 0;
20
21  int main()
22  {
23     const int MaxCats = 5; int i;
24     Cat *CatHouse[MaxCats];
25     for (i = 0; i<MaxCats; i++)
26        CatHouse[i ] = new Cat(i);
27
28     for (i = 0; i<MaxCats; i++)
29     {
30        cout « "There are ";
31        cout « Cat::HowManyCats;
32        cout « " cats left!\n";
33        cout « "Deleting the one which is
34        cout « CatHouse[i]->GetAge();
35        cout « " years old\n";
36        delete CatHouse[i1;
37        CatHouse[i] = 0;
38     }
39   return 0;
40  }
```



```
There are 5 cats left!
Deleting the one which is 0 years old
There are 4 cats left!
Deleting the one which is 1 years old
There are 3 cats left!
Deleting the one which is 2 years old
There are 2 cats left!
Deleting the one which is 3 years old
There are 1 cats left!
Deleting the one which is 4 years old
```

I.KMU/:^ U linijama 5-17 je deklarisana prosta klasa Cat. U liniji 12 HowManyCats je deklarisano kao statička promenljiva clan tipa int.

Deklaracija (funkcije) HowManyCats ne definiše integer i ne zauzima dodatni prostor. Za razliku od nestatičkih promenljivih članova, ne zauzima dodatni prostor kreiranjem Cat objekta, pošto se HowManyCats promenljiva clan ne nalazi u objektu. Stoga je u liniji 19 promenljiva definisana i inicijalizovana.

Ovo je uobičajena greška, kada se zaboravi na defmisanje statičke promenljive člana klase. Ne dozvolite da se ovo i Varna dogodi! Naravno, ukoliko se i desi, linker će je "uloviti" i javiti sledeću poruku o grešci:

```
undefined symbol Cat::HowManyCats
```

Ovo nije potrebno da radite za i tsAge, budući da je u pitanju nestatička promenljiva clan, koja se definiše svaki put kada kreirate Cat objekat, što ste učinili ovde, u liniji 26.

Konstruktor za Cat povećava statičku promenljivu člana u liniji 8. Destruktor ga smanjuje u liniji 9. Stoga, u ovom trenutku, HowManyCats ima preciznu informaciju koliko je Cats objekata bilo kreirano, a još uvek nije uništeno.

Driver program u linijama 20-40 kreira pet mačaka i smešta ih u jedan niz. Ovaj poziva pet Cat konstruktora i zato je HowManyCats uvećano pet puta, u odnosu na svoju početnu vrednost 0.

Program zatim provlad petlju kroz svaku od pet pozicija u nizu i prikazuje vrednost HowManyCats, pre nego što je izbrisan tekući Cat pointer. U izlazu se vidi da je početna vrednost 5 (na kraju krajeva, 5 je konstruisano) i svaki put kada se startuje petlja ostaje po jedna mačka (Cat) manje.

Primetite da je HowManyCats javno i da mu se pristupa direktno iz `main()`. Nema razloga da ovu promenljivu člana prikazujemo na ovaj način. Bilo bi uputnije da ga kreirate kao privatnog, zajedno sa ostalim promenljivim članovima, i da omogućite public accessor metod, dokle god budete pristupali podacima preko kopije Cata. Sa druge strane, ukoliko budete želeli da pristupate ovim podacima direktno, a da Vam za to nije neophodno da imate Cat objekat, na raspolaganju su Vam dve opcije: ili da ga ostavite da bude public (javan), kao što je prikazano u listingu 14.2, ili da obezbedite statičku funkciju člana, o čemu će biti reči nešto kasnije u ovom poglavlju.

Listing 14.2: Pristup stoticTrim clanovima bez objekta.

```
//Listing 14.2 statički podaci članovi

linclude <iostream.h>

class Cat
{
publ ic:
   Cat(int age):itsAge(age){HowManyCats++; }
   virtual -Cat() { HowManyCats-; }
   virtual int GetAgeO { return itsAge; }
   virtual void SetAge(int age) { itsAge = age; }
   static int HowManyCats;

private:
   int itsAge;

};

int Cat::HowManyCats = 0;

void TelepathicFunctionQ;

int main()
{
   const int MaxCats = 5; int i;
   Cat *CatHouse[MaxCats];
   for (i = 0; i<MaxCats; i++)
   {
      CatHouse[i] = new Cat(i);
      TelepathicFunctionQ;
   }

   for ( i = 0; i<MaxCats; i++)
   {
      delete CatHouse[i ];
      TelepathicFunctionQ;
   }
   return 0;
}

void TelepathicFunctionQ
{
   cout « "There are ";
   cout « Cat::HowManyCats « cats alive!\n"
}
```

```
There are 1 cats alive!
There are 2 cats alive!
There are 3 cats alive!
There are 4 cats alive!
There are 5 cats alive!
There are 4 cats alive!
There are 3 cats alive!
There are 2 cats alive!
There are 1 cats alive!
There are 0 cats alive!
```

Listing 14.2 ima puno sličnog sa listingom 14.1, osim što mu je pridodata nova funkcija, TelepathicFunction(). Ova funkcija ne kreira Cat objekat, ona, čak, i ne uzima Cat objekat za parametar, ali ipak može da pristupi HowManyCats promenljivoj članu. Treba napomenuti da se ova promenljiva clan ne nalazi ni u jednom odredenom objektu; ona je član u celini i, ako je javna (public), može joj se pristupiti iz bilo koje funkcije u programu.

Ova promenljiva člana može da bude i i privatna. Ako to učinite, mod ćete da joj pristupite iz funkcije člana, ali, u torn slučaju, moraćete da imate objekat te klase. U listingu 14.3 prikazan je ovaj pristup. Alternativna statička funkcija clan je objašnjena odmah posle analize tog listinga.

Listing 14.3: Pristup stotičkim članovima korišćenjem nestatickih funkcija članica.

```
1:  //Listing 14.3 privatni statički podaci članovi
2:
3:  linclude <iostream.h>
4:
5:  class Cat
6:  {
7:  public:
8:     Cat(int age):itsAge(age){HowManyCats++; }
9:     virtual -Cat() { HowManyCats-; }
10:    virtual int GetAgeO { return itsAge; }
11:    virtual void SetAge(int age) { itsAge = age; }
12:    virtual int GetHowManyQ { return HowManyCats; }
13:
14:
15: private:
16:    int itsAge;
17:    static int HowManyCats;
18: };
19:
20: int Cat::HowManyCats = 0;
21:
22: int main()
23: {
24:    const int MaxCats = 5; int i;
25:    Cat *CatHouse[MaxCatsl;
26:    for (i = 0; i<MaxCats; i++)
27:       CatHouse[i ] = new Cat(i);
```

nastavlja se

Listing 14.3: Pristup stahckim clanovima korišćenjem nestatičkih funkcija clanica.

```
28:
29:       for (i = 0; i<MaxCats;
30:       {
31:          cout « "There are ";
32:          cout « CatHouse[i]->GetHowMany();
33:          cout « " cats left!\n";
34:          cout « "Deleting the one which is "
35:          cout « CatHouse[i]->GetAge()+2;
36:          cout « " years old\n";
37:          delete CatHouse[i ];
38:          CatHouse[i] = 0;
39:       }
40:    return 0;
41:    }
```

```
There are 5 cats left!
Deleting the one which is 2 years old
There are 4 cats left!
Deleting the one which is 3 years old
There are 3 cats left!
Deleting the one which is 4 years old
There are 2 cats left!
Deleting the one which is 5 years old
There are 1 cats left!
Deleting the one which is 6 years old
```

U liniji 17 statička promenljiva clan HowManyCats je deklarisana tako da ima private pristup. Sada ovoj promenljivoj ne možete pristupiti iz funkcija koje nisu članovi, kao što je, na primer, Tel epathi c funkcija iz prethodnog listinga.

Čak i da je promenljiva HowManyCats statička, ona se i dalje nalazi unutar opsega klase. Bilo koja funkcija klase, kao na primer, GetHowManyQ , može joj pristupiti, baš kao što i funkcije-clanovi mogu da pristupe bilo kojem podatku-članu. Međutim, funkcija koja bi pozvala GetHowManyQ morala bi da ima objekat preko koga bi izvršila poziv funkcije.

*gj **PA2TIT** t^- Koristite statičke promenljive članove, da biste delili podatke duž svih kopija klasa.

Neka statičke promenljive članovi budu Protected, ili Private, ako želite da ogrartičite pristup njiima.

Nemojte koristiti statičke promenljive clanove za čuvanje podataka jednog objekta. Statički podaci clanovi su del j i vi izmedu svih objekata njihove klase.

Statičke funkcije članovi

Statičke funkcije članovi su slične statičkim promenljivim clanovima. One ne postoje samo u jednom objektu, već u opsegu čitave klase. Stoga, mogu biti pozvane,)aez potrebe da postoji objekat u toj klasi, a što je ilustrovano u listingu 14.4.

Listing 14.4: Statičke funkcije članice.

```
//Listing 14.4 statički podaci članovi

linclude <iostream.h>

class Cat
{
publi c:
   Cat(int age):itsAge(age){HowManyCats++; }
   virtual -Cat() { HowManyCats-; }
   virtual int GetAgeO { return itsAge; }
   virtual void SetAge(int age) { itsAge = age; }
   static int GetHowManyQ { return HowManyCats; }
private:
   int itsAge;
   static int HowManyCats;

};

int Cat::HowManyCats = 0;

void TelepathicFunctionQ;
int mainQ
{
   const int MaxCats = 5;
   Cat *CatHouse[MaxCats]; int i;
   for (i = 0; i<MaxCats; i++)
   {
      CatHouse[i ] = new Cat(i);
      TelepathicFunctionQ;
   }

   for ( i = 0; i<MaxCats; i++)
   {
      delete CatHouse[i ];
      TelepathicFunctionQ;
   }
   return 0;
}

void TelepathicFunctionQ
{
   cout « "There are " « Cat::GetHowMany() « " cats alive!\n"
}
```

```
There are 1 cats alive!
There are 2 cats alive!
There are 3 cats alive!
There are 4 cats alive!
There are 5 cats alive!
```

```
There are 4 cats alive!
There are 3 cats alive!
There are 2 cats alive!
There are 1 cats alive!
There are 0 cats alive!
```

Statička funkcija clan HowManyCats je u liniji 15 deklarisana da ima Private pristup. Publ ic funkcija GetHowManyQ je deklarisana kao Publ ic i Static u liniji 12.

S obzirom da je GetHowMany deklarisana kao Public, njoj može pristupiti bilo koja funkcija, a budući da je ona i Static, nema potrebe da imate objekat tipa Cat kome biste uputili poziv. Stoga je u liniji 42 funkcija Tel epathi cFunct i on () u mogućnosti da pristupi Public Static funkciji, iako nema pristup Cat objektu. Naravno, funkciju GetHowManyQ mogli ste da pozovete i iz main(), kao i bilo koju drugu funkciju.

^МАИМШЦ* Statičke funkcije clanovi nemaju pointer This. Stoga, one ne mogu biti deklarisane kao const. Takode, s obzirom da se promenljivim clanovima podataka pristupa iz funkcija clanova, korišćenjem pointera Thi s, statičke funkcije članovi ne mogu pristupiti ni jednoj nestatičkoj promenljivoj članu.

Statičke funkcije članovi

Statičkim funkcijama clanovima možete pristupiti tako što ćete ih pozvati iz jednog objekta klase, kao što to činite sa bilo kojim funkcijama clanovima, ili ih možete pozvati bez ikakvog objekta, tako što ćete u potpunosti kvalifikovati klasu i ime objekta.

Primer:

```
class Cat
{
public:
static int GetHowManyQ { return HowManyCats; }
pri vate:
static int HowManyCats;
};
int Cat::HowManyCats = 0;
int mainQ
{
int howMany;
Cat theCat;                          // definiše mačku
howMany = theCat.GetHowMany();       // pristup kroz objekat
howMany = Cat::GetHowMany();         // pristup bez objekta
}
```


Pointeri na funkcije

Kao što je ime niza pointer na prvi element niza, ime funkcije je constant pointer na funkciju. Moguće je deklarisati promenljivu pointer, koja ukazuje na funkciju, i, zatim, pozvati funkciju, korišćenjem tog pointera. Ovo može biti veoma korisno, pošto Vam omogućava da kreirate programe za odlučivanje koju funkciju treba pozvati, a na osnovu korisničkog ulaza.

Jedini problem sa pointerima funkcija je razumevanje tipa objekta na koji se ukazuje. Pointer na int ukazuje na celobrojnu promenljivu, a pointer na funkciju mora da ukazuje na funkciju sa odgovarajućim tipom, koji će biti vraćen, i potpisom.

```
U deklaraciji
long (* funcPtr) (int);
```

funcPtr je deklarisan da bude pointer (primetićete * ispred imena), - on ukazuje na funkciju koja uzima celobrojni parametar, a vraća long. Zagrade oko *funcPtr su neophodne, s obzirom da zagrade oko i nt povezuju mnogo čvršće, zato što one imaju visi prioritet od operatora (*). Bez prvih zagrada, deklarisali biste funkciju koja za parametar uzima integer, a vraća pointer na long (primetićete da prazna mesta ovde ne znače ništa).

```
Ispitajte sledeče dve deklaracije:
long * Function (int);
long (* funcPtr) (int);
```

Prva deklaracija, FunctionQ, je funkcija, koja uzima integer i vraća pointer na promenljivu tipa long. Druga deklaracija, funcPtr, je pointer na funkciju, koja uzima integer i vraća promeljivu tipa long.

Deklaracija pointera funkcije će uvek uključivati tip koji će biti vraćen, kao i zagrade koje indiciraju tip parametara, ako postoje. U listingu 14.5 ilustrovani su deklaracija i korišćenje pointera - pokazivača na funkcije.

Listing 14.5: Pokazivači na funkcije.

```
1:    // Listing 14.5 Korišćenje funkcijskih pokazivača
2:
3:    linclude <iostream.h>
4:
5:    void Square (int&,int&);
6:    void Cube (int&, int&);
7:    void Swap (int&, int &);
8:    void GetVals(int&, int&);
9:    void PrintValsQnt, int);
10:   enum BOOL { FALSE, TRUE };
11:
12:   int mainQ
13:   {
```

nastavlja se

Listing 14.5: Pokazivači na funkcije. *nastavak*

```
14      void (* pFunc) (int &, int &);
15      BOOL fQuit = FALSE;
16
17      int val0ne=l, valTwo=2;
18      int choice;
19      while (fQuit == FALSE)
20      {
21         cout « "(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap:
22         cin » choice;
23         switch (choice)
24         {
25            case 1: pFunc = GetVals; break;
26            case 2: pFunc = Square; break;
27            case 3: pFunc = Cube; break;
28            case 4: pFunc = Swap; break;
29            default : fQuit = TRUE; break;
30         }
31         if (fQuit)
32            break;
33
34         PrintVals(val0ne, valTwo);
35         pFunc(val0ne, valTwo);
36         PrintVals(val0ne, valTwo);
37      }
38   return 0;
39
40
41   void PrintVals(int x, int y)
42
43      cout « "x: " « x « " y: " « y « endl;
44
45
46   void Square (int & rX, int & rY)
47
48      rX *= rX;
49      rY *- rY;
50
51
52   void Cube (int & rX, int & rY)
53
54      int tmp;
55
56      tmp = rX;
57      rX *= rX;
58      rX = rX * tmp;
59
60      tmp = rY;
61
62      rY *= rY;
63      rY = rY * tmp;
64   }
65
66   void Swap(int & rX, int & rY)
67   {
68      int temp;
69      temp = rX;
70      rX = rY;
71      rY = temp;
72   }
73
74   void GetVals (int & rValOne, int & rValTwo)
75   {
76      cout « "New value for ValOne: ";
77      cin » rValOne;
78      cout « "New value for ValTwo: ";
79      cin » rValTwo;
80   }
```

Ш Л А>

```
(O)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 1

x: 1 y: 2
New value for ValOne: 2
New value for ValTwo: 3
x: 2 y: 3
(0)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 3
x: 2 y: 3
: 8 y: 27
(0)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 2
x: 8 y: 27
x: 64 y: 729
(0)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 4
x: 64 y: 729
x: 729 y: 64
(0)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 0
```

§P-'^^|^ U linijama 5-8 deklarisane su četiri funkcije (sve sa istim tipom podataka, koji će biti vraćen, i potpisom), koje vraćaju void, a uzimaju dve reference na integer-e.

U liniji `14 pFunc` je deklarisana da bude pointer na funkciju koja vraća void i uzima dve integer reference kao parametre. Na bilo koju od prethodnih funkcija može se ukazati uz pomoć `pFunc`. Korisniku se u petlji nudi da izabere funkciju koju želi, a, u zavisnosti od izbora, `pFunc` dobija odgovarajuću vrednost. U linijama 35-36 prikazuje se tekuća vrednost ova dva integer-a, tekuće dodeljena funkcija se poziva i zatim se vrednosti ponovo prikazuju.

Pointer na funkciju

Pointer na funkciju se poziva, baš kao i funkcija na koju on ukazuje, osim što se, umesto imena pointera na funkciju, koristi naziv funkcije.

Dodelite pointer na funkciju određenoj funkciji, tako što ćete dodeliti ime funkciji bez zagrade. Ime funkcije je konstantan pointer na nju samu. Koristite pointer na funkciju, baš kao što biste koristili i ime funkcije.

Pointer na funkciju se mora slagati u vraćenim vrednostima i potpisu sa funkcijom, kojoj ste ga dodelili.

Primer:

```
long (*pFuncOne) (int, int);
long SomeFunction (int, int);
pFuncOne = SomeFunction;
pFunc0ne(5,7);
```

Zašto koristiti pointere funkcije?

Sasvim je moguće napisati program sličan onome iz listinga 14.5, bez pointera funkcija, ali korišćenje ovih pointera će objasniti namere i korišćenje programa će biti direktnije: pokupite funkciju iz liste i zatim je pozovite.

Listing 14.6 koristi prototipove funkcija i definicije iz listinga 14.5, ali telo programa ne koristi pointere funkcija. Ispitajte razlike između ova dva listinga.

yMPOMINA^ Da biste iskompajlirali ovaj program, iskopirajte linije 41-80 iz listinga 14.5, odmah iza linije 56.

Listing 14.6: Prepisan Listing 14.5 bez pokazivaca na funkciju.

```
1:    // Listing 14.6 Bez funkcijskih pokazivaca
2:
3:    linclude <iostream.h>
4:
5:    void Square (int&,int&);
6:    void Cube (int&, int&);
7:    void Swap (int&, int &);
8:    void GetVals(int&, int&);
9:    void PrintVals(int, int);
10:   enum BOOL { FALSE, TRUE };
11:
12:   int main()
13:   {
14:      BOOL fQuit = FALSE;
15:      int va!0ne=I, valTwo=2;
16:      int choice;
17:      while (fQuit == FALSE)
18:      {
```



```
cout « "(O)Quit (I)Change Values (2)Square (3)Cube (4)Swap:
cin » choice;
switch (choice)
{
   case 1.
      PrintVals(valOne, valTwo);
      GetVals(valOne, valTwo);
      PrintVals(valOne, valTwo);
      break;

   case 2:
      PrintVals(valOne, valTwo);
      Square(valOne,valTwo);
      PrintVals(valOne, valTwo);
      break;

   case 3:
      PrintVals(valOne, valTwo);
      Cube(valOne, valTwo);
      PrintVals(valOne, valTwo);
      break;

   case 4:
      PrintVals(valOne, valTwo);
      Swap(valOne, valTwo);
      PrintVals(valOne, valTwo);
      break;

      default :
      fQuit = TRUE;
      break;


   if (fQuit)
      break;
}
return 0;
}
```

```
(O)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 1
x: 1 y: 2
New value for ValOne: 2
New value for ValTwo: 3
(O)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 3
x: 2 y: 3
x: 8 y: 27
(O)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 2
x: 8 y: 27
x: 64 y: 729
(O)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 4
```

```
x: 64 y: 729
x: 729 y: 64
(O)Quit (I)Change Values (2)Square (3)Cube (4)Swap: 0
```

jffIEjlfei Implementacija funkcija je izostavljena, s obzirom da je identična onoj ^ u listingu 14.5. Kao što možete da vidite, i izlaz je nepromenjen, ali je telo programa prošireno od linije 27 do linije 38. Pozivi za PrintValsQ se moraju ponavljati za svaki pojedinačni slučaj.

Bilo je moguće staviti Print Val s() na početak Whi le petlje, a zatim i na dno, umesto u svaku Case naredbu. Ovim biste pozvali PrintValsO, čak i u slučaju Exit, a to nije bio deo specifikacije.

Na stranu to što je došlo do povećanja veličine koda i do ponavljanja poziva koji rade isti posao - još gore je što je izgubljena opšta čitljivost programa. Ovo je, naravno, namešten slučaj, koji je kreiran da bi se prikazalo kako funkcionišu pointeri na funkciju. U realnim uslovima, prednosti su čak mnogo jasnije: pointeri na funkcije mogu da eliminišu duplirani kod, da razjasne Vaš program i omoguće Vam da napravite tabele funkcija koje ćete pozivati, u zavisnosti od uslova u fazi izvršenja.

Skraćeno pozivanje

Pointed na funkcije ne moraju da budu dereferencirani, iako to možete da učinite. Stoga, ako je `pFunc` pointer na funkciju koja uzima integer i vraća promenljivu tipa `1 ong`, i Vi dodelite `pFunc` odgovarajućoj funkciji. Tada tu funkciju možete pozvati bilo sa

```
DFunc(x);
bilo sa
(*pFunc)(x);
```

Ove dve forme su identične, odnosno, prva je samo skraćena verzija druge.

Nizovi pointera na funkcije

Baš kao i niz pointera na integer-e, možete da deklarišete i niz pointera na funkcije, koje vraćaju specifičan tip podataka sa specifičnim potpisom. Listing 14.7 je još jedna verzija listinga 14.5, koja ovoga puta koristi niz za poziv svih mogućih izbora u jednom trenutku.

yWPOMENA^, Da biste kompajlirali ovaj program, smestite linije 41-80 iz listinga 14.5 odmah iza linije 39.

Listing 14.7: Demonstrira upotrebu niza pokazivaca na funkcije.

```
// Listing 14.7 demonstrira upotrebu niza pokazivaca na funkcije

#include <iostream.h>

void Square (int&,int&);
void Cube (int&, int&);
```



```
void Swap (int&, int &);
void GetVals(int&, int&);
9:   void PrintVals(int, int);
10:  enum BOOL { FALSE, TRUE };
11:
12:  int main()
13:  {
14:     int val0ne=l, valTwo=2;
15:     int choice, i;
16:     const MaxArray = 5;
17:     void (*pFuncArray[MaxArray])(int&, int&);
18:
19:     for (i=0;i<MaxArray;i++)
20:     {
21:        cout « "(l)Change Values (2)Square (3)Cube (4)Swap:
22:        cin » choice;
23:        switch (choice)
24:        {
25:           case l:pFuncArray[i ] = GetVals; break;
26:           case 2:pFuncArray[ i ] = Square; break;
27:           case 3:pFuncArray[i ] = Cube; break;
28:           case 4:pFuncArray[i] = Swap; break;
29:           default:pFuncArray[i]  = 0 ;
30:        }
31:     }
32:
33:     for (i=0;i<MaxArray; i++)
34:     {
35:        pFuncArray[i] (valOne,valTwo);
36:        PrintVals(val0ne,valTwo);
37:     }
38:  return 0;
39:  }
```

```
(l)Change Values (2)Square (3)Cube (4)Swap: 1
(l)Change Values (2)Square (3)Cube (4)Swap: 2
(l)Change Values (2)Square (3)Cube (4)Swap: 3
(l)Change Values (2)Square (3)Cube (4)Swap: 4
(l)Change Values (2)Square (3)Cube (4)Swap: 2
New Value for ValOne: 2
New Value for ValTwo: 3
x: 2 y: 3
x: 4 y: 9
x: 64 y: 729
x: 729 y: 64
x: 7153 y:4096
```

Još jednom je implementacija funkcija izostavljena, radi uštede prostora; ista je kao ona iz listinga 14.5. U liniji 17 niz `pFuncArray` je deklarisan kao niz od pet pointera na funkcije, koji vraća void i (pre)uzima dve celobrojne reference.

U linijama 19-31 od korisnika je txazeno da izabere funkciju koja će biti pozvana i svakom članu niza je dodeljena adresa odgovarajuće funkcije. U linijama 33-37 funkcije se pozivaju jedna za drugom. Posle svakog poziva prikazuje se rezultat.

Prosleđivanje pointera na funkcije drugim funkcijama

Pointeri na funkcije (kao i nizovi pointera na funkcije) mogu biti prosledeni drugim funkcijama, koje mogu da preuzmu posao, tako što će, uz pomoć pointera, pozvati ispravnu funkciju.

Na primer, listing 14.5 možete poboljšati tako što ćete proslediti izabrani pointer funkcije drugoj funkciji (izvan mai n()), koja će prikazati vrednosti, pozvati funkciju i, na kraju, ponovo prikazati vrednosti. Listing 14.8 ilustruje ovu varijaciju.

ozotENJ|| Za kompajliranje ovog programa smestite linije 46-80 iz listinga 14.5 odmah iza linije 45.

Listing 14.8: Predavanje pokazivaca na funkcije kao argumente funkcije.

```
1:    // Listing 14.8 Bez funkcijskih pokazivaca
2:
3:    linclude <iostream.h>
4:
5:    void Square (int&,int&);
6:    void Cube (int&, int&);
7:    void Swap (int&, int &);
8:    void GetVals(int&, int&);
9:    void PrintVals(void (*)(int&, int&),int&, int&);
10:   enum BOOL { FALSE, TRUE };
11:
12:   int main()
13:   {
14:      int va!0ne=l, valTwo=2;
15:      int choice;
16:      BOOL fQuit = FALSE;
17:
18:      void (*pFunc)(int&, int&);
19:
20:      while (fQuit == FALSE)
21:      {
22:         cout « "(O)Quit (l)Change Values (2)Square (3)Cube (4)Swap: ";
23:         cin » choice;
24:         switch (choice)
25:         {
26:            case l:pFunc = GetVals; break;
27:            case 2:pFunc = Square; break;
28:            case 3:pFunc = Cube; break;
29:            case 4:pFunc = Swap; break;
30:            default:fQuit = TRUE; break;
```



```
31:         }
32:         if (fQuit    TRUE)
33:            break;
34:         PrintVals ( pFunc, valOne, valTwo);
35:
36:
37:   return 0;
38:   }
39:
40:   void PrintVals( void (*pFunc)(int&, int&),int& x, int& y)
41:
42:      cout « «x: - « x « - y : " « y « endl;
43:      pFunc(x,y);
44:      cout « Mx: N « x « " v.- » « y « endl;
45:   }
```

```
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 1
x: 1 y: 2
New value for ValOne: 2
New value for ValTwo: 3
x: 2 y: 3
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 3
x: 2 y: 3
x: 8 y: 27
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 2
x: 8 y: 27
x: 64 y: 729
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 4
x: 64 y: 729
x: 729 y:64
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 0
```

U liniji 18 pFunc je deklarisano kao pointer na funkciju koja vraća void i uzima dva parametra, (oba su celobrojne reference). U liniji 9 Pri ntVal s je deklarisano kao funkcija koja uzima tri parametra. Prvi je pointer na funkciju koja vraća voi d, ali uzima dva celobrojna parametra, a drugi i treći argument za Pri ntVal s su celobrojne reference. Korisnik je ponovo upitan koju funkciju da pozove, a zatim se u liniji 34 poziva Pri ntVal s.

Pronadite C + + programera i upitajte ga šta znači sledeća deklaracija:

```
void PrintVals(void (*)(int&, int&),int&, int&);
```

Ovu vrstu deklaracije ćete retko koristiti i verovatno ćete, svaki put kada Vam bude zatrebala, pogledati u knjigu, ali ona će sačuvati Vaš program u onim retkim situacijama kada je upravo to zahtevana konstrukcija.

Korišćenje typedef sa pointerima na funkcije

Konstrukcija void (*) (i nt&, i nt&) je glomazna. Možete koristiti typedef, da biste je pojednostavili, tako što ćete deklarisati tip VPF kao pointer na funkciju koja vraća void i uzima dve celobrojne reference. Listing 14.9 preinačava listing 14.8, tako Sto koristi typedef naredbu.

v **ИАРОМБКА** > Da biste izvršili kompajliranje ovog programa, smestite linije 46-80 iz listinga 14.5 odmah iza linije 45.

Listing 14.9: Korišćenje typedef da bi pokazivači no funkcije bili čitljiviji.

```
// Listing 14.9. Korišćenje typedef da bi pokazivači na funkcije bili čitljiviji

#include <iostream.h>

void Square (int&,int&);
void Cube (int&, int&);
void Swap (int&, int &);
void GetVals(int&, int&);
typedef void (*VPF) (int&, int&) ;
 void PrintVals(VPF,int&, int&);
 enum BOOL { FALSE, TRUE };

 int main()
 {
 int val0ne=l, valTwo=2;
 int choice;
 BOOL fQuit = FALSE;

 VPF pFunc;

 while (fQuit == FALSE)
 {
 cout « "(O)Quit (l)Change Values (2)Square (3)Cube (4)Swap:
 cin » choice;
 switch (choice)
 {
 case l:pFunc = GetVals; break;
 case 2:pFunc = Square; break;
 case 3:pFunc = Cube; break;
 case 4:pFunc = Swap; break;
 defaultrfQuit = TRUE; break;
 }
 if (fQuit == TRUE)
 break;
 PrintVals ( pFunc, valOne, valTwo);
 }
 return 0;
 }
```


```
void PrintVals( VPF pFunc.inti x, int& y)
{
cout « "x: " « x « " y: " « y « endl;
pFunc(x.y);
cout « "x: " « x « " y: " « y « endl;
}
```

```
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 1
x: 1 y: 2
New value for ValOne: 2
New value for ValTwo: 3
x: 2 y: 3
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 3
x: 2 y: 3
x: 8 y: 27
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 2
x: 8 y: 27
x: 64 y: 729
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 4
x: 64 y: 729
x: 729 y: 64
(0)Quit (l)Change Values (2)Square (3)Cube (4)Swap: 0
```

U liniji 9 typedef je upotrebljen da bi VPF deklarisao da bude tip "funkcije koja vraća void i uzima dva parametra - oba su integer reference."

U liniji 10 funkcija Pri ntVal s () je deklarisana tako da uzima tri parametra: VPF i dve integer reference. U liniji 19 pFunc je deklarisano da bude VPF tipa.

Pošto je VPF tip definisan, sva naredna korišćenja, koja služe za deklarisanje pFunc i PrintValsO, mnogo su čistija. Kao što možete da vidite, izlaz je identičan.

Pointeri na funkcije članove

Sve do ovog trenutka, svi pointeri funkcija koje ste kreirali bili su za opšte, neklasifikovane funkcije. Međutim, moguće je kreirati i pointere na funkcije koje su članovi klasa.

Da biste kreirali pointer na funkciju člana upotrebite istu sintaksu koju ste koristili za pointer na funkciju, ali uključite ime klase i operatora opsega (::). Stoga, ako pFunc ukazuje na funkciju člana klase Shape, koja uzima dva integera i vraća void, deklaracija pFunc će izgledati ovako:

```
void (Shape::*pFunc) (int, int);
```

Pointeri na funkcije članove su upotrebljeni potpuno isto onako kao i pointeri na funkcije, osim što ovi zahtevaju neki objekat odgovarajuće klase, koji će ih pozvati. Listing 14.10 ilustruje korišćenje pointera na funkcije članove.

Listing 14.10: Pokqzivoci no funkcije clonice.

```
1:    //Listing 14.10 PokazivaCi na funkcije članice koji koriste virtuelne metode
2:
3:    linclude <iostream.h>
4:
5:    enum BOOL {FALSE, TRUE};
6:    class Mammal
7:    {
8:    public:
9:       Mammal():itsAge(l) {  }
10:      -Mammal() { }
11:      virtual void Speak() const = 0;
12:      virtual void MoveQ const = 0;
13:   protected:
14:      int itsAge;
15:   };
16:
17:   class Dog : public Mammal
18:   {
19:   public:
20:      void SpeakQconst { cout « "Woof!\n"; }
21:      void MoveQ const { cout « "Walking to heel...\n"; }
22:   };
23:
24:
25:   class Cat : public Mammal
26:   {
27:   public:
28:      void SpeakQconst { cout « "Meow!\n"; }
29:      void MoveQ const { cout « "slinking...\n"; }
30:   };
31:
32:
33:   class Horse : public Mammal
34:   {
35:   public:
36:      void SpeakQconst { cout « "Winnie!\n"; }
37:      void MoveQ const { cout « "Galloping...\n"; }
38:   };
39:
40:
41:   int mainQ
42:   {
43:      void (Mammal::*pFunc)Q const =0;
44:      Mammal* ptr =0;
45:      int Animal;
46:      int Method;
47:      B00L fQuit = FALSE;
48:
49:      while (fQuit == FALSE)
50:      {
51:         cout « "(0)Quit (l)dog (2)cat (3)horse:
52:         cin » Animal;
53:         switch (Animal)
54:         {
55:            case 1: ptr = new Dog; break;
56:            case 2: ptr = new Cat; break;
57:            case 3: ptr = new Horse; break;
58:            default: fQuit = TRUE; break;
59:         }
60:         if (fQuit)
61:            break;
62:
63:         cout « "(1)Speak (2)Move: ";
64:         cin » Method;
65:         switch (Method)
66:         {
67:            case 1: pFunc = Mammal::Speak; break;
68:            default: pFunc = Mammal::Move; break;
69:         }
70:
71:         (ptr->*pFunc)();
72:         delete ptr;
73:      }
74:   return 0;
75:
```

```
(0)Quit (l)dog (2)cat (3)horse: 1
(l)Speak (2)Move: 1
Woof!
(0)Quit (l)dog (2)cat (3)horse: 2
(l)Speak (2)Move: 1
Meow!
(0)Quit (l)dog (2)cat (3)horse: 3
(l)Speak (2)Move: 2
Galloping
(0)Quit (l)dog (2)cat (3)horse: 0
```

^.I'MjW^fr U linijama 6-15 apstraktni tip podataka Mammal je deklarisan dvema čisto virtuelnim metodama, Speak() i Move(). Mammal sadrži potklase Dog, Cat i Horse, od kojih svaka vrši override Speak() i Move() metoda.

Drajver program u mai n ()-u zahteva od korisnika da izabere tip životinje koji će da kreira, a onda se nova potklasa Animal a kreira na slobodnom prostoru i dodeljuje se ptr-u, u linijama 55-57.

Zatim je korisnik upitan za metod koji bi pozvao i tada se metod dodeljuje pointeru pFunc. U liniji 71 kreirani objekat poziva izabrani metod, tako što koristi pointer ptr za pristup objektu i pFunc za pristup funkciji.

Konačno, u liniji 72 poziva se delete pointera ptr, da bi se u Slobodan prostor vratila memorija tog objekta. Primetićete da nema razloga da pozivate delete za pFunc, zato što je on pointer na kod, a ne na objekat u slobodnom prostoru. U stvari, ako ovo pokušate da uradite, program će javiti grešku u kompilaciji.

Nizovi pointera na funkcije članove

Kao što je slučaj sa pointerima na funkcije, pointeri na funkcije članove mogu se smestiti u niz. Taj niz se može inicijalizovati adresama različitih funkcija članova, a ove se, pak, pozivaju uz pomoć ofset-a u nizu. U listingu 14.11 ilustrovana je ova tehnika.

Listing 14.11: Niz pokazivaca na funkcije članice.

```
1:    //Listing 14.11 Niz pokazivaca na funkcije članice
2:
3:    linclude <iostream.h>
4:
5:    enum BOOL {FALSE, TRUE};
6:
7:    class Dog
8:    {
9:    public:
10:      void SpeakQconst { cout « "Woof!\n"; }
11:      void Move() const { cout « "Walking to heel...\n"; }
12:      void Eat() const { cout « "Gobbling food...\n"; }
13:      void Growl() const { cout « "Grrrrr\n"; }
14:      void Whimper() const { cout « "Whining noises...\n"; }
15:      void Roll0ver() const { cout « "Rolling over...\n"; }
16:      void PlayDeadO const { cout « "Is this the end of Little Caeser?\n"; }
17:   };
18:
19:   typedef void (Dog::*PDF)Oconst ;
20:   int main()
21:   {
22:      const int MaxFuncs = 7;
23:      PDF DogFunctions[MaxFuncs] =
24:         {  Dog::Speak,
25:            Dog::Move,
26:            Dog::Eat,
27:            Dog::Growl,
28:            Dog::Whimper,
29:            Dog::Roll0ver,
30:            Dog::PlayDead }; I
31:
32:      Dog* pDog =0;
33:      int Method;
34:      BOOL fQuit = FALSE;
35:
```



```
36:      while (IfQuit)
37:      {
38:         cout «     "(O)Quit (I)Speak (2)Move (3)Eat (4)Growr
39:         cout « " (5)Whimper (6)Roll Over (7)Play Dead: ";
40:         cin » Method;
41:         if (Method == 0)
42:         {
43:            fQuit = TRUE;
44:            break;
45:         }
46:         else
47:         {
48:            pDog = new Dog;
49:          (pDog->*DogFunctions[Method-1])();
50:            delete pDog;
51:         }
52:      }
53:    return 0;
54:   }
```

```
(0)Quit (I)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll Over (7)Play
•-•Dead: 1
Woof!
 (0)Quit (I)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll Over (7)Play
^Dead: 4
Grrr
 (0)Quit (I)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll Over (7)Play
^Dead: 7
Is this the end of Little Caeser?
 (0)Quit (I)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll Over (7)Play
^Dead: 0
```

u linijama 7-17 kreirana je klasa Dog, sa sedam funkcija članova, koje sve dele isti povratni tip i potpis. U liniji 19 typedef deklariše PDF kao pointer na funkciju člana Dog, koja niti uzima parametre, niti vraća vrednosti, i stoga je const: potpis za sedam funkcija članova je Dog.

U linijama 23-30 niz DogFunction je deklarisan da sadrži sedam takvih funkcija članova i inicijalizovan je adresama tih funkcija.

U linijama 38 i 39, od korisnika je zatraženo da izabere metod. Osim ako ne izabere Qui t, kreira se novi Dog na steku i tada se iz niza, definisanog u liniji 49, poziva korektni metod. Ovde postoji još jedna dobra linija, koju možete pokazati "zapaljenim" C + + programerima u Vašoj firmi; upitajte ih šta ova linija radi:

```
(pDog->*DogFunctions[Method-1]) ();
```

Još jednom Vam skrećemo pažnju da je ovo pomalo ezoterično, ali kada Vam je potrebno da od funkcija članova napravite tabelu, ovo može učiniti Vaš program mnogo čitljivijim i razumljivijim.

PAZITE $p **Pozivajte pointere na funkcije clanove za specifične objekte klase.**

Koristite typedef, da bi pointeri na deklaracije funkcija cianova bili citljiviji.

Nemojte koristiti pointere na funkcije clanove onda kada postoje jednostavnija resenja.

Rezime

Danas ste naučili kako da kreirate statičke promenljive članove u Vašoj klasi. Svaka klasa, pre nego svaki objekat, ima jednu kopiju statičke promenljive člana. Moguće je pristupiti ovoj promenljivoj članu bez objekta tipa klase, tako što ćete navesti puno ime, pod pretpostavkom da ste statički član deklarisali tako da ima public (javni) pristup.

Statičke promenljive članovi mogu se koristiti kao brojači unutar kopija klase. Pošto one nisu deo objekta, deklaracija statičkih promenljivih cianova ne alocira memoriju i statičke promenljive članovi moraju biti definisane i inicijalizovane izvan deklaracije klase.

Statičke funkcije članovi su deo klase na isti način kao što su to i statičke promenljive članovi. Njima se može pristupiti bez posebnog objekta klase i mogu se koristiti za pristupanje statičkim podacima clanovima. Statičke funkcije članovi ne mogu se koristiti za pristupanje nestatičkim podacima clanovima, budući da one nemaju this pointer.

Pošto statičke funkcije članovi nemaju this pointer, one, takođe, ne mogu da budu const; const u funkciji članu ukazuje da je this pointer const.

Danas ste takođe naučili kako da deklarišete i koristite pointere na funkcije i pointere na funkcije clanove. Videli ste kako se kreiraju nizovi ovih pointera i kako se prosleduju funkcijama.

Pointeri na funkcije i pointeri na funkcije clanove mogu se koristiti za kreiranje tabela funkcija, iz kojih se vrši izbor u u fazi izvršavanja. To pruža Vašem programu fleksibilnost, koju nije Iako "steći" bez ovih pointera.

Pitanja i odgovori

P Zašto koristiti statičke, kada možete da koristite globalne podatke?

O Statički podaci su ograničeni na klasu. Oni su dostupni samo preko nekog objekta klase, kroz eksplicitni poziv koji koristi ime klase, ako su publ i c, ili uz pomoć statičke funkcije člana. Međutim, statički podaci su istog tipa kao i klasa i restriktivan pristup i izražena tipizacija čine statičke mnogo sigurnijim od globalnih podataka.

P **Zašto koristiti statičke funkcije clanove, kada možete da koristite globalne funkcije? (**

O Članovi statičke funkcije **su** ograničeni samo na klasu i mogu **se** pozvati samo uz pomoć nekog objekta klase, ili eksplicitnom punom specifikacijom (kao, na primer: `ClassName: :FunctionName()`).

P **Da li je uobičajeno koristiti mnogo pointera na funkcije i na funkcije clanove?**

O Ne, oni imaju svoju specifičnu upotrebu, ali nisu uobičajene konstrukcije. Mnogi kompleksni i moćni programi ih nemaju.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje predenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Pitanja

1. Mogu li statičke promenljive članovi da budu private?
2. Prikažite deklaraciju za statičke promenljive clanove?
3. Prikažite deklaraciju za statičke pointere funkcija?
4. Prikažite deklaraciju za pointere na funkciju, koja vraća long, a uzima integer kao parametar.
5. Izmenite pointer u pitanju 4, tako da on bude pointer na funkciju člana klase Car.
6. Prikažite deklaraciju za niz od 10 pointera, kao što je definisano u pitanju 5.

Vežbe

1. Napišite kratak program, koji deklariše klasu sa jednom promenljivom članom i jednom statičkom promenljivom članom. Neka konstruktor inicijalizuje promenljivu člana i poveća statičku promenljivu člana. Uključite destruktor, koji će umanjiti promenljivu člana.
2. Koristeći program iz Vežbe 1, napišite kratak drajver program, koji će da kreira tri objekta i da, zatim, prikaže tri promenljive člana i statičku promenljivu člana. Zatim, uništite sve objekte i prikažite efekat na statičkoj promenljivoj članu.

3. Izmenite program iz Vežbe 2, tako da koristite statičku funkciju člana za pristupanje statičkoj promenljivoj članu. Neka statička promenljiva clan bude pri vate.
4. Napišite pointer na funkciju člana za pristup nestatičkom podatku članu u programu iz Vežbe **3** i upotrebite taj pointer za prikaz vrednosti tog podatka.
5. Dodajte još dve promenljive člana klasi iz prethodne vežbe. Dodajte funkciju koja će preuzeti vrednost i dodeliti svim funkcijama clanovima iste povratne vrednosti i potpise. Upotrebite pointer na funkciju člana za pristup ovim funkcijama.

Pregled sadržaja

Program koji je dat u Pregledu za drugu nedelju objedinjuje veliki broj onih znanja koja ste stekli u protekle dve sedmice i formira moćan program.

U ovoj demonstraciji povezanih lista korišćene su virtuelne funkcije, čiste virtuelne funkcije, overrriding funkcija, polimorfizam, public (javno) nasledivanje, overload funkcija, beskonačne petlje, pointeri, reference i još mnogo toga.

Osnovna svrha ovog programa je da kreira povezanu listu. Cvorovi u listi su dizajnirani tako da "drže" delove, koji se mogu koristiti u proizvodnji. Iako ovo nije i finalna forma programa, ona može da posluži kao dobra demonstracija naprednih struktura podataka. Kod se sastoji od 311 linija. Pokušajte sami da analizirate kod, pre nego što pročitate analizu koja sledi posle izlaza.

Listing R2.1: Pregled druge nedelje.

```
Q.    J!************************************
1:    //
2:    // Naslov:        Pregled druge nedelje
3:    //
4:    // Datoteka:        Week2
5:    //
6:    // Opis:    Obezbeduje demonstracioni program povezane liste
7:    //
8:    // Klase:      PART - čuva brojeve delova i potencijalno druge
9:    //                    informacije o delovima
```

nastavlja se

Listing R2.1: Pregled druge nedelje. ***nastavak***

```
//
//              PartNode - ponaša se kao Cvor u PartsList
//
//              PartsList - obezbeduje mehanizme za povezanu listu delova
//
// Autor:       Jesse Liberty (jl)
//
// Razvijeno na:   486/66 32rab RAM  HVC 1.5
//
// Cilj:        Nezavisna platform
//
// Rev istorija:   9/94 - Prvo izdanje (jl)
//
//**************************************************
//

linclude <iostream.h>

typedef unsigned long ULONG;
typedef unsigned short USHORT;


// ********** Deo ***********

// Abstraktna osnovna klasa delova
class Part
{
public:
   Part():itsPartNumber(l)  {}
   Part(ULONG PartNumber):itsPartNumber(PartNumber){}
   virtual ~Part(){};
   ULONG GetPartNumber() const { return itsPartNumber; }
   virtual void DisplayO const =0;  // mora se preklopiti
private:
   ULONG itsPartNumber;
};

// implementacija čistih virtuelnih funkcija tako da se
// izvedena klasa može povezati
void Part::Display() const

    cout « "ĐnPart Number: " « itsPartNumber « endl;
}

// ********** Automobil ********

class CarPart : public Part
{
public:
```



```
58:        CarPart():itsModelYear(94){}
59:       CarPart(USHORT year, ULONG partNumber);
60:       virtual void DisplayO const
61:      {
62:         Part::DisplayO; cout « "Model Year: ";
63:         cout « itsModelYear « endl;
64:      }
65:    private:
66:       USHORT itsModelYear;[4]
67:    };
68:
69:   CarPart::CarPart(USHORT year, ULONG partNumber):
70:       i tsModelYear(year),
71:       Part(partNumber)
72:    {}
73:
74:
75:    // ********** Avion ***********
76:
77:    class AirPlanePart : public Part
78:    {
79:    public:
80:       Ai rPlanePart():itsEngineNumber(1){};
81:       AirPlanePart(USHORT EngineNumber, ULONG PartNumber);
82:       virtual void DisplayO const
83:      {
84:         Part::Display(); cout « "Engine No.: ";
85:         cout « itsEngineNumber « endl;
86:      }
87:    private:
88:       USHORT itsEngineNumber;
89:    };
90:
91:   AirPlanePart::AirPlanePart(USHORT EngineNumber, ULONG PartNumber):
92:      itsEngineNumber(EngineNumber),
93:       Part(PartNumber)
94:    {}
95:
96:     // ********** £... za deo ***********
97:    class PartNode
98:    {
99:    public:
100:       PartNode  (Part*);
101:       -PartNode();
102:       void SetNext(PartNode * node) { itsNext = node; }
103:       PartNode * GetNext() const;
104:       Part * GetPart() const;
105:    private:
106:       Part *itsPart;
```

nastavlja se

Listing R2.1: Pregled druge nedelje. *nastavak*

```
107:        PartNode * itsNext;
108:    };
109:
110:    // PartNode implemetnacije...
III:
112:    PartNode::PartNode(Part* pPart):
113:    itsPart(pPart),
114:    itsNext(O)
115:    {}
116:
117:    PartNode: :-PartNode()
118:    {
119:        delete itsPart;
120:        itsPart = 0;
121:        delete itsNext;
122:        itsNext = 0;
123:    }
124:
125:    // Vrača NULL ako nema sledećeg PartNode
126:    PartNode * PartNode::GetNext() const
127:    {
128:          return itsNext;
129:    }
130:
131:    Part * PartNode::GetPart() const
132:    {
133:        if (itsPart)
134:            return itsPart;
135:        else
136:            return NULL; //greška
137:    }
138:
139'    // ***************** Lista delova ************
140:    class PartsList
141:    {
142:    public:
143:        PartsList();
144:        -PartsList();
145:        // potreban je konstruktor kopije i operator jednako!
146:        Part*       Find(UL0NG & position, ULONG PartNumber)  const;
147:        ULONG       GetCount() const { return itsCount; }
148:        Part*       GetFirst() const;
149:        static      PartsList& GetGlobalPartsList()
150:     {
151:         return   GlobalPartsList;
152:     }
153:        void        Insert(Part *);
154:        void        Iterate(void (Part: :*f) Oconst)  const;
    Part*       operator[](ULONG) const;
private:
    PartNode * pHead;
    ULONG itsCount;
    static PartsList GlobalPartsList;
};

PartsList PartsList::GlobalPartsList;

// Implementacije za liste...

PartsList::PartsList():
    pHead(O),
    itsCount(O)
    {}

PartsList::-PartsList()
{
    delete pHead;
}

Part*    PartsList::GetFirst() const
{
    if (pHead)
        return pHead->GetPart();
    el se
        return NULL;  // greška se hvata ovde
I

Part *  PartsList::operator[](ULONG offSet) const
{
    PartNode* pNode = pHead;

    if (lpHead)
        return NULL; // greška se hvata ovde

    if (offSet > itsCount)
        return NULL; // greška

    for (ULONG i=0;i<offSet; i++)
        pNode = pNode->GetNext();

  return    pNode->GetPart();
}

Part*   PartsList::Find(UL0NG & position, ULONG PartNumber)  const
{
    PartNode * pNode = 0;
    for (pNode = pHead, position = 0;
```

nastavlja se

Listing R2.1: Pregled druge nedelje.

nastavak

```
        pNode!=NULL;
        pNode • pNode->GetNext(), position++)
    {
       if (pNode->GetPart()->GetPartNumber() == PartNumber)
          break;
    }
    if (pNode == NULL)
       return NULL;
    else
       return pNode->GetPart();


void PartsList: :Iterate(void (Part: :*func) Oconst) const
{
    if (ipHead)
       return;
    PartNode* pNode = pHead;
    do
       (pNode->GetPart()->*func)();
    while (pNode = pNode->GetNext());


void PartsList::Insert(Part* pPart)
(
    PartNode * pNode = new PartNode(pPart);
    PartNode * pCurrent = pHead;
    PartNode * pNext = 0;

    ULONG New = pPart->GetPartNumber();
    ULONG Next = 0;
    itsCount++;

    if (IpHead)
    {
       pHead = pNode;
       return;


    // ako je ovaj manji od glave
    // on je nova glava
    if (pHead->GetPart()->GetPartNumber() > New)
    (
       pNode->SetNext(pHead);
       pHead = pNode;
       return;


    for (;;)
```



```
I
       // ako nema sledećeg, dodaj ovaj novi čvor
       if (!pCurrent->GetNext())
       {
          pCurrent->SetNext(pNode);
          return;
       }
                              f
       // ako ovaj dolazi posle ovog i pre sledećeg
       // onda ga umetni ovde, inače uzmi sledeći
       pNext = pCurrent->GetNext();
       Next = pNext->GetPart()->GetPartNumber();
       if (Next > New)
       {
          pCurrent->SetNext(pNode);
          pNode->SetNext(pNext);
          return;
       }
       pCurrent = pNext;
    }
}

int main()
{
    PartsList pi = PartsList: :GetGlobalPartsList();
    Part * pPart = 0;
    ULONG PartNumber;
    USHORT value;
    ULONG choice;

    while (1)
    {
       cout « "(O)Quit (I)Car (2)Plane: ";
       cin » choice;

       if (!choice)
          break;

       cout « "New PartNumber?: ";
       cin » PartNumber;

       if (choice == 1)
       {
          cout « "Model Year?: '*;
          cin » value;
          pPart = new CarPart(value,PartNumber);
       }
       else
       {
          cout « "Engine Number?: ";
```

nastavlja se

Listing R2.1: Pregled druge nedelje. *nastavak*

```
302:              cin » value;
303:              pPart = new AirPlanePart(value,PartNumber);
304:
305:
306:           pi.Insert(pPart);
307
308         void (Part::*pFunc)Oconst = Part::Display;
309         pi.Iterate(pFunc);
310       return 0;
311
```

```
(O)Quit (I)Car (2)Plane: 1
New PartNumber?: 2837
Model Year? 90
(O)Quit (I)Car (2)Plane: 2
New PartNumber?: 378
Engine Number?: 4938
(O)Quit (I)Car (2)Plane: 1
New PartNumber?: 4499
Model Year? 94
(O)Quit (I)Car (2)Plane: 1
New PartNumber?: 3000
Model Year? 93
(O)Quit (I)Car (2)Plane: 0

Part Number: 378
Engine No.: 4938

Part Number: 2837
Model Year: 90

Part Number: 3000
Model Year: 93

Part Number: 4499
Model Year: 94
```

IIIEEJ^ Listing Pregleda druge nedelje daje povezanu listu implementacije za `Part` objekte. Povezana lista je dinamička struktura podataka; što će red, podseća na niz, ali sa promenljivom veličinom, pošto se objekti dodaju i brišu. Ova posebna povezana lista je dizajnirana za čuvanje objekata klase `Part`, gde je `Part` apstraktan tip podataka, koji služi kao bazna klasa za bilo koji objekat sa brojem dela. U ovom primeru, `Part` je podeljena na potklase `CarPart` i `AirPlanePart`.

Klasa `Part` je deklarisana u linijama 34-44 i sastoji se od broja dela i nekoliko funkcija. Namena ove klase je da sadrži informacije o delovima kao što su: koja komponenta je iskorišćena, koliko ih je na lageru i tako dalje. `Part` je apstraktni tip podataka, pojačan čistom virtuelnom funkcijom `Display ()`.

Primetite da Display () zaista ima svoju implementaciju, u linijama 48-51. Ovo je dizajnerova namera da će izvedene klase biti primorane da kreiraju sopstvenu Display () metodu, ali da se isto tako mogu vezati za ovaj metod.

U linijama 55-67 i 77-89 obezbedene su dve jednostavne izvedene klase, CarPart i AirPlanePart. Svaka je obezbedila override Display () metode, koje se, u stvari, povezuju sa Display () metodom bazne klase.

Klasa PartNode se ponaša kao interfejs između klasa Part i PartList. Ona sadrži pointer na deo i pointer na sledeći čvor u listi. Njene jedine metode su one za preuzimanje i postavljanje sledećeg čvora u listi i za vraćanje Part-a na koji ukazuje.

Inteligencija liste je u klasi PartList, čija deklaracija se nalazi u linijama 1^0-160. PartList sadrži pointer na prvi element liste (pHead) i koristi ga za pristup svim drugim metodima, krećući se kroz listu. Kretanje kroz listu podrazumeva propitivanje svakog čvora u listi za sledeći čvor, sve dok ne dosegnete node, čiji je sledeći pointerNULL:

Ovo je samo delimična implementacija. Potpuno razvijena lista bi morala da obezbedi bolji pristup prvom i poslednjem članu, ili da obezbedi iterator objekat, koji bi omogućio klijentu da se jednostavnije kreće kroz listu.

PartList obezbeduje nekoliko interesantnih metoda, koji su prikazani u abecednom redosledu. Ovo je, obično, dobra ideja, s obzirom da olakšava pronalaženje funkcija.

Find () za parametre uzima PartNumber i ULONG. Ako je pronaden deo sa odgovarajućim PartNumber-om, ona vraća pointer na Part i popunjava ULONG sa pozicijom dela u listi. Ako PartNumber nije pronađen, ona će vratiti NULL, a pozicija je nevažna.

GetCount () vraća broj elemenata u listi. PartList čuva ovaj broj kao promenljivu člana itsCount, mada ona, naravno, može da izračuna taj broj prolazom kroz listu.

GetFi rs () vraća pointer na prvi Part u listi, ili vraća NULL, ako je lista prazna.

GetGl obal Part Li st () vraća referencu na statičku promenljivu člana Global PartsLi st. Ovo je statička kopija ove klase; svaki program sa PartsList takođe ima jednu GlobalPartsList. Naravno, on je Slobodan da kreira druge PartsList. Kompletna implementacija ove ideje bi trebalo da izvrši modifikaciju konstruktora za Part, kako bi bila sigurna daje svaki deo kreiran u GlobalPartsList.

Insert preuzima pointer na Part, kreira PartNode za njega i, zatim, dodaje Part u listu u redosledu odredenom sa PartNumber.

Iterate uzima pointer na funkciju člana Part-a, koji ne uzima parametre i vraća void, pa je const. On poziva ovu funkciju za svaki Part objekat iz liste. U programu, ona je nazvana Display (), što je virtuelna funkcija, tako da će odgovarajuća DisplayO metoda biti pozvana, u zavisnosti od tipa pozvanog Part objekta u trenutku izvršavanja.

Operator {} omogućava direktan pristup Part-u na obezbedenom ofset-u. Rudimentalna provera povezivanja je obezbedena; ako je lista NULL, ili je zahtevani ofset veći od veličine liste, NULL će biti vraćen u grešci.

Primetićete da će u stvarnim programima ovi komentari funkcija biti napisani u deklaraciji klase.

Drajver program se nalazi u linijama 274-311. Pointer na PartsList je deklarisan u liniji 266 i inicijalizovan sa Gl obal PartsLi st. Primetićete da je Gl obal PartsLi st inicijalizovana u liniji 162. Ovo je neophodno, s obzirom da je deklaracija statičke promenljive člana niza definisana; definicija je morala da bude uradena izvan deklaracije klase.

U linijama 282-307 od korisnika se u petlji zahteva da unese bilo deo kola, bilo deo aviona. U zavisnosti od njegovog izbora, kreira se odgovarajući deo. Kada je kreiran, deo se unosi u listu u liniji 306.

Implementacija za Insert () metodu PartList je 226-272. Kada je unet prvi broj dela, 2837, CarPart sa brojem dela i godištem modela 90 se kreira i prosleduje funkciji LinkedList :: Insert().

U liniji 228 kreira se novi PartNode za deo i inicijalizuje se promenljiva New sa brojem dela. Promenljiva clan itsCount Parts liste se inkrementira u liniji 234.

U liniji 236, testirajte, da li je pHead NULL i da li će proizvesti True. S obzirom da je ovo prvi čvor, tačno je da je pHead pointer nula. U liniji 238 pHead je postavljen da ukazuje na novi čvor, što će funkcija i vratiti.

Od korisnika je traženo da unese drugi deo, ovoga puta Ai rPl ane deo, čini je broj 378, a broj motora 4938. Još jednom je pozvana funkcija PartsList :: Insert() i još jednom pNode je inicijalizovan ovim čvorom. Statička promenljiva clan itsCount je povećana za dva i testiran je pHead. S obzirom da je pHead u prethodnom prolazu dobio vrednost, on vise nije NULL i test neće uspeti.

U liniji 244 broj dela 2837 koji je sadržan u pHead se poredi sa tekućim brojem dela 378. S obzirom da je novi broj dela manji od onog koji sadrži pHead, on mora da postane novi pointer i test u liniji 244 će uspeti.

U liniji 246 novi čvor je postavljen da ukazuje na čvor, na koji je ukazivao pHead. Primetićete da novi čvor neće ukazivati na pHead, nego na čvor na koji pHead takođe ukazuje! U liniji 247 pHead je postavljen da ukazuje na novi čvor.

Treći put unutar petlje korisnik je uneo 4499 za broj dela za Car, sa godištem 94. Brojač je uvećan i broj ovoga puta nije manji od broja na koji ukazuje pHead. Stoga će započeti For petlja, koja se nalazi na liniji 251

Vrednost na koju ukazuje pHead je 378. Vrednost na koju ukazuje drugi čvor je 2837. Tekuća vrednost je 4499. Pointer current ukazuje na isti čvor kao i pHead i, stoga, ima sledeću vrednost.; pCurrent ukazuje na drugi čvor i provera u liniji 254 neće uspeti.

Pointer pCurrent je postavlje da ukazuje na sledeći čvor i petlja se ponavlja. Ovoga puta provera liniji 254 će biti uspešna. Ne postoji sledeća stavka, tako da će tekućem čvoru biti rečeno da ukazuje na novi čvor u liniji 256 i ovim će dodavanje biti završeno.

U četvrtom krugu unet je broj 4000. Ovo će biti obradeno kao i prethodna iteracija, ali ovoga puta kada tekući čvor bude ukazivao 2837 i sledeći čvor na 4499, test u liniji 264 će vratiti True i novi node će biti pridodat.

Kada korisnik na kraju pritisne nulu (0), test u liniji 287 će proizveti True i Wh i l e (1). Petlja će se prikinuti. U liniji 308 funkcija-član Display () je dodeljena pointeru na funkciju člana pFunc. U stvarnim programima ovo bi bilo dodeljeno dinamički, u zavisnosti od korisnikovog izbora metode.

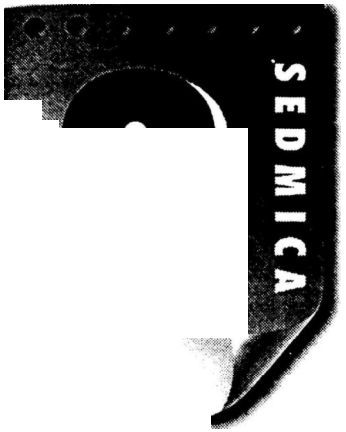
Pointer na funkciju člana je prosleden iterate () metodu PartsLi st. U liniji 216 i terete () metod proverava da li je lista prazna. U linijama 221-223 svaki Part liste se poziva korišćenjem pointera za funkciju člana. Ovim se poziva odgovarajući Display () metod za Part, kao što je prikazano u izlazu.

Pregled sadržaja

Završili ste drugu nedelju učenja C++. Do sada bi trebalo da ste stekli osećaj za napredne aspekte objektno-orijentisanog programiranja, uključujući enkapsulaciju i polimorfizme.

Kuda sada idemo

Poslednja nedelja započinje diskusijom o naprednom nasleđivanju. U Danu 16 "Strimovi" detaljno ćete se upoznati sa strimovima. U Danu 17 "Predprocesor" upoznaćete napredne "trikove" rada sa predprocesorom. U Danu 18 "Objektno-orijentisana analiza i dizajn" će biti detaljno opisani: ???umesto da se fokusirate na sintaksu jezika, iskoristićete ovaj dan da naučite nešto vise o objektnoj analizi i dizajnu. U Danu 19 "Templejti", upoznaćete se sa templejtima, a u Danu 20 "Izuzeci i obrada grešaka", biće objašnjeni izuzeci. Dan 21 "šta dalje" je poslednji dan ove knjige, kojim će biti pokriveni opšti subjekti koji nisu objašnjeni nigde drugde, a takode ćete se sresti sa diskusijom o sledećim koracima koje je potrebno da preuzmete kako biste postali C++ "guru".



Dan 15

Napredno nasledivanje

Do sada ste radili sa jednostrukim i višestrukim nasleđivanjem, kako biste kreirali je relaciju. Danas ćete naučiti:

- šta je "kontejner" i kako se modelira
- šta je delegacija i kako se modelira
- kako implementirati jednu klasu u uslovima druge
- kako se koristi privatno nasledivanje.

Kontejner

Kao što ste videli u prethodnim primerima, moguće je da podaci članovi jedne klase uključe objekte druge klase. C + + programed kažu da spoljna klasa *sadrzi* (kontejnira) unutrašnju. Stoga, klasa Employee može da sadrži string objekte (za ime zaposlenog), kao i integer-e (na primer, za platu zaposlenog) i tako dalje.

Listing 15.1 opisuje nekompletnu, ali ipak upotrebljivu string klasu. Ovaj listing ne proizvodi nikakav izlaz. Umesto toga, on će se koristiti u listinzima koji slede.

Listing 15.1: Klasa String.

```
linclude <iostream.h>
linclude <string.h>

class String
{
```

nastavlja se

Listing 15.1: Klasa String. *nastavak*

```
6:      public:
7:         // konstruktori
8:         StringO;
9:          String(const char *const);
10:         String(const String &);
11:        -StringO;
12:
13:        // preklopljeni operatori
14:        char & operator[](int offset);
15:        char operator[](int offset) const;
16:        String operator+(const String&);
17:        void operator+=(const String&);
18:        String & operator= (const String &);
19:
20:        ,// Opšte metode pristupa
21:        int Getlen()const { return itsLen; }
22:        const char * GetStringO const { return itsString; }
23:        // static int ConstructorCount;
24:
25:     private:
26:        String (int);          // orivatni konstruktor
27:        :Пar * itsString;
28:        unsigned short itsLen;
29:
30:  };
31:
32:  // podrazumevani konstruktor kreira string od 0 oajtova
33:  String::String()
34:  {
35:     itsString = new char[l];
36:     itsString[0] = '\0';
37:     itsLen=0;
38:     // cout « "\tDefault string constructor\n";
39:     // ConstructorCount++;
40:  }
41:
42:  // privatni (pomoćni) konstruktor, koga koriste samo
43:  // metode klase za kreiranje novog stringa
44:  // tražene veličine. Puni se sa null.
45:  String::String(int len)
46:  {
47:     itsString = new char[len+l];
48:     for (int i = 0; i<=len; i++)
49:        itsString[i] = '\0';
50:     itsLen=len;
51:     // cout « "\tString(int) constructor\n";
52:     // ConstructorCount++;
53:  }
```



```
// Konvertuje niz karaktera u String
String::String(const char * const cString)
{
   itsLen = strlen(cString);
   itsString = new char[itsLen+l];
   for (int i - 0; i<itsLen; i++)
      itsString[i] = cString[i];
   itsString[itsLen] = '\0';
   // cout « "\tString(char*) constructors";
   // ConStrUCtOrCount+-»;
}

// konstruktor kopije
String::String (const String & rhs)
{
   itsLen=rhs.GetLen();
   itsString = new char[itsLen+1];
   for (int i = 0; i<itsLen;i++)
      itsString[i] = rhs[i];
   itsString[itsLen] = '\0';
   // cout « "\tString(String&) constructor\n"
   // ConstructorCount++;


// destruktor, oslobada alociranu memoriju
Stri ng: :-String ()
{
   delete [] itsString;
   itsLen = 0;
   // cout « "UString destructor\n";


// operator jednako, oslobađa postojeću memoriju
// onda kopira string i veličinu
String& String::operator=(const String & rhs)
(
   if (this == &rhs)
      return *this;
   delete [] itsString;
   itsLen=rhs.GetLen();
   itsString = new char[itsLen+1];
   for (int i = 0; i<itsLen;i++)
      itsString[i] = rhs[i];
   itsString[itsLen] = '\0';
   return *this;
   // cout « "\tString operator=\n";
```

nastavlja se

Listing 15.1: Klasa String.

nastavak

```
103  //nekonstantni operator pomaka, vraća
104  // referencu na karakter tako da se on može
105  // promeniti!
106  char & String::operator[](int offset)
107  I
108     if (offset > itsLen)
109        return itsString[itsLen-l];
110     else
111        return itsString[offset];
112
113
114  // konstantni operator pomaka koji se koristi
115  // sa konstantnim objektima (pogledati konstruktor kopije!)
116  char String::operator[](int offset) const
117
118     if (offset > itsLen)
119        return itsString[itsLen-l];
120     else
121        return itsString[offset];
122
123
124  // kreira novi string dodajući tekuči
125  // string rhs stringu
126  String String::operator+(const String& rhs)
127  {
128     int totalLen = itsLen + rhs.GetLen();
129     String temp(totalLen);
130     int i, j;
131     for (i = 0; i<itsLen; i++)
132        temp[i] = itsString[i];
133     for (j = 0; j<rhs.GetLen(); j++, i++)
134        temp[i] = rhs[j];
135     temp[totalLen]='\0';
136     return temp;
137  }
138
139  // menja tekući string, ne vraća ništa
140  void String::operator+=(const String& rhs)
141  {
142     unsigned short rhsLen = rhs.GetLen();
143     unsigned short totalLen = itsLen + rhsLen;
144     String temp(totalLen);
145     for (int i = 0; i<itsLen; i++)
146        temp[i] = itsString[i];
147     for (int j = 0; j<rhs.GetLen();
148        temp[i] = rhs[i-itsLen];
149     temp[totalLen] = '\0';
150     *this - temp;
151  }
152
153  // int String::ConstructorCount

        Nema.
```


Listing 15.1 obezbeduje string klasu, sličnu onoj koja je korišćena u listingu 11.14 u Danu 11, "Nizovi." Bitna razlika je u tome da konstruktori, kao i nekoliko drugih funkcija u listingu 11.14, prikazuju naredbe, kako bi objasnili njihovo korišćenje, dok je u listingu 15.1 to iskomentarisano. Ove funkcije će se koristiti u primerima koji slede.

U liniji 23 definisana je statička promenljiva član ConstructorCount i u liniji 153 je izvršena njena inicijalizacija. Ova promenljiva se povećava u svakom string konstruktoru. Sve ovo je iskomentarisano i koristiće se u listinzima koji slede.

Listing 15.2 opisuje klasu Employee, koja sadrži tri sting objekta. Primetićete da je odredeni broj naredbi prokomentarisan; one će se koristiti u listinzima koji slede.

Listing 15.2: Klasa Employee i demonstracioni program.

```
1    class Employee
2    {
3
4    public:
5       Employee();
6       Employee(char *, char *, char *, long);
7       -Employee();
8       Employee(const Employees);
9       Employee & operator= (const Employee &);
10
11      const String & GetFirstNameO const
12         ( return itsFirstName; }
13      const String & GetLastName() const { return itsLastName; }
14      const String & GetAddressQ const { return itsAddress; }
15      long GetSalary() const { return itsSalary; }
16
17      void SetFirstName(const String & fName)
18          { itsFirstName = fName; }
19      void SetLastName(const String & lName)
20         { itsLastName = lName; }
21      void SetAddress(const String & address)
22           { itsAddress = address; }
23      void SetSalary(long salary) { itsSalary = salary; }
24   private:
25      String     itsFirstName;
26      String     itsLastName;
27      String     itsAddress;
28      long       itsSalary;
29
```

nastavlja se

Usting 15.2: Klasa Employee i demonstrocioni program. *nastavak*

```
30:
31:    Employee::Emp1oyee():
32:        itsFirstName(""),
33:        itsLastNameC"),
34:        itsAddress(""),
35:        itsSalary(O)
36:    {}
37:
38:    Employee::Employee(char * firstName, char * lastName,
39:        char * address, long salary):
40:        itsFirstName(firstName),
41:        itsLastName(lastName),
42:        itsAddress(address),
43:        itsSalary(salary)
44:    {}
45:
46:    Employee: .-Employee(const Employee & rhs):
47:        itsFirstName(rhs.GetFirstNameO),
48:        itsLastName(rhs.GetLastNameO),
49:        itsAddress(rhs.GetAddress()),
50:        itsSalary(rhs.GetSalary())
51:    {}
52:
53:    Employee::-Employee() {}
54:
55:    Employee & Employee::operator= (const Employee & rhs)
56:    {
57:        if (this == &rhs)
58:            return *this;
59:
60:        itsFirstName = rhs.GetFirstName();
61:        itsLastName = rhs.GetLastNameQ;
62:        itsAddress = rhs.GetAddressQ;
63:        itsSalary = rhs.GetSalary();
64:
65:        return *this;
66:    }
67:
68:    int main()
69:    {
70:        Employee Edie("Jane","Doe","1461 Shore Parkway", 20000);
71:        Edie.SetSalary(50000);
72:        String LastName("Levine");
73:        Edie.SetLastName(LastName);
74:        Edie.SetFirstName("Edythe");
75:
76:        cout « "Name: ";
77:        cout « Edie.GetFirstName() .GetStringO ;
78:        cout « " • « Edie.GetLastNameQ.GetStringO;
79:        cout « "AnAddress: ";
80:        cout « Edie.GetAddressO .GetStringO;
81:        cout « "AnSalary: " ;
82:        cout « Edie.GetSalaryO;
83:    return 0;
84:    }
```

^IIIO IIIIIy Smestite kod iz listinga 15.1 u datotekupod imenom STRING.HPP. Zatim, svaki put kada Vam zatreba klasa sting, možete ukljudti listing 15.1, korišćenjem naredbe `linclude`. Na primer, na vrhu listinga 15.2 dodajte liniju `#i nel ude Stri ng. hpp`. Ovim ćete dodati klasu string u Vaš program.

```
ДМГУ^.  Name:  Edythe  Levine.
~       Address:  1461  Shore  Parkway.
        Salary:  50000
```

Listing 15.2 prikazuje klasu Employee, koja sadrži tri string objekta: itsFirstName, itsLastName i itsAddress.

U liniji 70 kreira se objekat Employee i prosleduju mu se četiri vrednosti, koje ga inicijalizuju. U liniji 71 Employee pristupa funkciji SetSalaryQ sa konstantnom vrednošću 50.000. Primetićete da bi u stvarnim programima ovo bila dinamička vrednost (postavljena u vreme izvršavanja), ili konstanta.

U liniji 72 kreiran je i inicijalizovan string, korišćenjem C++ string konstante. Ovaj string objekat je, zatim, iskorišćen kao argument za SetLastNameQ u liniji 73.

U liniji 74 Empl oyee funkcija SetFi rstName je pozvana sa još jednom string konstantom. Medutim, ako obratite pažnju, primetićete da Employee nema funkciju SetFi rstName koja uzima karakter string kao argument; SetFi rstName zahteva konstantu.

Kompajler će rešiti ovaj problem, s obzirom da on zna kako da napravi string iz konstante. On ovo zna, s obzirom da ste mu to rekli u liniji 9 listinga 15.1.

Pristup clanovima sadržane klase

Objekti Employee nemaju pristup promenljivim clanovima stringa. Ako Employee objekat Edie pokuša da pristupi promenljivoj članu itsLen sopstvene promenljive člana itsFirstName, dobiće grešku u kompilaciji. Medutim, ovo i nije "strašno". Funkcija za pristup će obezbediti interfejs za klasu string i klasa Employee neće morati da brine o detaljima implementacije, ni kako će celobrojna promenljiva itsSalary smestiti informacije.

Filtriranje pristupa sadržanih cianova

Primetićete da klasa string obezbeduje operator +. Dizajner klase Employee je zabranio pristup operatoru+, koji bi bio pozvan iz Employee objekta, tako što biste

deklarisali string aksesore, kao što je GetFi rstName(), koji bi vratili konstantu. Pošto operator + nije (i ne može biti) const funkcija (on menja objekat za koji je pozvan), pokušaj izvršenja sledećeg bi izazvao grešku u kompilaciji.

```
String buffer = Edie.GetFirstNameO + Edie.GetLastNameQ;
```

GetFirstName() vraća konstantu String i Vi ne možete pozvati operator+ sa konstantnim objektom.

Da biste ovo rešili, izvršite overload funkcije GetFistName, kako ne bi bila vise konstantna:

```
const String & GetFirstName() const { return itsFirstName; }
String & GetFirstName()  { return itsFirstName; }
```

Primetićete da vraćena vrednost vise nije const, kao i da sama funkcija-član takođe vise nije const. Promena vraćene vrednosti nije dovoljna da biste izvršili overload imena funkcije; potrebno je da izvršite izmenu konstantnosti same funkcije.

Cena kontejnera

Važno je napomenuti da korisnici klase Employee plaćaju cenu za svaki od string objekata, svaki put kada se oni konstruišu, ili kada se napravi kopija klase Employee. Skidanjem komentara sa cout naredbi u listingu 15.1, a u linijama 38, 51, 63, 75, 84 i 100, saznaćete koliko često se izvršava poziv. U listingu 15.3 ispravljen je drajver program, tako što je dodata Print naredba, koja indicira kada je u programu kreiran objekat:

^NAPOMENA^ Da biste kompajlirali ovaj listing, sledite sledeće brake:

1. Skinite komentare se linija 38, 51, 63,75, 84 i 100 u listingu 15.1.
2. Izmenite listing 15.2. Uklonite linije 64-80 i zamenite listing 15.3.
3. Dodajte #include string.hpp, kao što je već rečeno.

Listing 15.3: Konstruktori sadržanih klasa.

```
1:    int main()
2:    {
3:       cout « "Creating Edie...\n";
4:       Employee Edie("Jane","Doe","1461 Shore Parkway", 20000);
5:       Edie.SetSalary(20000);
6:       cout « "Calling SetFirstName with char *...\n";
7:       Edie.SetFirstName("Edythe");
8:       cout « "Creating temporary string LastName...\n";
9:       String LastName("Levine");
10:      Edie.SetLastName(tastName);
11:
12:      cout « "Name: ";
13:      cout « Edie.GetFirstNameO .GetStringO ;
14:      cout « " " « Edie.GetLastNameO.GetStringO;
15:      cout « "\nAddress: ";
16:      cout « Edie.GetAddressO.GetStringO;
17:      cout « "\nSalary: " ;
18:      cout « Edie.GetSalary();
19:      cout « endl;
20:   return 0;
21:   }
```

```
Creating Edie...
        String(char*) constructor
        String(char*) constructor
        String(char*) constructor
Calling SetFirstName with char *..
        String(char*) constructor
        String destructor
Creating temporary string LstName.
        String(char*) constructor
Name: Edythe Levine
Address: 1461 Shore Parkway
Salary: 20000
        String destructor
        String destructor
        String destructor
        String destructor
```

Listing 15.3 koristi iste deklaracije klase kao i listinzi 15.1 i 15.2. Medutim, skinut je komentar sa cout naredbi. Izlaz iz listinga 15.3 je numerisan, da bi se lakše izvršila analiza.

U liniji 3 listinga 15.3 prikazana je naredba Creating Edie... koja se odražava u liniji 1 izlaza. U liniji 4 kreiran je Employee objekat Edie sa četiri parametra. Izlaz odražava konstruktora za String koji je pozvan tri puta, kao što se i očekivalo.

Linija 6 prikazuje informaciju i zatim je u liniji 7 naredba Edie.SetFi rstName ("Edythe"). Ova naredba prouzrokuje da se kreira privremeni string iz stringa "Edythe", kao što je prikazano u linijama 6 i 7 izlaza. Primetićete da je privremeni string nestao odmah pošto je iskorišćen u naredbi dodeljivanja.

U liniji 9 String objekat je kreiran u telu programa. Ovde programer radi tačno ono što kompajler implicitno radi sa prethodnom naredbom. Ovaj put Vi ćete videti konstruktor u liniji 9 izlaza, ali ne i destruktor. Ovaj objekat neće biti uništen kada izade iz opsega na kraju funkcije.

U linijama 13-19 stringovi u objektu zaposlenih su uništeni kada je objekat Employee izašao iz opsega i string LastName, koji je kreiran u liniji 9, takođe je uništen kada je izašao iz opsega.

Kopiranje po vrednosti

Listing 15.3 ilustruje kako kreiranje jednog Employee objekta prouzrokuje poziv pet konstruktora za stringove. U listingu 15.4 ponovo je ispravljen drajver program. Ovoga puta nisu korišćene print naredbe, ali je skinut komentar sa statičke promenljive člana ConstructorCount i ona je upotrebljena.

Ispitivanjem listinga 15.1 ustanovićete da je promenljiva ConstructorCount uvećana svaki put kada je pozvan string konstruktor. Drajver program iz listinga 15.4 poziva funkciju Print, prosledujući joj Employee objekat, prvo po referenci, a zatim po vrednosti. ConstructorCount prati broj string objekata koji su kreirani kada je radnik (empleyee) prosleden kao parametar.

^WPOMENAv, Da biste kompajlirali ovaj listing:

1. skinite komentar sa linija 23,39,52,64,76 i 152 iz listinga 15.1.
2. izmenite listing 15.2 i uklonite linije 68-84 i zamenite ih listingom 15.4
3. dodajte #include string.hpp, kako je prerhodno naznačeno.

Listing 15.4: Predavanje po vrednosti.

```
void  PrintFunc(Employee);
void rPrintFunc(const  Employees);

int  main()
{
   Employee Edie("Jane","Doe","1461 Shore Parkway", 20000);
   Edie.SetSalary(20000);
   Edie.SetFir£tName("Edythe");
   String  LastName("Levine");
   Edie.SetLastName(LastName);

   cout « "Constructor count: " ;
   cout « String:-.ConstructorCount « endl;
   rPrintFunc(Edie);
   cout « "Constructor count: ";
    cout « String::ConstructorCount « endl;
   PrintFunc(Edie);
   cout « "Constructor count: ";
    cout « String::ConstructorCount « endl;
 return 0;
}
void PrintFunc (Employee  Edie)


   cout «  "Name: ";
   cout «  Edie.GetFirstNameO .GetStringO ;
   cout «  " " « Edie.GetLastNameO .GetStringO;
   cout «  "AnAddress: ";
   cout «  Edie.GetAddress().GetStringO;
   cout « "AnSalary:  " ;
   cout «  Edie.GetSalaryO;
   cout «  endl;


void rPrintFunc (const Employees Edie)
{
   cout « "Name:  ";
   cout «  Edie.GetFirstNameO.GetStringO;
   cout « " · « Edie.GetLastNameO .GetStringO;
   cout « "\nAddress:  ";
   cout «  Edie.GetAddressO .GetStringO ;
   cout « "\nSalary:  " ;
   cout «  Edie.GetSalaryO;
   cout «  endl;
}
```

```
String(char*)  constructor
        String(char*)  constructor
        String(char*)  constructor
        String(char*)  constructor
        String  destructor
        String(char*)  constructor
Constructor count: 5
Name:  Edythe Levine
Address: 1461 Shore Parkway
Salary: 20000
Constructor count: 5
        String(StringS)  constructor
        String(StringS)  constructor
        String(StringS)  constructor
Name:  Edythe Levine.
Address: 1461 Shore Parkway.
Salary: 20000
        String  destructor
        String  destructor
        String  destructor
Constructor count: 8
        String  destructor
        String  destructor
        String  destructor
        String  destructor
```

Izlaz prikazuje da je kreirano pet string objekata, koji su deo kreiranog Employee objekta. Kada je Employee objekat prosleden funkciji rPrintFunc() po referenci, nije kreiran ni jedan dodatni Employee objekat, pa, stoga, nisu kreirani ni dodatni String objekti (i oni su, takođe, prosledeni po referenci).

Cppp Kada je u liniji 14 Employee objekat prosleden PrintFuncO po vrednosti, kreirana je kopija Employee, kao što su kreirani i dodatni string objekti (tri string objekta su kreirana uz pomoć CopyConstructora).

Implementacija izraza Nasleđivanje/Kontejner protiv Delegacije

U nekim situacijama jedna klasa želi da povuče neke atribute druge klase. Na primer, pretpostavimo da želite da kreirate klasu PartsCatalog. Specifikacija koju ste dali definiše PartsCatalog kao kolekciju delova, u kojoj svaki deo ima svoj jedinstveni broj. PartsCatalog ne dozvoljava duple ulaze i dozvoljava pristup po broju dela.

Listing u Pregledu za Nedelju 2 obezbeduje klasu LinkedList. Ova LinkedList klasa je razumljiva i dobro testirana i na osnovu nje ćete želeti da kreirate tehnologiju za Vašu PartsCatalog, radije nego da izmišljate nešto novo.

Možete da kreirate novu klasu PartsCatalog, tako da ona sadrži LinkedList. PartsCatalog može da delegira upravljanje povezanom listom njenom LinkedList objektu koji sadrži.

Alternativa može da bude da se klasa PartsCatalog izvede iz klase LinkedList, koja će, stoga, naslediti i sve osobine klase LinkedList. Međutim, primetićete da javno nasledivanje obezbeduje jednu je relaciju, pa se, stoga, morate upitati da li je PartsCatalog stvarno tip LinkedList klase.

Jedan od načina da odgovorite na pitanje da li je PartsCatalog deo LinkedList je da prihvatite da je LinkedList bazna i da je PartsCatalog izvedena klasa i da, zatim, postavite sledeća pitanja:

1. Da li postoji nešto u baznoj klasi što ne može da se nalazi u izvedenoj? Na primer, da li LinkedList bazna klasa ima funkcije koje ne odgovaraju klasi PartsCatalog. Ako je tako, verovatno ćete želeti javno nasledivanje.
2. Može li klasa koju kreirate imati vise od jedne bazne klase? Na primer, može li klasa PartsCatalog imati potrebu za dvema LinkedList u svakom objektu? Ako može, verovatno će biti potrebno da koristite kontejner.
3. Da li imate potrebu da nasledujete iz bazne klase, kako biste dobili prednosti virtuelnih funkcija, ili pristupili zaštićenim clanovima? Ako je tako, obavezno koristite nasledivanje, bilo public, bilo private.

Na osnovu odgovora na ova pitanja, morate se odlučiti između javnog (je relacije) i privatnog nasledivanja, ili kontejnera.

- Kontejner - objekat deklarisan kao clan druge klase koja sadrži tu klasu.
- Delegacija - korišćenje atributa sadržane klase koji služe za izvršavanje funkcija. U drugim slučajevima nisu na raspolaganju klasi koja ih sadrži.
- Implementacija - izgradnja jedne klase sa mogućnostima druge klase, bez korišćenja javnog nasledivanja.

Delegacija

Zašto ne izvesti klasu PartsCatalog iz LinkedList? PartsCatalog nije LinkedList, s obzirom da su LinkedLists uređene kolekcije i svaki clan kolekcije se može ponavljati. PartCatalog ima jedinstvene ulaze i nije ureden. Peti clan PartsCataloga nije deo No. 5.

Naravno, moguće je naslediti javnost iz PartsLista i, zatim, izvršiti override Insert() i ofset operatora. Da biste ovo izvršili, biće potrebno da izvršite izmenu PartsList klase. Umesto toga, izgradićete PartsCatalog, koji nema ofset katalog, ne dozvoljava duplikate i definiše operator + za kombinovanje dva seta.

Prvi način da obavite ovaj posao je uz pomoć kontejnera. PartsCatalog će delegirati upravljanje listom, sadržanoj u LinkedList. Listing 15.5. ilustruje ovaj pristup.

Listing 15.5: Upravljanje sadriano u LinkedList.

```
linclude <iostream.h>

typedef unsigned long ULONG;
typedef unsigned short USHORT;


// ********** Deo ***********

// Abstraktna osnovna klasa delova
class Part
 {
 public:
    Part():itsPartNumber(l)  {}
    Part(ULONG PartNumber):
        i tsPartNumber(PartNumber){}
    virtual  -Part(H)
    ULONG GetPartNumberO const
        { return itsPartNumber; }
    virtual void DisplayO const =0;
private:
    ULONG itsPartNumber;
```

nastavlja se

Listing 15.5: Upravljanjesadržano u LinkedList. *nastavak*

```
// implementacija čiste virtuelne funkcije tako dar>e
// izvedena klasa može nadovezati
void Part::Display() const
{
    cout « "\nPart Number: " « itsPartNumber « endl;


jl ********** Automobil **********

class CarPart : public Part
(
public:
   CarPart():itsModelYear(94){}
   CarPart(USHORT year, ULONG partNumber);
   virtual void DisplayO const
   i
      Part::Display();
      cout « "Model Year: ";
      cout « itsModelYear « endl;
   }
private:
   USHORT itsModelYear;
);

CarPart::CarPart(USHORT year, ULONG partNumber):
   itsModelYear(year),
   Part(partNumber)
0


// ********** Avion **********

class AirPlanePart : public Part
{
public:
   AirPlanePartO :itsEngineNumber(I) {};
   AirPlanePart
      (USHORT EngineNumber, ULONG PartNumber)
   virtual void DisplayO const
   {
      Part::DisplayO;
      cout « "Engine No.: ";
      cout « itsEngineNumber « endl;
   }
private:
   USHORT itsEngineNumber;
```



```
AirPlanePart::AirPlanePart
     (USHORT EngineNumber, ULONG PartNumber):
   itsEngineNumber(EngineNumber),
   Part(PartNumber)
0

// ********** £yor za **********
class PartNode
{
public:
   PartNode (Part*);
   -PartNodeO;
   void SetNext(PartNode * node)
      { itsNext = node; }
   PartNode * GetNextO const;
   Part * GetPart() const;
private:
   Part *itsPart;
   PartNode * itsNext;
};
// PartNode implementacije...

PartNode::PartNode(Part* pPart):
itsPart(pPart),
itsNext(0)
0

PartNode::-PartNode()
{
    delete itsPart;
    itsPart = 0;
    delete itsNext;
    itsNext = 0;
}

// Vraća NULL ako nema sledećeg PartNode
PartNode * PartNode::GetNext() const
{
     return itsNext;
}

Part * PartNode::GetPart() const
I
   if (itsPart)
      return itsPart;
   else
      return NULL; //greška
```

nastavljase

Listing 15.5: Upravljanje sadržano u LinkedList. *nastavak*

```
118
119
120
121
122     jl ***************** Lista delova ************
123     class PartsList
124     {
125     public:
126        PartsList ();
127        -PartsList();
128        // potreban konstruktor kopije i operator jednako!
129        void        Iterate(void (Part::*f)()const) const;
130        Part*     Find(ULONG & position, ULONG PartNumber)  const;
131        Part*      GetFirst() const;
132        void       Insert(Part *);
133        Part*      operator[](ULONG) const;
134        ULONG      GetCountO const { return itsCount; }
135        static      PartsList& GetGlobalPartsList()
136        {
137            return    GlobalPartsList;
138
139     private:
140        PartNode * pHead;
141        ULONG itsCount;
142        static PartsList GlobalPartsList;
143:
144
145     PartsList PartsList::GlobalPartsList;
146
147
148      PartsList: -.PartsList():
149         pHead(O),
150         itsCount(0)
151         0
152
153      PartsList::-PartsList()
154      {
155         delete pHead;
156
157
158      Part*     PartsList::GetFirst() const
159      {
160         if (pHead)
161            return pHead->GetPart();
162         el se
163            return NULL;   // ovde se hvata greška
164
165
```



```
Part *   PartsList: operator[] (ULONG offSet) const
{
   PartNode* pNode - pHead;

   if (IpHead)
      return NULL; // ovde se hvata greška

   if (offSet > itsCount)
      return NULL; // greška

   for (ULONG i=0;i<offSet; i++)
      pNode = pNode->GetNext();

  return    pNode->GetPart();
}

Part*    PartsList::Find(
    ULONG & position,
    ULONG PartNumber)  const
{
   PartNode * pNode = 0;
   for (pNode = pHead, position = 0;
        pNode!=NULL;
        pNode = pNode->GetNext(), position++)
   {
      if (pNode->GetPart()->GetPartNumber()  ==  PartNumber)
         break;
   }
   if (pNode  ==  NULL)
      return NULL;
   el se
      return  pNode->GetPart();
}

void  PartsLi st:: Iterate(void  (Part: :*func) Oconst)  const
{
   if  (IpHead)
      return;
   PartNode* pNode = pHead;
   do
      (pNode->GetPart()->*func)();
   while (pNode = pNode->GetNext());


void  PartsList::Insert(Part*  pPart)
{
   PartNode * pNode = new PartNode(pPart);
   PartNode * pCurrent = pHead;
   PartNode * pNext = 0;
```

nastavlja se

Listing 15.5: Upravljanje sadrzano u LinkedList.

nastavak

```
215
216         ULONG New = pPart->GetPartNumber();
217         ULONG Next = 0;
218         itsCount++;
219
220         if (IpHead)
221         {
222            pHead = pNode;
223            return;
224         }
225
226         // ako je ovaj manji od glave
227         // ovaj je nova glava
228         if (pHead->GetPart()->GetPartNumber() > New)
229         {
230            pNode->SetNext(pHead);
231            pHead = pNode;
232            return;
233
234
235         for (;;)
236         {
237            // ako nema sledećeg, dodaj novi
238            if (!pCurrent->GetNext())
239            {
240               pCurrent->SetNext(pNode);
241               return;
242            }
243
244            // ako ovaj ide posle ovog i pre sledećeg
245            // onda ga umetni ovde, inače uzmi sledećeg
246            pNext = pCurrent->GetNext();
247            Next = pNext->GetPart()->GetPartNumber();
248            if (Next > New)
249            {
250               pCurrent->SetNext(pNode);
251               pNode->SetNext(pNext);
252               return;
253            }
254            pCurrent = pNext;
255
256
257
258
259
260      class PartsCatalog
261      {
262      public:
```



```
   void Insert(Part *);
   ULONG Exists(ULONG PartNumber);
   Part * Get(int PartNumber);
   operator+(const PartsCatalog &);
   void ShowAllO { thePartsList.Iterate(Part::0isplay); }
private:
   PartsList thePartsList;


void PartsCatalog::Insert(Part * newPart)
{
   ULONG partNumber = newPart->GetPartNumber();
   ULONG offset;

   if (.'thePartsList. Find (offset, partNumber))

      thePartsLi st.Insert(newPart);
   else
   {
      cout « partNumber « " was the ";
      switch (offset)
      {
         case 0:  cout « "first "; break;
         case 1:  cout « "second "; break;
         case 2:  cout « "third "; break;
         default: cout « offset+1 « "th ";
      }
      cout « "entry. Rejected!\n";
   I
}

ULONG PartsCatalog:.-Exists(ULONG PartNumber)
{
   ULONG offset;
   thePartsList.Find(offset,PartNumber);
      return offset;


Part * PartsCatalog::Get(int PartNumber)
{
   ULONG offset;
   Part * thePart = thePartsList.Find(offset, PartNumber);
   return thePart;



int main()
{
   PartsCatalog pc;
```

nastavljase

Listing 15.5: Upravljanje sadrzano u LinkedList.

```
312:        Part * pPart = 0;
313:        ULONG PartNumber;
314:        USHORT value;
315:        ULONG choice;
316:
317:        while (1)
318:        {
319:           cout « "(O)Quit (I)Car (2)Plane: ";
320:           cin » choice;
321:
322:           if (!choice)
323:              break;
324:
325:           cout « "New PartNumber?: ";
326:           cin »  PartNumber;
327:
328:           if (choice == 1)
329:           {
330:              cout « "Model Year?: ";
331:              cin » value;
332:              pPart = new CarPart(value,PartNumber);
333:           )
334:           el se
335:           {
336:              cout « "Engine Number?: ";
337:              cin » value;
338:              pPart = new AirPlanePart(value,PartNumber);
339:           }
340:           pc.Insert(pPart);
341:        }
342:        pc.ShowAllQ;
343         return 0;
344      }
```

```
fc> (0)Quit (I)Car (2)Plane:  1
    New PartNumber?: 1234
    Model Year?: 94
    (0)Quit (I)Car (2)Plane:  1
    New PartNumber?: 4434
    Model Year?: 93
    (0)Quit (I)Car (2)Plane:  1
    New PartNumber?: 1234
    Model Year?: 94
    1234 was the first entry.  Rejected!
    (0)Quit (I)Car (2)Plane:  1
    New PartNumber?: 2345
    Model Year?: 93
    (0)Quit (I)Car (2)Plane:  0
```


nastavak

```
Model Year: 94

Part Number: 2345
Model Year: 93

Part Number: 4434
Model Year: 93
```

Listing 15.7 prikazuje interfejs za Part, PartNode i PartList klase iz Pregleda Nedelje 2, ali da bi one sačuvale prostor, on ne prikazuje implementaciju svojih metoda.

Nova klasa PartCatal og je deklarisana u linijama 260-270. PartsCatal og ima artsLi st kao svoj podatak clan, kome delegira upravljanje listom. Drugi način da ovo kažemo je daje PartsCatalog implementirana u uslovima ove PartsList.

Primetićete da klijenti PartsCatalog nemaju direktan pristup PartsLi st. Interfejs ide kroz PartsCatalog i stoga je ponašanje PartsList dramatično izmenjeno. Na primer, PartsCatalog:: InsertO metod ne dozvoljava duplirane ulaza u PartsList.

Implementacija PartsCatalog::InsertO počinje u liniji 272. Part koji je prosleden kao parametar upitan je za sopstvenu vrednost, i tsPartNumber promenljive člana. Ta vrednost je data Part Li st Fi nd metodu i ako ne postoji odgovarajuća vrednost broj će biti dodat, dok će u drugom slučaju biti prikazana poruka o grešci.

Primetićete da PartCatal og stvarno izvršava dodavanje, pozivanjem InsertO sa njegovom promenljivom članom PI, koja je PartsList. Mehanika stvarnog dodavanja i održavanja povezane liste, kao i pretraživanje i dobijanje podataka iz povezane liste se nalaze u članu PartsList PartsCataloga. Ne postoji razlog PartsCataloga da izvršava ovaj kod. On može da koristi sve prednosti dobro dizajniranog interfejsa.

Ovo je suština višestrukog korišćenja unutar C++-a. PartsCatalog može visestruko da koristi PartsList kod i dizajner PartsCatal oga je Slobodan da ignoriše detalje implementacije PartsList. Interfejs za PartsList (ovo je deklaracija klase) obezbeduje sve informacije koje su neophodne dizajneru klase PartsCatalog.

Privatno nasledivanje

Ako PartsCatalog ima potrebu da pristupa zaštićenim clanovima LinkedList i da izvrši override bilo koje LinkedList metode, tada će PartsCatalog biti primoran da bude nasleden iz PartsList.

S obzirom da PartsCatalog nije PartsList objekat i da Vi ne želite da date ceo niz funkcionalnosti PartsLi sta klijentu PartsCatalog, potrebno je da koristite privatno nasledivanje.

Prva stvar koju treba da znate o privatnom nasledivanju je da se sve bazne promenljive članovi i funkcije tretiraju, kao da su deklarisane kao privatne, što zavisi od njihovog stvarnog nivoa pristupa. Stoga, za bilo koju funkciju koja nije funkcija član PartsCataloga, svaka funkcija nasledena iz PartsList je nepristupačna. Ovo je kritično: privatno nasledivanje ne uključuje nasledivanje interfejsa, nego samo implementaciju.

Za klijente klase PartsCatalog klasa PartsList je nevidljiva. Ni jedan deo njenog interfejsa nije na raspolaganju što će reći da ne možete pozvati ni jedan od njenih metoda. Međutim, Vi možete pozvati metodu PartsCatalog i ona će moći da pristupi celoj klasi LinkedLists, s obzirom daje izvedena iz LinkedLists.

Važna stvar ovde je, da PartsCatalog nije PartsList, kao što bi bilo implicirano javnim nasledivanjem. Ona je implementirana kao PartsList, kao što bi bio slučaj sa kontejnerom. Privatno nasledivanje je samo jedna vrlo zgodna osobina.

Listing 15.6 demonstrira korišćenje privatnog nasledivanja, tako što prepravlja klasu PartsCatalog kao privatno izvedenu iz PartsList.

Da biste kompajlirali ovaj program, zamenite linije 260-344 listinga 15.5. listingom 15.6. i izvršite ponovno kompajliranje.

Listing 15.6: Privatno nasledivanje.

```
1   //listing 15.6 demonstrira privatno nasledivanje
2
3   //prepisuje PartsCatalog iz listinga 15.5
4
5   //pogledajte beleške o kompajliranju
6
7      class PartsCatalog : private PartsList
8      {
9      public:
10        void Insert(Part *);
11        ULONG Exists(ULONG PartNumber);
12        Part * Get(int PartNumber);
13        operator+(const PartsCatalog &);
14        void ShowAllQ { Iterate(Part::Display); }
15     private:
16     };
17
18     void PartsCatalog::Insert(Part * newPart)
19     {
20        ULONG partNumber = newPart->GetPartNumber();
21        ULONG offset;
22
23        if (!Find(offset. partNumber))
24           PartsLi st::Insert(newPart);
25        el se
26        {
```



```
          cout « partNumber « " was the ";
          switch (offset)

             case 0:  cout « "first "; break;
             case 1:  cout « "second "; break;
             case 2:  cout « "third "; break;
             default: cout « offset+1 « "th "•
          }
          cout « "entry. Rejected!\n";



ULONG PartsCatalog::Exists(ULONG PartNumber)

    ULONG offset;
    Find(offset,PartNumber);
    return offset;


Part * PartsCatalog::Get(int PartNumber)

    ULONG offset;
    return (Find(offset, PartNumber));



int main()
{
   PartsCatalog pc;
   Part * pPart = 0;
   ULONG PartNumber;
   USHORT value;
   ULONG choice;

  while (1)
  {
     cout « "(0)Quit (l)Car (2)Plane: ";
     cin » choice;

     if (.'choice)
        break;

     cout « "New PartNumber?: ";
     cin »  PartNumber;

     if (choice == 1)
     {
        cout « "Model Year?: ";
        cin » value;
```

nastavljase

nastavak

dEp^ **Listing 15,6: Privatno nasledivanje.**

```
76:              pPart = new CarPart(value,PartNumber);
77:           }
78:           else
79:           {
80:              cout « "Engine Number?: ";
81:              cin » value;
82:              pPart = new AirPlanePart(value,PartNumber);
83            }
84            pc.Insert(pPart);
85         }
86         pc.ShowAll();
87      return 0;
```

```
(O)Quit (I)Car (2)Plane:   1
New PartNumber?: 1234
Model Year?: 94
(O)Quit (I)Car (2)Plane:   1
New PartNumber?: 4434
Model Year?: 93
(O)Quit (I)Car (2)Plane:   1
New PartNumber?: 1234
Model Year?: 94
1234 was the first entry. Rejected!
(O)Quit (I)Car (2)Plane:   1
New PartNumber?: 2345
Model Year?: 93
(O)Quit (I)Car (2)Plane:   0

Part Number: 1234
Model Year: 94

Part Number: 2345
Model Year: 93

Part Number: 4434
Model Year: 93
```

Ш3EJjJ> Listing 15.6 prikazuje samo izmenjen interfejs za PartCatal og i ponovo napisan drajver program. Interfejsovi za ostale klase su neizmenjeni u odnosu na listing 15.5.

U liniji 7 listinga 15.6 deklarisan je PartsCatalog, tako da se privatno izvodi iz PartsList. Interfejs za PartCatal og nije izmenjen u odnosu na listing 15.5, osim što, naravno, vise nema potrebu za objektom PartsLi st kao podatkom članom.

Funkcija ShowAll() PartsCataloga poziva funkciju Iterate() PartsListe sa odgovarajućim pointerom na funkciju člana klase Part. ShowAll () ponaša se kao Public interfejs za IterateO, obezbeđujući korektne informacije, ali sprečavajući klijente

ove klase da direktno pozivaju IterateO. Iako PartsList može da odobri drugim funkcijama da budu prosledene Iterate()u, PartsCatalog to ne dozvoljava.

Funkcija InsertO je izmenjena. Primetićete da se u liniji 23 Find() sada direktno poziva, pošto je nasledena iz bazne klase. Poziv funkcije Insert () u liniji 24morao je da bude potpuno kvalifikovan, ili bi beskonačno rekurzivno pozivao samog sebe.

Ukratko, kada metodi PartsCataloga žele da pozivaju PartsList metode, one to mogu da učine direktno. Jedini izuzetak je kada PartsCatal og ima metod nadkojim je izvršen override i kada je potrebna verzija iz PartsLi stsa, u kojim slučajevima ime funkcije mora biti potpuno kvalifikovano.

Privatno nasledivanje omogućava PartsCatalogu da nasledi ono što može da koristi, ali mu i dalje omogućava umereni pristup Insert i drugim metodama, kojima klijentove klase nemaju direktan pristup.

<jj PAzm Koristite javno nasledivanje kada je izvedeni objekat vrsta bazne klase.

Koristite kontejnere kada želite da delegirate funkcionalnost drugoj klasi, ali ne želite pristup njenim zaštićenim clanovima.

Koristite privatno nasledivanje kada želite da implementirate klasu sličnu nekoj drugoj i potreban Vam je pristup zaštićenim clanovima bazne klase.

Nemojte koristiti privatno nasledivanje kada želite da koristite vise od jedne bazne klase. U torn slučaju, morate koristiti kontejner. Na primer, ako PartsCatalog ima potrebu za dve PartsLi ste, nećete mod da koristite privatno nasledivanje.

Nemojte koristiti javno nasledivanje kada clanovi bazne klase ne treba da budu na raspolaganju klijentima izvedene klase.

Prijateljske klase

Ponekad ćete imati potrebu da kreirate klase zajedno, tj. u setovima. Na primer, PartNode i PartsList deluju kao par i bilo bi zgodno kada bi PartsList mogla direktno da čita Part pointer PartsNode-a - itsPart.

Nećete želeti da itsPart bude Publ ic, ili, čak, Protected, pošto je ovo implementacioni detalj PartNodea, i želećete da on ostane Private. Međutim, želećete da ga prikažete PartsListi.

Ako želite da prikažete svoj privatni podatak clan, ili funkciju drugoj klasi, moraćete da deklarišete tu klasu kao Friend. Ovim se proširuje interfejs Vaše klase, kako bi uključio prijateljske klase.

Kada PartsNode deklariše PartsList kao prijatelja, svi PartsNode podaci članovi i funkcije će postati javni, dokle god je PartsLi sta u pitanju.

Važno je zapamtiti da se prijateljstvo ne može prenositi. Samo zato što ste Vi moj a Joe Vaš prijatelj, to ne znači da je Joe i moj prijatelj. Prijateljstvo se, takode, ne nasleduje ako: ste Vi moj prijatelj i ja želim da delim moje tajne sa Varna, to ne znači da želim da ih delim i sa Vašom decom.

I na kraju, prijateljstvo nije komutativno. Time što smo dodelili da je klasa One prijatelj sa klasom Two, ne znači i da je klasa Two prijatelj sa klasom One. To što Vi želite da meni saopštite svoje tajne, ne znači da ću ja Varna saopštiti svoje.

Listing 15.7 ilustruje prijateljstvo, tako što je izmenjen primer iz listinga 15.6, pri čemu je PartsList klasa postala prijatelj klase PartNode. Zapamtite da time PartNode nije postao prijatelj PartsListe.

Listing 15.7: Ilustradja prijateljske klase.

```
0:  linclude <iostream.h>
1:
2:  typedef unsigned long ULONG;
3:  typedef unsigned short USHORT;
4:
5:
g.  JI ********** QgQ ***********
7:
8:  // Abstaktna osnovna klasa delova
9:  class Part
10:   {
11:   public:
12:     Part():itsPartNumber(I)  {}
13:     Part(ULONG PartNumber):
14:         itsPartNumber(PartNumber){}
15:     virtual -Part(){}
16:     ULONG GetPartNumber() const
17:         { return itsPartNumber; }
18:     virtual void DisplayO const =0;
19:   private:
20:     ULONG itsPartNumber;
21:   };
22:
23:   // implementacija čiste virtuelne funkcije tako da se
24:   // izvedena klasa može nadovezati
25:   void Part::Display() const
26:   {
27:       cout « "\nPart Number: ";
28:       cout « itsPartNumber « endl;
29:   }
30:
31:   // ********** Automobil ***********
32:
33:   class CarPart : public Part
34:   {
35:   public:
36:       CarPart():itsModelYear(94){}
37:       CarPart(USHORT year, ULONG partNumber);
38:       virtual void DisplayO const
39:       {
40:           Part::Display();
41:           cout « "Model Year: ";
42:           cout « itsModelYear « endl;
43:       }
44:   private:
45:       USHORT itsModelYear;
46:   };
47:
48:   CarPart::CarPart(USHORT year, ULONG partNumber):
49:       itsModelYear(year),
50:       Part(partNumber)
51.   O
52:
53:
      // ********** Avlon

56:   class AirPlanePart : public Part
57:   (
55:   puDIIC:
59:       rPlanePart():itsEngineNumber(1){}
60:     AirPlanePart
61:         (USHORT EngineNumber, ULONG PartNumber)
62:     virtual void DisplayO const
63:     {
64:         Part::Display();
65:         cout « "Engine No.: ";
66:         cout « itsEngineNumber « endl;
67:     }
68:   private:
69:     USHORT itsEngineNumber;
70:   }.
71:
72:   AirPlanePart::AirPlanePart
73:       (USHORT EngineNumber, ULONG PartNumber):
74:     itsEngineNumber(EngineNumber).
75:     Part(PartNumber)
76:   {}
77:
78: // ********** čvor za deo ***
79: class PartNode

81:  public:
82:    friend class PartsList;
83:    PartNode (Part*);
```

nastavlja se

Usting 15.7: Ilustracija prijateljske klase. *nastavak*

```
84:       -PartNodeQ;
85:       void SetNext(PartNode * node)
86:           { itsNext = node; }
87:       PartNode * GetNextQ const;
88:       Part * GetPart() const;
89:    private:
90:       Part  *itsPart;
91:       PartNode * itsNext;
92:    };
93:
94:
95:       PartNode::PartNode(Part* pPart):
96:       itsPart(pPart),
97:       itsNext(O)
98:       {}
99:
100:      PartNode::-PartNode()
101:      {
102:          delete  itsPart;
103:          itsPart = 0;
104:          delete  itsNext;
105:          itsNext = 0;
106:       }
107:
108:       // Vraća NULL ako nema sledećeg PartNode
109:       PartNode * PartNode::GetNext() const
110:       {
111:            return  itsNext;
112:       }
113:
114:       Part * PartNode::GetPart() const
115:       {
116:          if  (itsPart)
117:             return  itsPart;
118:          else
119:             return NULL; //greška
120:       }
121:
122:
123:    // ********** Lista delova ***********
124:    class PartsList
125:    {
126:    public:
127:       PartsListO;
128:       -PartsListO;
129:       // potreban je konstruktor kopije i  operator jednako!
130:       void       Iterate(void  (Part: :*f) Oconst)  const;
131:       Part*     Find(UL0NG & position, ULONG PartNumber) const;
```



```
    Part*     GetFirstQ  const;
    void        Insert(Part  *);
    Part*     operatorQ (ULONG)  const;
    ULONG     GetCount() const { return itsCount; }
    static      PartsList& GetGlobalPartsListQ
              {
                 return   GlobalPartsList;
              }
private:
    PartNode * pHead;
    ULONG itsCount;
    static  PartsList  GlobalPartsList;
};

PartsList  PartsList::G1obalPartsList;

// Implementacije za liste...

PartsList::PartsList():
   pHead(O),
   itsCount(0)
   {}

PartsList: :-PartsList()
(
   delete pHead;
}

Part*    PartsList::GetFirst() const
(
   if  (pHead)
      return pHead->itsPart;
   else
      return NULL;  // ovde se hvata greška


Part * PartsList::operator[](ULONG offSet) const
{
   PartNode* pNode = pHead;

   if  (ipHead)
      return NULL; // ovde se hvata greška

   if  (offSet  >  itsCount)
      return NULL; // greška

   for (ULONG i=0;i<offSet; i++)
      pNode = pNode->itsNext;
```

nastavljase

Listing 15.7: Hustracija prijateljske klase.

nastavak

```
181:      return    pNode->itsPart;
182:    }
183:
184:    Part* PartsList::Find(UL0NG & position, ULONG PartNumber) const
185:    {
186:       PartNode * pNode = 0;
187:       for (pNode = pHead, position = 0;
188:             pNode!=NULL;
189:             pNode = pNode->it$Next, position**)
190:       {
191:          if (pNode->itsPart->GetPartNumber() == PartNumber)
192:             break;
193:       }
194:       if (pNode == NULL)
195:          return NULL;
196:       el se
197:          return pNode->itsPart;
198:
199:
200:    void PartsList::Iterate(void (Part: :*func) Oconst) const
201:    {
202:       if (IpHead)
203:          return;
204:       PartNode* pNode = pHead;
205:       do
206:          (pNode->itsPart->*func)();
207:       while (pNode = pNode->itsNext);
208:
209:
210:    void PartsList::Insert(Part* pPart)
211:    {
212:       PartNode * pNode = new PartNode(pPart);
213:       PartNode * pCurrent = pHead;
214:       PartNode * pNext = 0;
215:
216:       ULONG New = pPart->GetPartNumber();
217:       ULONG Next = 0;
218:       itsCount++;
219:
220:       if (IpHead)
221:
222:          pHead = pNode;
223:          return;
224:       }
225:
226       // ako je ovaj manji od glave
227       // on je nova glava
228       if (pHead->itsPart->GetPartNumber() > New)
```



```
       {
          pNode->itsNext = pHead;
          pHead = pNode;
          return;


       for (;;)
       {
          // ako nema sledećeg, dodaj ovaj novi čvor
          if (!pCurrent->itsNext)
          {
             pCurrent->itsNext = pNode;
             return;
          }

          // ako ovaj dolazi iza ovog i pre sledećeg
          // onda ga umetni ovde, inače uzmi sledećeg
          pNext = pCurrent-HtsNext;
          Next = pNext->itsPart->GetPartNumber();
          if (Next > New)
          {
             pCurrent->itsNext = pNode;
             pNode->itsNext = pNext;
             return;
          }
          pCurrent = pNext;



    class PartsCatalog : private PartsList
    {
    public:
       void Insert(Part *);
       ULONG Exists(ULONG PartNumber);
       Part * Get(int PartNumber);
       operator+(const PartsCatalog &);
       void ShowAll() { Iterate(Part::Display);
    private:


    void PartsCatalog::Insert(Part * newPart)
    {
       ULONG partNumber = newPart->GetPartNumber();
       ULONG offset;

       if (!Find(offset, partNumber))
          PartsList::Insert(newPart);
       el se
```

nastavlja se

S»»JP* Listing 15.7: Ilustracija prijateljske klase.

nastavak

```
278          cout « partNumber « " was the ";
279          switch (offset)
280          {
281             case 0:   cout « "first "; break;
282             case 1:   cout « "second "; break;
283             case 2:   cout « "third "; break;
284             default:  cout « offset+1 « "th ";
285          }
286          cout « "entry.  Rejected!\n";
287
288
289
290       ULONG PartsCatalog::Exists(ULONG PartNumber)
291       {
292          ULONG offset;
293          Fi nd(offset,PartNumber);
294          return  offset;
295
296
297       Part * PartsCatalog::Get(int PartNumber)
298       {
299          ULONG offset;
300          return (Find(offset, PartNumber));
301
302       }
303
304       int main()
305       (
306          PartsCatalog pc;
307          Part * pPart = 0;
308          ULONG PartNumber;
309          USHORT value;
310          ULONG choice;
311
312          while (1)
313          (
314             cout « "(0)Quit (l)Car (2)Plane:   ;
315             cin » choice;
316
317             if (!choice)
318                break;
319
320             cout « "New PartNumber?: ";
321             cin »   PartNumber;
322
323             if (choice == 1)
324             {
325                cout « "Model Year?: ";
326                cin » value;
327                pPart = new CarPart(value,PartNumber);
328             }
329             else
330             {
331                cout « "Engine Number?: ";
332                cin » value;
333                pPart = new AirPlanePart(value,PartNumber);
334             }
335             pc.Insert(pPart);
336          }
337          pc.ShowAll();
338       return 0;
339
```

```
(0)Quit (l)Car (2)Plane:   1
New PartNumber?: 1234
Model Year?: 94
(0)Quit (l)Car (2)Plane:   1
New PartNumber?: 4434
Model Year?: 93
(0)Quit (l)Car (2)Plane:   1
New PartNumber?: 1234
Model Year?: 94
1234 was the first entry.  Rejected!
(0)Quit (l)Car (2)Plane:   1
New PartNumber?: 2345
Model Year?: 93
(0)Quit (l)Car (2)Plane:   0

Part Number: 1234
Model Year: 94

Part Number: 2345
Model Year: 93

Part Number: 4434
Model Year: 93
```

Ц:u\u^+» U liniji 82 klasa PartsList je deklarisana kao prijatelj klase PartNode. Pošto PartsList još nije deklarisana, kompajler će se požaliti da mu ovaj tip nije poznat. Ovaj listing smešta deklaraciju prijateljstva u Publ ic odeljak, ali to nije neophodno; ona može biti smeštena bilo gde u deklaraciji klase, bez izmene značenja naredbe. Zbog ove naredbe, svi privatni podaci članovi i funkcije će biti na raspolaganju svim funkcijama clanovima klase PartsList.

U liniji 160 implementacija funkcije člana GetFi rst () odražava ovu izmenu. Umesto da vrati pHead->GetPart, ova funkcija sada može da vrati bilo koji privatni podatak clan, tako što će napisati pHead->itsPart. Na sličan način, InsertO funkcija sada može da uradi pNode->i tsNext=pHead, umesto pNode->setNext(pHead).

Iako su ovo trivijalne izmene i ne postoje dovoljno dobri razlozi da bi PartsList klasa bila prijatelj klase PartNode, ovde je to ipak uradeno, kako bi se ilustrovala ključna reč friend.

Deklaracije prijateljskih klasa bi trebalo da se koriste vrlo oprezno. Ako su dve klase izuzetno srodne i jedna često mora da pristupa podacima iz druge, tada ima smisla da se koristi ova deklaracija. Obično je jednostavnije koristiti Public metode i na taj način Vam je omogućeno da menjate jednu klasu, bez potrebe za ponovnom kompilacijom drugih.

.^HAPOMIMA^ Često ćete čuti početnike programere u C++ - u kako se žale da Friend deklaracije narušavaju enkapsulaciju, koja je izuzetno važna u objektno-orijentisanom programiranju. To je, naravno, besmislica. Friend deklaracije čine da deklarisani Friend postane deo interfejsa klase. I to nije narušavanje enkapsulacije, nego je javno izvodenje.

Prijateljske klase

Deklarišite jednu klasu da bude prijatelj druge, tako što ćete staviti reč Friend u klasu, čime ćete odobriti prava pristupa. Ja mogu deklarisati da ste Vi moj prijatelj, ali Vi ne možete deklarisati samog sebe za mog prijatelja:

Primer:

```
class PartNode{
publ "ic:
friend class PartList;   // deklariše PartList kao prijatelja klase PartNode
};
```

Prijateljske funkcije

U nekom trenutku može Vam zatrebati da dodelite mvo pristupa, koji će biti ograničen samo na jednu, ili par funkcija neke klase, umesto na celu klasu. Ovo možete uraditi tako što ćete deklarisati funkciju člana druge klase za prijatelja, umesto da deklarišete da Vam je cela klasa prijatelj. U stvari, Vi možete deklarisati bilo koju funkciju za funkciju prijatelja, bez obzira da li je ona funkcija clan druge klase, ili nije.

Prijateljske funkcije i overload operatora

Listing 15.1 obezbeduje klasu string, koja vrši overload operatora +. On. takode, obezbeduje konstruktor, koji uzima constant karakter pointer, pa taj string objekat može biti kreiran iz stringova koji su C stila.

v^MAPOMINA^ C stil stringovi su NULL terminirani nizovi karaktera, kao na primer,

"MyStn•"*" Ic word."

Nećete, međutim, mod da kreirate string C stila (string karaktera) i da na njega dodate drugi string, korišćenjem string objekta, kao što je prikazano u sledećem primeru:

```
char cString[] = {"Hello"};
String sStringC World");
String sStringTwo = cString + sString;  //greška!
```

Stringovi C stila nemaju operator+ nad kojim je izvršen overload. Kao što je objašnjeno u Danu 10, "Napredne funkcije", kada kažete cString+sString, Vi, u stvari, pozivate cString.Operator+(sString). Pošto ne možete da pozovete operator* za stringove C stila, ovim ste prouzrokovali grešku u kompilaciji.

Ovaj problem možete rešiti deklarisanjem prijateljske funkcije, koja će izvršiti overload operatora Plus, ali će uzimati dva string objekta. String C stila će biti konvertovan u string objekat odgovarajućim konstuktorom, a operator će biti pozvan korišćenjem dva string objekta.

¸MAPOMEHA^ Da biste kompajlirali ovaj listing, kopirajte linije 33-123 iz listinga 15.1, iza linije 33 u listingu 15.8.

Listing 15.8: Prijateljski operator*.

```
1:    //Listing 15.8 - prijateljski operatori
2:
3:    #include <iostream.h>
4:    ^include <string.h>
5:
6:    // Osnovna string klasa
7:    class String
8:    {
9:       public:
10:         // konstruktori
11:         StringO;
12:         String(const char *const);
13-         String(const String &);
14:         -StringO:
15:
16:          , prekloo1jeni operatori
             nar & ODerator[](int offset);
             •v.r operator[](int offset) const;
             :ring operator+(const Strmg&);
20:          friend String operator+(const String&, const String&);
21:          void operator+=(const String&)
r2:          String & operator (const String &)
23:
24           // Opšte metode pristupa
25:          int GetLen()const { return itsLen; }
26:          const char * GetStringO const { return itsString;
27:
```

nastavlja se

Listing 15.8: Prijateljski operator*.

```
28      private:
29         String  (int);          // privatni konstruktor
30         char * itsString;
31         unsigned short itsLen;
32
33
34    // kreira novi string dodajući tekući
35    // string rhs stringu
36    String String::operator+(const String* rhs)
37
38       int  totalLen = itsLen + rhs.GetLen();
39       String  temp(totalLen);
40       for (int i = 0; i<itsLen; i++)
41          temp[i] = itsString[i];
42       for (int j = 0; j<rhs.GetLen(); j++,
43          temp[i] = rhs[j];
44       temp[totalLen]='\0';
45       return  temp;
46    }
47
48    // kreira novi string dodajući
49    // jedan string drugom
50    String operator+(const String& lhs, const String& rhs)
51
52       int  totalLen = lhs.GetLen() + rhs.GetLen();
53       String  temp(totalLen);
54       for (int i = 0; i<lhs.GetLen(); i++)
55          temp[i] = lhs[i];
56       for (int j = 0; j<rhs.GetLen();       i++)
57          temp[i] = rhs[j];
58       temp[totalLen]='\0';
59       return  temp;
60
61
62    int  main()
63    {
64       String sl("String One ");
65       String s2("String Two ");
66       char *cl = { "C-String One " } ;
67       String s3;
68       String s4;
69       String s5;
70
71       cout « "Sl:  " « si.GetStringO  « endl;
72       cout « "s2:  " « s2.GetStringO  « endl;
73       cout « "cl:  " « cl « endl;
74       S3 = sl + s2;
75       cout « "s3:  " « s3.GetStringO  « endl;
```


nastavak

```
76:      s4 = sl + cl;
77:      cout « "s4:  "    s4.GetStringO  « endl;
78:      s5 = cl + sl;
79:      cout « "s5:  "    s5.GetStringO  « endl;
80:   return 0;
1:
```

```
sl: String One
s2: String Two
cl: C-String One
s3: String One String Two
s4: String One C-String One
s5: C-String One String Two
```

Implementacija svih string metoda, osim operatora+, nepromenjena je u odnosu na listing 15.1, tako da su izostavljene iz ovog listinga. U liniji 20 izvršen je overload nad operatorom+, kako bi uzimao dve konstantne sting reference i vraćao string, i ova funkcija je deklarisana kao prijateljska.

Primetite da ovaj operator* nije funkcija clan ove, niti bilo koje druge klase. On je deklarisan unutar deklaracije za klasu string, samo zato da bi mogao da se proglasi za prijatelja, ali pošto je deklarisan, ni jedna druga prototip funkcija Vam neće biti potrebna.

Implementacija ovog operatora* je u linijama 50-60. Primetićete da je on sličan prethodnom operators, osim što sada uzima dva stringa, kojima pristupa putem njihovih public metoda.

Drajver program demonstrira korišćenje ove funkcije u liniji 78, gde je operator* sada pozvan stringom C stila.

Prijateljske funkcije

Deklarišite funkciju kao prijateljsku, tako što ćete koristiti ključnu reč `Friend` i, zatim, potpunu specifikaciju funkcije. Deklaracijom da je funkcija prijatelj, nećete toj funkciji dozvoliti pristup Vašem `This` pointeru, ali ćete joj obezbediti potpuni pristup svim privatnim podacima clanovima i funkcijama.

Primer:

```
class PartNode
{
// proglašava funkciju članicu druge klase prijateljem
friend void PartsList::Insert(Part *);
// proglašava globalnu funkciju prijateljem
friend int SomeFunction();
```

Overload Insert operatora

Na kraju ste spremni da Vašoj string klasi date mogućnost da koristi cout, kao i bilo koji drugi tip. Do sada, kada ste želeli da odštampate string, bilo je potrebno da uradite sledeće

```
cout « theString.GetStringO ;
```

Šta će se dogiti ako biste želeli da napišete

```
cout « theString;
```

Da biste ovo izvršili, potrebno je da uradite override operatora «( ). U Danu 16, "Strimovi", biće prikazani ins i outs (cine i couts), koji rade sa iostreams; za sada će listing 15.9 ilustrovati kako se može izvršiti overload operatora « uz pomoć prijatelj skih funkcija.

Da biste kompajlirali ovaj listing, iskopirajte linije 33-153 iz listinga 15.1, iza linije 31 u listingu 15.9.

Listing 15.9: Preklapajud operator«( ).

```
1   linclude <iostream.h>
2   ^include <string.h>
3
4   class String
5   {
6      publi c:
7         // konstruktori
8         StringO;
9          String(const char *const);
10         String(const String &);
11        StringO;
12
13        // preklopljeni operatori
14        char & operators(int offset);
15        char operator[](int offset) const;
          String operator+(const String&);
          void operator+=(const String&);
18        String & operator= (const String &);
          friend ostream& operator«
20           ( ostream& theStream,String& theString);
          // Opšte metode pristupa
22        int GetLen()const { return itsLen; }
23        const char * GetStringO const { return itsString;
24        // static int ConstructorCount;
25     private:
26        String (int);          // privatni konstruktor
27        char * itsString;
28        unsigned short itsLen;
29
```



```
ostream& operator«
    ( ostream& theStream,String& theString)
{
    theStream « theString.GetStringO);
    return theStream;
}
int main()
{
   String theString("Hello world.");
   cout « theString;
 return 0;
}
```

IIIKJjJ^* Hell o world.

Da bismo sačuvali prostor, izostavljena je implementacija svih String metoda, s obzirom da su one nepromenjene u odnosu na prethodne primere.

U liniji 19 operator «je deklarisan tako da bude prijateljska funkcija, koja, uzima ostream referencu i String referencu i, zatim, vraća ostream referencu. Primetićete da ovo nije funkcija clan stringa. Ona vraća referencu na ostream, pa, stoga, možete da sastavite pozive operatora«, kao u sledećem primeru:

```
cout « "myAge: " « itsAge « " years.";
```

Implementacija ove prijateljske funkcije je u linijama 32-35. Sve što ona radi je, u stvari, da sakriva detalje implementacije, koji dostavljaju string ostream-u. 0 overloads ovog operatora i operatora» vise ćete naučiti u Danu 16.

Rezime

Danas ste videli kako da delegirate funkcionalnost sadržanom objektu. Takođe ste naučili kako da implementirate jednu klasu u uslovima druge, korišćenjem bilo kontejnera, bilo privatnog nasledivanja. Kontejneri su ograničeni time da nova klasa nema pristup zaštićenim clanovima sadržane klase i ne može da izvrši override funkcija članova sadržanih objekata. Kontejneri su jednostavniji za korišćenje od privatnog nasledivanja i trebalo bi ih koristiti kad god je to moguće.

Takođe ste videli kako se deklarišu prijateljske funkcije i prijateljske klase. Uz pomoć prijateljskih funkcija videli ste kako se može izvršiti overload operatora, koji će Vam omogućiti da Vaša nova klasa koristi cout, na isti način kao što to rade i ugradene klase.

Zapamtite da se javno nasledivanje izražava sa/e, da se kontejner izražava sa *ima,* dok se privatno nasledivanje izražava sa *implementiran u uslovima od.* Relacija *dclcgira nekome* može biti izražena bilo korišćenjem kontejnera, bilo privatnog nasledivanja, dok su kontejneri ipak češće u upotrebi.

Pitanja i odgovori

P Zašto je važno napraviti razliku između *je, ima* **i** *implemenHran* **u** *uslovima od?*

0 Svrha C + + je da implementira dobro dizajnirane, objektno-orijentisane programe. Držeći se čvrsto ovih relacija, Vi ćete mod da se uverite da Vaš dizajn odgovara realnosti, na osnovu koje vršite modeliranje. I, nadalje, dobro razumljiv dizajn **će** biti odražen u dobro dizajniranom kodu.

P Zašto je bolje koristiti kontejner, umesto privatnog nasledivanja?

O Izazov u modernom programiranju je izbegavanje kompleksnosti. Što ste vise u mogućnosti da koristite objekte kao "erne kutije", biće manje detalja o kojima ćete morati da brinete i, samim tim, manje kompleksnosti, sa kojom treba da se borite. Sadržane klase skrivaju svoje detalje; privatno nasledivanje prikazuje detalje implementacije.

P Zašto da sve klase ne budu prijatelji **sa svim klasama koje koriste?**

O Kada jedna klasa postane prijatelj druge, ona prikazuje detalje implementacije i ograničava enkapsulaciju. Ideja je da se što vise detalja svake klase zadrži što skrivenijim od drugih klasa.

P Ako je izvršen overload funkcije, **da li moram da** deklarišem svaku **formu** funkcije **kao** prijatelj**sku?**

O Da. Ako izvršite overload funkcije i deklarišete je kao prijatelja neke druge klase, neophodno je da sa Friend deklarišete svaku formu kojoj želite da date pristup.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što pređete na sledeće poglavlje.

Pitanja

1. Kako ćete ostvariti *je* relaciju?
2. Kako ćete ostvariti *ima* relaciju?
3. Koja razlika je između kontejnera i delegacija?
4. Koja razlika je između delegacije i *implementiran u uslovima od?*

5. Šta je prijateljska funkcija?
6. STA je prijateljska **klasa?**
7. **Ako** je Dog prijatelj Boy-u, **da li** je Boy prijatelj Dog-u?
8. Ako je Dog prijatelj Boy-u i Terrier se izvodi iz Dog-a, da li je Terrier prijatelj Boy-a?
9. Ako je Dog prijatelj Boy-a i Boy je prijatelj House-a, da li je Dog prijatelj House-a?
10. Gde bi deklaracija prijateljske funkcije morala da se pojavi?

Vežbe

1. Prikažite deklaraciju klase Animal sa podatkom članom, koji je string objekat.
2. Prikažite deklaraciju klase BoundedArray, koja je niz.
3. Prikažite deklaraciju klase Set, koja je deklarisana u uslovima od jednog niza.
4. Modifikujte listing 15.1, kako biste omogućili klasi String da koristi operator za ekstrakciju (»).

ISTERIVČI BAGOVA: šta nije u redu sa ovim programom:

```
1    linclude <iostream.h>
2
3    class Animal;
4
5    void setValue(Animal& , int);
6
7
8    class Animal
9;   {
10   public:
11      int GetWeight()const { return itsWeight;
12      int GetAgeO const { return itsAge; }
13   private:
14      int itsWeight;
15      int itsAge;
16
17
18   void setValue(Animal& theAnimal, int theWeight)
19   {
20      friend class Animal;
21      theAnimal.itsWeight = theWeight;
22   }
23
24   int main()
25   {
26     Animal peppy;
27      setValue(peppy,5);28:    }
```

Ispravite listing iz vežbe 5, tako da prode kompilaciju.

ISTERIVAČI BAGOVA: šta nije u redu sa sledećim kodom:

```
1:    #include <iostream.h>
2:
3:    class Animal;
4:
5:    void setValue(Animal& , int);
6:    void setValue(Animal& ,int,int);
7:
8:    class Animal
9:    {
10:    friend void setValue(Animal& ,int);ll:     private:
12:       int itsWeight;
13:       int itsAge;
14:   };
15:
16:    void setValue(Animal& theAnimal, int theWeight)
17:   {
18:        theAnimal.itsWeight = theWeight;
19:   }
20:
21:
22:    void setValue(Animal& theAnimal, int theWeight, int theAge
23:   {
24:       theAnimal.itsWeight = theWeight;
25:       theAnimal.itsAge = theAge;
26:   }
27:
28:    int main()
29:   {
30:       Animal peppy;
31:       setValue(peppy,5);
32:       setValue(peppy,7,9);
83:   }
```

Ispravite vežbu 7. tako da prode kompilaciju.

Iff!
O
S

/Dan 16

Strimovi

Do sada ste koristili cout za pisanje na ekranu i cin za čitanje sa tastature, a da niste u potpunosti razumeli kako oni rade. Danas ćete naučiti

- šta su strimovi i kako se koriste
- kako se, pomoću strimova, upravlja input-om i output-om
- kako se pišu fajlovi i čitaju uz pomoć strimova.

Pregled strimova

C+ + , kao deo jezika ne defmiše kako se podaci prikazuju na ekranu, ili u datoteci, niti kako se podaci učitavaju u program. Medutim, ovo su suštinski delovi u radu sa C + + i standardna C+ + biblioteka sada uključuje iostream biblioteku, koja treba da olakša input i output (U/I).

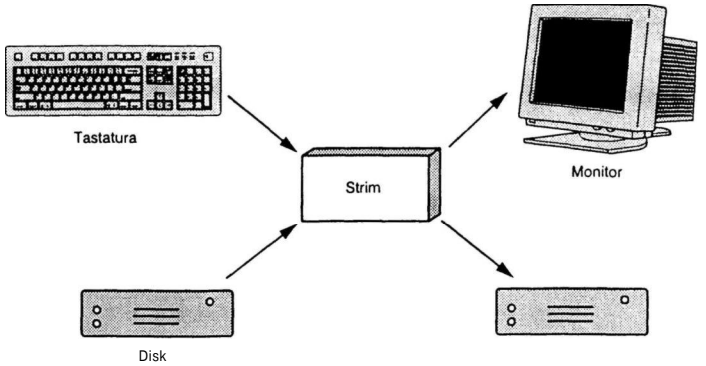
Prednost smeštanja input-a i output-a van jezika i njihova podrška iz biblioteka je što je to jednostavniji način da se jezik izgradi kao nezavisan od platforme. To znači da možete pisati C+ + programe na PC-u i da ih, zatim , ponovo kompajlirate i startujete na Sun radnoj stanici. Proizvođač kompajlera Vam samo isporuči pravu biblioteku i sve radi. Tako je bar u teoriji.

^**MAPOMEMA**^ Biblioteka je kolekcija OBJ datoteka, koje mogu biti povezane sa Vašim programima, da bi omoguc'ile dodatnu funkconalnost. Ovo je najjednostavnija forma ponovnog korišćenja koda i bila je prisutna još od kada su rani programeri "klesali' jedinice i nule na zidovima "pec'ina."

Enkapsulacija

Iostream klase gledaju na protok podataka iz Vašeg programa na ekran kao na "mlaz" podataka, u kojima jedan bajt sledi drugi. Ako je odredište strima datoteka, ili ekran, izvor je, obično, neki deo Vašeg programa. Ako je strim u suprotnom smeru, tada podaci mogu doći od tastature, ili iz datoteke sa diska i biće isporučeni u Vase promenljive, koje sadrže podatke.

Glavni cilj korišćenja strimova je enkapsulacija problema, preuzimanje podataka na disk, ili ekran., ili sa njih. Kada je strim kreiran, Vaš program radi sa strimom, dok se strim brine o detaljima. Na slici 16.1 prikazana je osnova ove ideje.



Slika 16.1.
Enkapsulacija
kroz strimove

Baferovanje

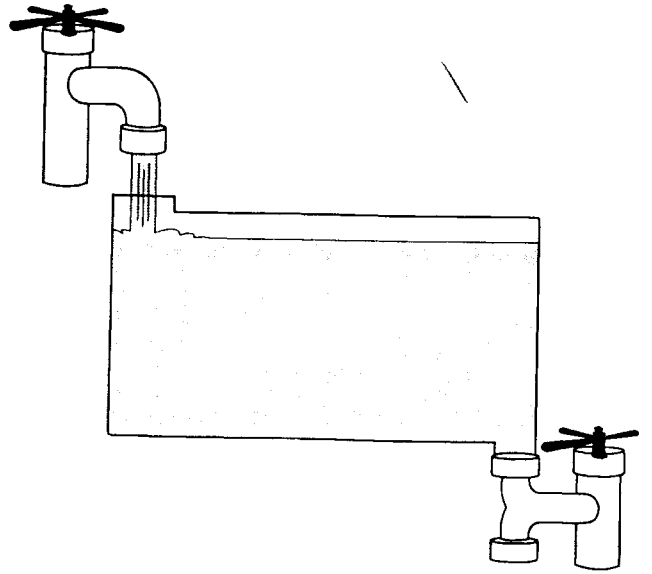
Upisivanje na disk je vrlo skupo. Prilično vremena treba za upis na disk, ili čitanje sa diska, a izvršenje programa je, obično, blokirano tim upisima i čitanjima. Da bi rešili ovaj problem, strimovi obezbeduju baferovanje. Podatak je upisan u strim, ali nije istog trenutka i vraćen nazad na disk. Umesto toga, bafer strima se neprekidno popunjava i tek kada je pun upisuje se na disk, ceo i odjednom.

Zamislite vodu koja curi u otvor na vrhu rezervoara i rezervoar se popunjava, a voda ne izlazi na dnu. Na slici 16.2 prikazana je ova ideja.

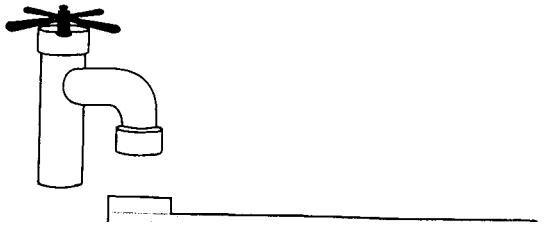
Kada voda (podaci) dosegne vrh rezervoara, otvara se ventil i sva voda ističe velikom brzinom. Ovo je ilustrovano na slici 16.3.

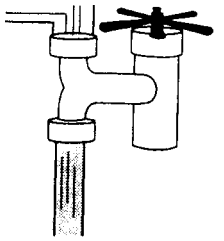
Kada se bafer isprazni, ventil na dnu se zatvara, zatim se ventil na vrhu otvara i nova voda dolazi u bafer. Ovo je ilustrovano na slici 16.4.

Povremeno je potrebno da ispraznite rezervoar, čak i ako nije napunjen. Ovo se naziva flushing bafera. Na slici 16.5 prikazana ja ova ideja.

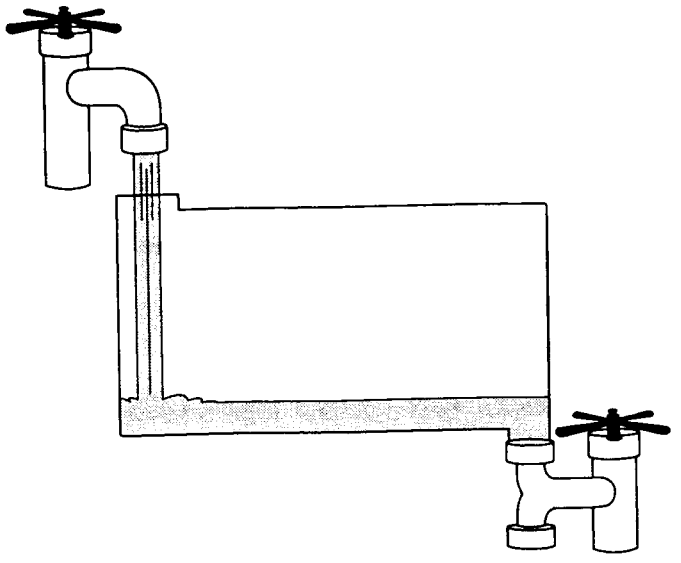


Slika 16.2.
Punjenje bafera

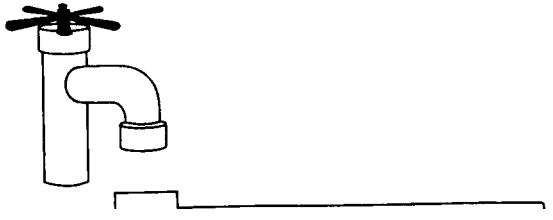


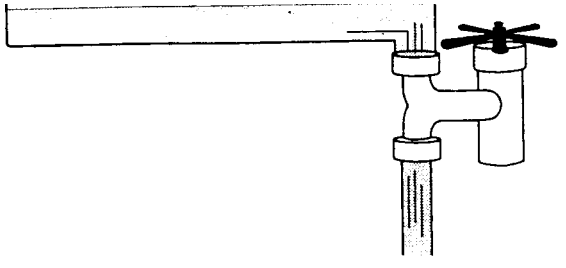


°''ika 16.3.
froinjenje bafera



Slika 16.4.
Ponovno
punenje bafera





Slika 16.5.
Flushing bafera

Strimovi i baferi

Kao što ste i očekivali, C + + koristi jedan objektno-orijentisani pristup za implementaciju strimova i bafera:

- klasa streambuf upravlja baferima i/njene funkcije članovi Vam pružaju mogućnost da napunite, ispraznite, izvršite flush i na neki drugi način manipulišete baferom
- klasa ios je bazna za ulazne i izlazne stream klase. Ona ima streambuf objekat kao promenljivu člana
- klase i stream i ostream su izvedene iz ios klase i vrše specijalizaciju ponašanja ulaznog i izlaznog strima respektivno
- klasa iostream je izvedena iz klasa istream i ostream i obezbeduje ulazne i izlazne metode za rad sa ekranom
- klasa iostream obezbeduje ulazne i izlazne metode za rad sa datotekama.

Standardni U/I objekti

Kada se startuje C + + program koji, sadrži U/I stream klase, dolazi do kreiranja četiri objekta i njihove inicijalizacije.

^pAPOЖИи^ Biblioteku U/I stream klase kompajler automatski dodaje Vašem programu. Sve što je potrebno da biste koristili ove funkcije je da dodate odgovarajuću `include` naredbu na vrh Vašeg programa:

- cin (čita se "`si-in`"), podržava ulaz iz standardnog ulaza, tastarure
- cout (čita se "`si-aut`"), podržava izlaz na standardni izlaz, ekran
- cerr (čita se "`si-er`"), podržava nebaferovani izlaz na standardnu jedinicu za grešku, ekran, s obzirom da je ovo nebaferovano, sve što je poslato sa cerr se upisuje na standardni uredaj za grešku, trenutno bez čekanja na to da se bafer napuni, ili da se izvrši komanda flesh.
- cl og (čita se si -1 og), podržava baferovane poruke o greškama, koje se upućuju na standardni uređaj za greške, ekran; za njih je uobičajeno da se izvrši redirekcija u log datoteku, kao što će biti opisano u sledećem odeljku.

Redirekcija

Svi standardni uredaji ulaza, izlaza i greške mogu se redirektovati u druge uredaje. Standardna greška se često preusmerava u datoteku, dok standardni ulaz i izlaz mogu biti povezani sa datotekama, korišćenjem komandi operativnog sistema.

Redirekcija predstavlja slanje izlaza (ili ulaza) na mesto koje se razlikuje od podrazumevanog. Operatori za redirekciju i u DOS-u i UNIX-u su: (<) vrši redirekciju ulaza i (>) vrši redirekciju izlaza.

Piping označava korišćenje izlaza jednog programa kao ulaza za drugi.

DOS obezbeduje najjednostavnije komande za redirekciju, kao što su *redirekcija izlaza* (>) i *redirekcija ulaza* (<). UNIX obezbeduje mnogo naprednije mogućnosti, ali je opšta ideja ista: preuzeti izlaz koji je namenjen ekranu i upisati ga u datoteku, ili ga povezati sa nekim drugim programom. Alternativno, ulaz iz programa može biti ekstraktovan iz datoteke, umesto sa tastature.

Redirekcija je vise funkcija operativnog sistema, nego U/I stream biblioteke. C++ Vam samo obezbeduje pristup svakom od ova Četiri standardna uredaja; korisnik vrši redirekciju uređaja na alternative koje su mu potrebne.

Ulaz uz pomoć naredbe cin

Globalni objekat cin je odgovoran za input i postaje dostupan Vašem programu kada uključite i ostream. U prethodnim primerima koristili ste operator za ekstrakciju, nad kojim je izvršen overload (»), da biste smestili podatke u promenljive Vašeg programa. Kako ovo radi? Sintaksa je, ako se setite, sledeća:

```
int someVariable;
cout « "Enter a number: ";
cin » someVariable;
```

0 globalnom objektu cout će kasnije biti reči; za sada, usmerimo pažnju na treću liniju cin» someVariable: šta mislite o cin?

Jasno je da se radi o globalnom objektu, budući da ga niste definisali u Vašem kodu. Poznato Vam je iz prethodnog iskustva u radu sa operatorima da je ci n izvršio overload nad operatorom za ekstrakciju (») i daje efekat od ovoga, da se upisuje bilo koji podak koje ci n ima u svom baferu u Vašu lokalnu promenljivu someVari abl e.

Ono što nije tako očigledno je da je cin izvršio overload operatora za ekstrakciju za veliki niz različitih vrsta parametara: int&, short&, long&, double&, float&, char&, char* i tako dalje. Kada napišete cin » someVariable biće prihvaćen tip someVariable. U prethodnom primeru someVariable je integer, tako da će biti pozvana sledeća funkcija:

```
istream & operator» (int &)
```

Primetićete, s obzirom da je parametar prosleden po referenci, da operator za ekstrakciju može da deluje nad originalnom promenljivom. Korišćenje naredbe cin je ilustrovano u listingu 16.1.

Listing 16.1: cin rukuje razlifltim tipovima podataka.

```
1:    //Listing 16.1 - stringovi karaktera i cin
2:
3:    linclude <iostream.h>
4:
5:    int mainQ
6:    {
7:       int mylnt;
8:       long myLong;
9:       double myDouble;
10:   float my Float;
11:      unsigned int myUnsigned;
12
13       cout « "int: ";
14:       cin » mylnt;
15•       cout « "Long: ";
16:       cin » myLong;
17        cout « "Double: ";
18        cin » myDouble;
19        cout « "Float: ";
20        cin » myFloat;
21        cout « "Unsigned: ";
22        cin » myUnsigned;
23.
24:       cout « "\n\nInt:\t" « mylnt « endl;
25:       cout « "Long:\t" « myLong « endl;
26:       cout « "Double:\t" « myDouble « endl;
27:       cout « "Float:\t" « myFloat « endl;
28:       cout « "Unsigned:\t" « myUnsigned « e
29:    return 0;
30:   }
```

```
1ITTTT^  int:  2
         Long:  70000
         Double: 987654321
         Float:  3.33
         Unsigned: 25

         Int:    2
         Long:    70000
         Double: 9.87654e+08
         Float:  3.33
         Unsigned:       25
```

U linijama 7-11 deklarisane su promenljive različitih tipova. U linijama 13-22 od korisnika je traženo da unese vrednosti za ove promenljive i rezultat je, zatim, prikazan (korišćenjem cout) u linijama 24-28.

Izlaz reflektuje, da su promenljive bile smeštene u odgovarajuću vrstu promenljivih l program funkcioniše, kao što se i moglo očekivati.

"ZP** Stringovi

Cin takode podržava karakter pointer (char*) argumente. Stoga, Vi možete kreirati karakter bafer i koristiti **cin,** kako biste ga popunili. Na primer, možete napisati sledeće:

```
char YourName[50]
cout « "Enter your name: ";
cin » YourName;
```

Akounesete **Jesse,** promenljiva **YourNamece** biti popunjena karakterima **J, e, s, s, e,** \0. Poslednji karakter je NULL karakter**-cin** automatski završava stringove njime i neophodno je da imate dovoljno prostora u baferu, kako bi stao ceo string plus NULL karakter. NULL karakter signalizira kraj stringa standardnim bibliotečkim funkcijama, o kojima će se diskutovati u Danu 21, "Šta dalje?"

Problemi sa stringovima

Posle svih ovih uspeha sa **cin**-om, verovatno ćete biti iznenadeni kada pokušate da unesete Vaše ime i prezime u string. Naime, **cin** veruje da je razmak između imena i prezimena separator. Kada ugleda razmak, ili **NewLine,** on smatra da je ulaz za taj parametar kompletiran i u slučaju stringova dodaće NULL karakter, baš na to mesto. U listingu 16.2 ilustrovan je ovaj problem.

Listing 16.2: Pokušaj da se zapiše vise reči na cin.

```
1   //Listing 16.2 - stringovi karaktera i cin
2
3   linclude <iostream.h>
4
5   int main()
6   {
7      char YourName[50];
8      cout « "Your first name:
9      cin » YourName;
10     cout « "Here it is: " « YourName « endl;
11     cout « "Your entire name: ";
12     cin » YourName;
13     cout « "Here it is: " « YourName « endl;
14   return 0;
15
```

```
Your first name: Jesse
Here it is: Jesse
Your entire name: Jesse Liberty
Here it is: Jesse
```

U liniji 7 kreiran je niz karaktera koji će sadržati ono što korisnik unosi. U liniji 8 korisnik je upitan da unese svoje ime i ono je smesteno kako treba, a što je prikazano u izlazu.

U liniji 11 od korisnika je traženo da unese i ime i prezime. **Cin** je učitao ulaz i kada je video prazan karakter između imena i prezimena, on je smestio NULL karakter posle prve reči i terminirao ulaz. Ovo baš i nije ono što smo nameravali.

Da biste razumeli zašto se ovo dogada, ispitajte listing 16.3, koji prikazuje ulaz za veći broj polja.

Listing 16.3: Višestruki ulaz.

```
1   //Listing 16.3 - stringovi karaktera i cin
2
3   linclude <iostream.h>
4
5   int main()
6   {
7      int myInt;
8      long myLong;
9      double myDouble;
10     float myFloat;
11     unsigned int myUnsigned;
12     char myWord[50];
13
14     cout « "int: ";
15     cin » myInt;
16     cout « "Long: ";
17:    cin » myLong;
18     cout « "Double: ";
19     cin » myDouble;
20     cout « "Float: ";
21     cin » myFloat;
22     cout « "Word: ";
23     cin » myWord;
24     cout « "Unsigned: ";
25     cin » myUnsigned;
26
27     cout « "\n\nInt:\t" « myInt « endl;
28     cout « "Long:\t" « myLong « endl;
29     cout « "Double:\t" « myDouble « endl;
30     cout « "Float:\t" « myFloat « endl;
31     cout « "Word: \t" « myWord « endl;
32     cout « "Unsigned:\t" « myUnsigned « endl;
33
34     cout « "\n\nInt, Long, Double, Float, Word, Unsigned:
35     cin » myInt » myLong » myDouble;
36     cin » myFloat » myWord » myUnsigned;
37     cout « "\n\nInt:\t" « myInt « endl;
38     cout « "Long:\t" « myLong « endl;
39     cout « "Double:\t" « myDouble « endl;
40     cout « "Float:\t" « myFloat « endl;
41     cout « "Word: \t" « myWord « endl;
```

nastavljase

Listing 16.3: Višestruki ulaz. *nastavak*

```
42        cout « "Unsigned:\t" « myUnsigned « endl;
43
44
45      return 0;
46    }
```

```
Int: 2
Long: 30303
Double: 393939397834
Float: 3.33
Word: Hello
Unsigned: 85

Int:    2
Long:   30303
Double: 3.93939e+II
Float:  3.33
Word:   Hello
Unsigned:       85

Int, Long, Double, Float, Word, Unsigned: 3 304938 393847473 6.66 bye -2

Int:    3
Long:   304938
Double: 3.93847e+08
Float:  6.66
Word:   bye
Unsigned:       65534
```

Kreirano je nekoiiko promenljivih, ovaj put uključujući i niz karaktera. Od korisnika je traženo da unese podatke koji su zatim prikazani.

U liniji 34 od korisnika je traženo da unese sve podatke odjednom i zatim je svaka reč iz tog ulaza dodeljena odgovarajućoj promenljivoj. Ovo je u skladu sa osobinom višestrukog dodeljivanja, o kojoj ci n mora da vodi računa, jer svaka reč u ulazu predstavlja kompletan ulaz za svaku promenljivu. Ako bi cin smatrao da je kompletan ulaz deo jedne promenljive, ova vrsta konkatenovanog ulaza ne bi bila moguća.

Primetićete u liniji 35 da je poslednji objekat koji je zahtevan bio neoznačeni integer, ali je korisnik uneo -2. Pošto cin veruje da treba da upiše neoznačeni integer, bit šema broja - 2 je evaluirana kao neoznačeni integer i kada je, zatim, ta vrednost ispisana sa cout, prikazano je 65.534. Neoznačena vrednost 65.534 ima istu bit šemu kao i označena vrednost -2.

Kasnije, u ovom poglavlju, videćete kako se unose celi stringovi u bafer, uključujući i višestruke reči. Za sada, samo postoji pitanje: "Kako operator za ekstrakciju upravlja ovim trikom konkatenacije."

operator» vraća referencu na istream objekat.

Vrednost koju cin vraća je referenca na istream objekat. Pošto je i cin sam istream objekat, vraćena vrednost jednog operatora za ekstrakciju može biti ulaz za sledeću ekstrakciju.

```
int VarOne, varTwo, varThree;
cout « "Enter three numbers: "
cin » VarOne » varTwo » varThree;
```

Kada napišete cin » VarOne » VarTwo » VarThree izvršiće se prva ekstrakcija ci n»Var0ne. Vraćena vrednost je jedan istream objekat i operator za ekstrakciju tog objekta preuzima promenljivu VarTwo. To je kao da ste napisali sledeće:

```
((cin » varOne) » varTwo) » varThree;
```

Ostale funkcije članovi cin-a

Osim što vrši overload operatora», cin ima i odredeni broj drugih funkcija operatora. One se koriste kada je zahtevana finija kontrola nad ulazom.

Unos jednog karaktera

operator», koji uzima karakter referencu, može se koristiti za preuzimanje jednog karaktera iz standardnog ulaza. Funkcija član get() se, takođe, može koristiti za dobavljanje jednog karaktera, na dva načina: bez parametara, u kom slučaju možete koristiti vraćenu vrednost, ili sa referencom na karakter.

Korišćenje Get () bez parametara

Prva forma funkcije get() je bez parametara. Ona će vratiti vrednost pronadenog karaktera, ili će vratiti EOF (kraj datoteke), ako je dosegnut kraj datoteke. Funkcija get() se ne koristi često bez parametara. Nije moguće vršiti konkatenaciju ove verzije funkcije, s obzirom da vraćena vrednost nije U/I stream objekat. Stoga, sledeće neće funkcionisati:

```
cin.get() »myVar0ne » myVarTwo; //    neispravno
```

Vraćena vrednost naredbe cin.Get()»MyVarOne je integer, a ne U/I objekat.

Uobičajeno korišćenje funkcije get() bez parametara je ilustrovano u listingu 16.4.

Listing 16.4: Korišćenje get () bez parametara.

```
// Listing 16.4 - Koriščenje get() bez parametara
linclude <iostream.h>

int  main()
{
```

nastavlja se

^^^^^ **Listing 16.4: Korišćenje get() bez parametara.** *nastavak*

```
    char ch;
    while ( (ch = cin.getO) !- EOF)
    {
       cout « "ch: " « ch « endl;
    }
    cout « "\nDone!\n";
  return 0;
}
```

lupoid Da biste napustili ovaj program, morate poslati kod za EOF sa tastature. Na DOS kompjuterima koristite Ctrl+Z, dok na UNIX-u treba pritisnuti Ctrl* + D.

```
Hello
ch: H
ch: e
ch: 1
ch: 1
ch: o
ch:

World
ch: W
ch: o
ch: r
ch: 1
ch: d
ch:

(ctrl-z)
Done!
```

Д Ш Е ^ > U liniji 6 deklarisana je lokalna karakter promenljiva. Petlja While dodeljuje ulaz primljen iz cin.Get() promenljivoj ch i ako taj ulaz nije EOF, string će bit prikazan. Ovaj izlaz je baferovan, sve dok se ne dostigne EOL (end of line). Kada se dostigne EOF (pritiskom na Ctrl+Z) na DOS mašinama, odnosno Ctrl+D na UNIX mašinama, napustićete petlju.

Primetićete da postoje istream implementacije, koje ne podržavaju ovu verziju funkcije get().

Korišćenje get() sa referencom na karakter parametar

Kada je karakter prosleden kao ulaz za get(), on se puni narednim karakterom u ulaznom strimu. Vraćena vrednost je iostream objekat, tako da ovaj oblik get()-a može biti konkateniran, kao što je prikazano u listingu 16.5.

Listing 16.5: Korišćenje get() sa parametrima.

```
// Listing 16.5 - Korišćenje get() sa parametrima
linclude <iostream.h>

int main()
{
   char a, b, c;

   cout « "Enter three letters: ";

   cin.get(a).get(b).get(c);

   cout « "a: " « a « "\nb: " « b « "\nc: " « c « endl;
 return 0;
}
```

```
p H > ; У ^ Enter three letters: one
          a: o
          b: n
          c: e
```

U liniji 6 kreirana su tri karakter promenljive. U liniji 10 cin.getO je pozvan tri puta konkatovanim podacima. Prvo je pozvan cin.get (a). Ovo je smestilo prvo slovo u promenljivu a i cin je završio sa radom. Zatim je pozvan cin.get (b), čime je smešteno sledeće slovo u b. Rezultat ovoga je zatim pozivano cin.get (c) i treće slovo je smešteno u promenljivu c.

Pošto cin.get (a) evaluira u cin, nije bilo potrebe da napišete sledeće:

cin.get(a) » b;

U ovoj formi cin.get (a) evaluira u cin, tako da je druga faza cin » b ;

<JT **мпп** Koristite operator za ekstrakciju > >, kada želite da preskočite razmake.

Koristite get() sa karakter parametrom, kada je potrebno da ispitate svaki karakter, uključujući i prazna mesta. Nemojte koristiti get() bez parametara. Ova verzija je manje-više zastarela.

Preuzimanje stringova sa standardnog ulaza

Operator ekstrakcije » može se koristiti za popunjavanje niza karaktera, baš kao i funkcije članovi get(), getLineO .

Poslednja forma get() preuzima tri parametra. Prvi parametar je pointer na niz karaktera, drugi parametar je maksimalan broj karaktera koji treba pročitati + jedan i treći parametar je karakter za terminaciju.

Ako unesete 20 za drugi parametar, get() će učitati 19 karaktera, zatim će dodati NULL karakter i smestiti ga i prvi parametar. Treći parametar je karakter za terminaciju. Podrazumevana vrednost je newline (\n). Ako se karakter za terminaciju dosegne pre maksimalnog broja karaktera za čitanje, NULL će biti upisan, a karakter za terminaciju će biti ostavljen u baferu.
U listingu 16.6. ilustrovano je korišćenje ove forme get ().

```
1:  // Listing 16.6 - Korišćenje get() sa nizom karaktera
2:  linclude <iostream.h>
3:
4:  int main()
5:  {
6:     char stringOne[256];
7:     char stringTwo[256];
8:
9:     cout « "Enter string one: ";
10:    cin.get(string0ne,256);
11:     cout « "stringOne: " « stringOne «  endl;
12:
13:    cout « "Enter string two: ";
14:     cin » stringTwo;
15:     cout « "StringTwo: " « stringTwo «  endl;
16:   return 0;
17: }
```

```
Enter string one: Now is the time
stringOne: Now is the time
Enter string two: For all good
StringTwo: For
```

U linijama 6 i 7 kreirana su dva niza karaktera. U liniji 9 od korisnika je traženo da unese string, a u liniji 10 pozvan je ci n.get ( ). Prvi parametar je bafer, koji treba napuniti, a drugi je za jedan veći od maksimalnog broja koji će get() prihvatiti - (dodatna pozicija će biti data NULL karakteru (\0)). Podrazumevani treći parametar je newl i ne.

Korisnik je uneo Now is the time, s obzirom da je završio frazu sa newline, koja je smeštena u stringOne i iza nje je dodat terminator NULL.

Korisnik je, zatim, upitan za još jedan string u liniji 13, a ovoga puta je korišćen operator za ekstrakciju. Pošto operator za ekstrakciju preuzima sve karaktere do prvog praznog mesta, stringFor i NULL karakter su smešteni u drugi string, što, naravno, nije ono što smo nameravali.

Još jedan od načina za rešavanje ovog problema je korišćenje GetLine(), kao što je ilustrovano u listingu 16.7.

Listing 16.7: Korišćenje getl i ne().

```
// Listing 16.7 - Korišćenje getline()
linclude <iostream.h>

int mainQ
{
   char string0ne[256];
   char stringTwo[256];
   char stringThree[256];

   cout « "Enter string one: ";
   cin.getline(string0ne,256);
   cout « "stringOne: " « stringOne « endl;

   cout « "Enter string two: ";
   cin » StringTwo;
   cout « "StringTwo: " « StringTwo « endl;

   cout « "Enter string three: ";
   cin.getline(stringThree,256);
   cout « "stringThree: " « stringThree « endl;
return 0;
}
```

```
1          Enter string one: one two three
           stringOne: one two three
           Enter string two: four five six
           StringTwo: four
           Enter string three: stringThree: five six
```

1.lMl^jjJE* Ovaj primer zahteva pažljivo ispitivanje, s obzirom da postoji nekoliko potencijalnih iznenadenja. U linijama 6-8 deklarisana su tri niza karaktera.

U liniji 10 od korisnika je traženo da unese string, koji je procitan uz pomoć GetLine(O). Kao i get(), GetLineO uzima za parametre bafer i max. broj karaktera. Za razliku od get(), NewLine će biti pročitan i izbačen. Sa get() NewLine neće biti izbačeno,već ostavljeno u ulaznom baferu.

U liniji 14 od korisnika je ponovo traženo da unese podatke, a ovoga puta je korišćen operator za ekstrakciju. Korisnik je uneo "four, five, six" i prva reč "for" je smeštena u string two. String "Enter string three" je, zatim, prikazan i ponovo je pozvana GetLineO . Pošto je "five six" i dalje u ulaznom baferu, ono je trenutno učitano sve do karaktera newl i ne.

GetLi ne() je terminirano i string iz promenljive string three je prikazan u liniji 20.

Korisnik nije imao šansu da unese treći string, s obzirom daje drugi poziv GetLi ne() popunjen preostalim delom stringa u ulaznom baferu, a koji je preostao posle poziva operatora za ekstrakciju u liniji 15.

Operator za ekstrakciju (») je čitao do prvog praznog mesta i smestio je reč u niz karaktera. Izvršen je overload funkcije člana get(). U jednoj verziji ona ne uzima parametre i vraća vrednost karaktera koji je primila. U drugoj verziji ona uzima jednokarakternu referencu i vraća i strim objekat po referenci.

U trećoj finalnoj verziji get() preuzima niz karaktera, broj karaktera koji treba da preuzme i karakter za terminaciju (podrazumevano newline). Ova verzija get() učitava karaktere u niz, sve dok ne dostigne broj jedan manji od maksimalnog broja karaktera, ili dok ne naide na karakter za terminaciju, bez obzira šta će se prvo desiti. Ako get() naide na karakter za terminaciju, ostaviće taj karakter u ulaznom baferu i prestaće sa čitanjem karaktera.

Funkcija član get(), takođe, uzima tri parametra: bafer koji će puniti, jedan 4- maksimalan broj karaktera koje treba uzeti i karakter za terminaciju. GetLine() funkcioniše, baš kao i get(), osim što GetLine0 odbacuje karakter za terminaciju.

KORIŠĆENJE CIN.IGNORE()

Postoje situacije kada ćete želeti da ignorišete preostale karaktere linije, pre eol-a, ili eof-a. Za ovo se koristi funkcija clan Ignore(). Funkcija Ignore() uzima dva parametra, maksimalan broj karaktera koje treba ignorisati i karakter za terminiranje. Ako napišete Ignore (80, "\n"), do 80 karaktera će biti odbačeno, dok se ne pronade karakter newline. Zatim će i karakter newline biti odbačen i naredba Ignore0 će završiti sa radom. U listingu 16.8 ilustrovano je korišćenje naredbe Ignore0 .

Listing 16.8: Korišćenje ignore().

```
1:    // Listing 16.8 - Korišćenje ignore()
2:    linclude <iostream.h>
3:
4:    int main()
5:    {
6:       char stringOne[255];
7:       char stringTwo[255];
8:
9:       cout « "Enter string one:";
10:      cin.get(string0ne,255);
11:      cout « "String one" « stringOne « endl;
12:
13:      cout « "Enter string two: ";
14:      cin.getline(stringTwo,255);
15:      cout « "String two: " « StringTwo « endl;
16:
17:      cout « "\n\nNow try again...\n";
18:
19:      cout « "Enter string one: ";
20:      cin.get(string0ne,255);
21:      cout « "String one: " « stringOne« endl;
22:
23:      cin.ignore(255,'\n')i
24:
25:      cout « "Enter string two: ";
26:      cin.getline(stringTwo,255);
27:      cout « "String Two: " « stringTwo« endl;
28:   return 0;
29:   }
```

```
ИЗЕЗ|^  Enter string one:once upon a time
        String oneonce upon a time
        Enter string two: String two:

        Now try again^..
        Enter string one: once upon a time
        String one: once upon a time
        Enter string two: there was a
        String Two: there was a
```

}.шп/.^ U linijama 6 i 7 kreirana su dva niza karaktera. U liniji 9 od korisnika je traženo da unese tekst "Once upon a time", koji je sledio Enter. U liniji 10 naredba Get() je iskorišćena za čitanje tog stringa. Naredba Get() je popunila string one i terminirala ga sa newline, ali je ostavila karakter newline u ulaznom baferu.
U liniji 13 od korisnika je ponovo traženo da unese tekst, ali je funkcija GetLine u liniji 14 pročitala karakter newline, koji je (pre)ostao u ulaznom baferu, i izvršenje programa je terminirano, pre nego što je korisnik mogao bilo šta da unese.
U liniji 19 korisnik je ponovo upitan i uneo je jednu liniju. Medutim, ovoga puta u liniji 23 naredba Ignore0 je iskorišćena da "pojede" newline karakter. Stoga, kada je poziv za GetLine0 dosegnut u liniji 26, ulazni bafer je bio prazan i korisnik je mogao da unese sledeću liniju price.

PeekQ iPutBack()

Ulazni objekat cin poseduje dva dodatna metoda, koji se mogu zgodno iskoristiti: Peek(), koji traži karakter, ali ne izvlači sledeći karakter, i PutBackf), koji insertuje karakter u izlazni string. U listingu 16.9 ilustrovano je kako se ovi metodi mogu koristiti.

Listing 16.9: Korišćenje peekQ i putback().

```
// Listing 16.9 - Koriščenje peekQ i putbackQ
linclude <iostream.h>

int mainQ
{
   char ch;
   cout « "enter a phrase: ";
   while ( cin.get(ch) )
```

nastavljase

Usting 16.9: Korišćenje peek() i putbackQ. *nastavak*

```
            if (ch == '!')
               cin.putback(ž$');
            else
               cout « ch;
            while (cin.peek() == '#')
               cin.ignore(l,'#');
         }
      return 0;
8:    }
```

IZLAZ

```
enter a phrase: Now!isIthe!timeIfor!funI!
Now$i sthe$timefor$fun$
```

ANALIZA U liniji 6 deklarisana je karakter promenljiva ch, a u liniji 7 od korisnika je traženo da unese frazu. Svrha ovog programa je da zameni bilo koji uzvičnik (!) dolarskim znakom ($), kao i da ukloni sve znakove (#).

Program se vrti u petlji, sve dok dobija karaktere koji su različiti od eof-a (zapamtite da cin.getO vraća 0 za eof). Ako je tekući karakter znak uzvika (!), on će se odbaciti i znak dolar ($) će biti smešten u ulazni bafer. Ako tekući znak nije bio uzvičnik, on će biti prikazan takav kakav jeste.

Ovo nije baš najefikasniji način za rešavanje ovih problema, (ne bi pronašao čak ni znak #, ako bi bio na prvoj poziciji). Ali, ipak je izvršena ilustracija rada ovih metoda. Oni su relativno komplikovani i zato nemojte trošiti previse vremena, sve dok ne dođete u situaciju da ih stvarno upotrebite. Smestite ih u Vašu kutiju trikova, postaće praktični pre, ili kasnije.

SAVn *II,* PeekQ i PutBack() se, obično, koriste za prosledivanje stringova i drugih podataka kada se piše kompajler.

Izlaz sa cout

Do sada ste koristili cout, zajedno sa Insert operatorom («), nad kojim je izvršen overload, da biste ispisivali stringove, integer-e i druge numeričke podatke na ekran. Takode je moguće formatizovati podatke, poravnati kolone i ispisivati numeričke podatke u decimalnom i heksadecimalnom obliku. U sledećem odeljku će biti objašnjeno kako se ovo radi.

Flushing izlaza

Do sada ste već videli da, korišćenjem end naredbe, možete izvršiti flush izlaznog bafera. Endl poziva cout-ovu funkciju člana flush(), koja ispisuje sve baferovane podatke. Metod flush() možete pozvati direktno, bilo pozivanjem člana flush() metoda, bilo na sledeći način:

```
cout « flush
```


Ovo može biti zgodno kada želite da se uverite da je izlazni bafer ispražnjen, i da je njegov sadržaj ispisan na ekranu.

Srodne funkcije

Baš kao što operator za ekstrakciju može da ima get() i getlineO, tako i Insert operator može da ima put() i write().

Funkcija put() se koristi za upisivanje jednog karaktera na izlaznu jedinicu. S obzirom da put() vraća ostream referencu i pošto je cout ostream objekat, možete izvršiti konkatenaciju putQ . Listing 16.10 ilustruje ovu ideju.

Listing 16.10: Korišćenje putQ.

```
// Listing 16.10 - Korišćenje put()
linclude <iostream.h>

int main()
{
   cout.put('H').put('e').put(4').put(4').p ut('o').put('\n')'
return 0;
```

Linija 6 se izvršava na sledeći način: cout.put ("H") upisuje H na ekran i vraća cout objekat. Preostaje sledeće:

```
cout.put('e').put('1').put('Г).put('o').put('\n');
```

Zatim se ispisuje slovo e, ostavljajući cout.put("1"). Proces se ponavlja-svako slovo se prikazuje na ekranu, sve dok se ne dode do poslednjeg karaktera ("\n"). Funkcija write() radi potpuno isto kao i Insert operator («), osim što uzima parametar koji odreduje maksimalan broj karaktera koji mora da prikaže. U listingu 16.11 ilustrovano je njeno korišćenje.

Listing 16.11: Korišćenje writeQ.

```
// Listing 16.11 - Koriščenje writeQ
linclude <iostream.h>
linclude <string.h>

int mainQ
{
   char One[] = "One if by land";


   int fullLength = strlen(One);
```

nastavlja se

Listing 16.11: Korišćenje write( ) .

```
12:       int tooShort = fullLength -4;
13:       int tooLong = fullLength + 6;
14:
15:       cout.write(One,ful1 Length)  «  "\n";
16:       cout.write(One,tooShort)  «  "\n";
17:       cout.write(One.tooLong)  «  "\n";
18:     return 0;
19:   }
```

```
ШШПjШ& One if by land
 ' O  n   e if by
        One if by land i?!
```

„ДШЧ>МНА^ Poslednja linija izlaza se može razlikovati na Vašem kompjuteru.

Ш Ш J ^ ^ nm ^ 7 kreirana J e j ecma fraza- U n n ^ 11 integer-u ful1 Lenght je dodeljena vrednost dužine fraze, a tooShort-u je dodeljena dužina –4, dok je tooLong-u dodeljena ful 11 ength plus 6.

U liniji 15 kompletna fraza je prikazana korišćenjem wri te ( ). Dužina je postavljena na stvarnu dužinu fraze i fraza je korektno prikazana.

U liniji 16 fraza je ponovo odštampana, ali je za četiri karaktera kraća od pune fraze i to se odrazilo i u izrazu.

U liniji 17 fraza je ponovo prikazana, ali ovoga puta je od write zahtevano da prikaže dodatnih sedam karaktera. Kada je fraza prikazana, sledećih sedam bajtova memorije koja sledi je takođe napisano.

Manipulator!, flag-ovi i instrukcije za formatizaciju

Izlazni stream održava veći broj statusnih flag-ova, određuje koju bazu (decimalnu, ili heksadecimalnu) treba koristiti, određuje širinu polja i karaktere koji će se koristiti za popunjavanje polja. *State Flag* je bajt, čiji individualni bitovi imaju specijalno značenje. Manipulisanje ovim bitovima će biti objašnjeno u Danu 21. Svaki od ostream-ovih flagova može biti postavljen, korišćenjem funkcija cianova, ili manipulatora.

Korišćenje cout.width ()

Podrazumevana širina Vašeg izlaza će biti dovoljan prostor za prikaza brojeva, karaktera, ili stringova u izlaznom baferu. Ona se može promeniti, korišćenjem width0. S obzirom da je width0 funkcija clan, ona mora biti pozvana sa cout objektom. Ona samo vrši izmenu širine sledećeg izlaznog polja i zatim se odmah vraća na podrazumevanu vrednost. U listingu 16.12 prikazano je njeno korišćenje.

Listing 16.12: Prilagođavanje širine izlaza.

```
      // Listing 16.12 - Prilagođavanje Sirine izlaza
2:    linclude <iostream.h>
3:
4:    int  main()
5:    {
6:       cout « "Start >";
7:       cout.width(25);
8:       cout « 123 « "< End\n";
9:
10:      cout « "Start >";
11:      cout.width(25);
12:      cout « 123« "< Next >";
13:      cout « 456 « "< End\n";
14:
15:      cout « "Start >";
16:      cout.width(4);
17:      cout « 123456 « "< End\n
18:
19:    return 0;
20:   }
```

```
Start                    123< End
Start                    123< Next >456< End
Start  -123456< End
```

Prvi izlaz u linijama 6-8 prikazuje broj 123 unutar polja, čija širina je postavljena na 25 u liniji 7. Ovo je prikazano u prvoj liniji izlaza.

Druga linija izlaza, najpre, prikazuje vrednost 123 u istom polju čija širina je postavljena na vrednost 25, i, zatim, prikazuje vrednost 456. Primetićete da je 456 prikazano u polju čija je širina postavljena da bude dovoljno velika; kao što je receno, efekat width() je prestao odmah čim je vrednost prikazana.

Poslednji izlaz prikazuje da je postavljanje širine koja je manja od izlaza isto kao i postavljanje širine koja je dovoljno velika.

Postavljanje karaktera za popunjavanje

U normalnim uslovima cout popunjava prazna polja na dužinu width() sa praznim mestima, kao što je već ranije prikazano. Vremenom ćete požci-n da niz popunite nekim drugim karakterima, kao što je. na primer, Da biste to ucinili, pozovite funkciju fi 11 () i prosledite joj, kao parametar, karakter kojim zehte da bude popunjena. Ovo je ilustrovano u listingu 16.13.

Listing 16.13: Korišćenje fill 0»

```
// Listing 16.3 - fill()

linclude <iostream.h>

int main()
{
   cout « "Start >";
   cout.width(25);
   cout « 123 « "< End\n"


    cout « "Start >";
   cout.width(25);
   cout.filKž*');
   cout « 123 « "< End\n"
 return 0;
```

```
WETuth* Start >           123< End
        Start >******************123< End
```

ТДТТФ^ Linije 7-9 ponavljaju funkcionalnost iz prethodnog primera. U linijma 12-15 ponovljeno je to isto, ali ovoga puta u liniji 14 za karakter za popunjavanje odredena je *, kao što se vidi u izlazu.

Postavljanje flag-ova

Objekti U/I stream prate sopstveno stanje, korišćenjem flag-ova. Ovi flag-ovi se mogu menjati pozivanjem funkcije `setf()`, tako što ćete joj proslediti jednu, ili vise predefinisanih konstanti.

СДшШЈ3j Za objekat se kaže da ima *stanje* kada neki, ili svi od njegovih podataka zadovoljavaju *uslov* koji se može menjati tokom rada programa.

Na primer, možete postaviti da li će se, ili ne prikazivati nevažeće nule (tako da bi 20,00 bilo skraćeno na 20). Da biste uključili nevažeće nule, pozovite `setf (ios:: showpoi nt).`

Predefinisane konstante se nalaze u klasi `iostream (ios)` i stoga se one pozivaju sa punom kvalifikacijom, `ios:: f 1 agname,` ili, na primer, `ios:: showpoi nt.`

Možete uključiti znak + ispred pozitivnih brojeva, korišćenjem `ios:: showpos.` Možete promeniti podešavanje izlaza, korišćenjem `ios: .-left, ios:write,` ili `ios: .-internal.`

Na kraju, možete postaviti bazu za brojeve za prikaz, korišćenjem `ios: :dec` (decimalno), `ios`::oct (oktalno), ili `ios::hex` (heksadecimalno). Ovi `flag-ovi,` takođe, mogu biti konkatenirani u `Insert` operatoru, a ilustrovani su u listingu 16.14. Kao dodatak, listing 16.14, takođe, predstavlja manipulator `setw,` koji postavlja širinu, `ali` se može iskoristiti i za konkatenaciju sa `Insert` operatorom.

Listing 16.14: Koriscenje setf.

```
1:   // Listing 16.14 - Koriscenje setf
2:   linclude <iostream.h>
3r   linclude <iomanip.h>
4:
5:   int main()
6:   {
7:      const int number = 185;
8:      cout « "The number is " « number « endl;
g.
10:     cout « "The number is " « hex «   number « endl;
11:
12:     cout.setf(ios::showbase);
13:     cout « "The number is " « hex «   number « endl;
14:
15:     cout « "The number is " ;
16:     cout.width(IO);
17:     cout « hex « number « endl;
18:
19:     cout « "The number is " ;
20:     cout.width(IO);
21:     cout.setf(ios::1eft);
22:     cout « hex « number « endl;
23:
24:     cout « "The number is " ;
25:     cout.width(IO);
26:     cout.setf(ios::internal);
27:     cout « hex « number « endl;
28:
29:     cout « "The number is:" « setw(10) « hex « number « endl;
30:  return 0;
31:
```

```
The number is 185
The number is b9
The number is 0xb9
The number is          0xb9
The number is 0xb9
The number is Ox         b9
The number is:0x         b9
```

U liniji 7 konstanta `i nt number` je inicijalizovana na vrednost 185. Ovo je prikazano u liniji 8.

Vrednost je ponovo prikazana u liniji 10, ali ovoga puta manipulatorom `hex,` koji je omogućio da vrednost bude prikazana kao `heksadecimal na, tj. B9 (B=11; 11*16=176+9=185).`

U liniji 12 postavljen je `f 1 ag showbase.` Ovo je prouzrokovalo pojavljivanje prefiksa `0x` *heksadecimalnim* brojevima, kao što je prikazano u izlazu.

U liniji 16 širina je postavljena na 10 i vrednost je ekstremno uvećana. U liniji 20 širina je postavljena ponovo na 10, ali u ovom slučaju poravnanje je postavljeno nalevo i brojevi su prikazani levo poravnati.

U liniji 25 jo§ jedanput je širina postavljena na 10, ali u ovom slučaju poravnanje je postavljeno na internal. Stoga, 0x je odštampano sa leve strane, a vrednost B9 sa desne.

Na kraju, u liniji 29, operator za konkatenaciju setw() je iskorišćen da bi postavio širinu na 10 i vrednost je ponovo prikazana.

Stream-ovi protiv funkcije printf()

Većina implementacija C++ podržava standardne C U/I biblioteke, uključujući printf() naredbu. Iako je printf u nekim slučajevima jednostavnija za koriscenje od cout-a, ipak je nije preporučljivo koristiti.

Funkcija printf() ne obezbeduje sigurnost tipova, tako da joj je moguće red da prikaže integer, iako je on, u stvari, bio karakter i obrnuto. Funkcija printf() takođe ne podržava klase i stoga je nije moguće naučiti kako da prikaže podakte Vase klase, odnosno, da biste prikazali clanove klase sa printf(), morate je snabdevati podacima pojedinačno.

Sa druge strane, printf() Vam omogućava jednostavnu formatizaciju, s obzirom da karaktere za formatizaciju možete direktno smestiti u printf() naredbu. Mnogi programed koriste printf() funkciju i zbog toga ćemo u ovom odeljku razmotriti njenu upotrebu. Da biste koristitli funkciju printf(), obavezno uključite STDIO.H hederdatoteku. U njenoj najprostijoj formi funkcija printf() uzima string za formatizaciju kao prvi parametar I, zatim, seriju vrednosti kao preostale parametre.

String za formatizaciju su tekst i specifikatori za konverziju, koji se nalaze pod navodnicima. Svi specifikatori za konverziju moraju da započnu znakom %. Uobičajeni specifikatori za konverziju su prikazani u tabeli 16.1.

Tabela 16.1: Jednostavni specifikatori za konverziju.

Specifikator	Koristi se za
%s	strings
%d	integers
%1	long integer
%ld	long integers
%f	float

Svaki specifikator za konverziju obezbeduje i naredbe za širinu i preeiznost, koje se izražavaju kao float, gde su cifre levo od decimala korišćene za ukupnu širinu. a cifre desno od decimalne zapete odreduju preeiznost. Stoga, *-sć* predstavlja specifikatora za petocifreni integer, a procenat (%) 15.5f predstavlja specifikator za 15-cifreni float, od čega je poslednjih pet cifara odredeno za decimalni deo. U listingu 16.15 ilustrovane su različite verzije korišćenja funkcije printf().

Listing 16.15: Stampanje sa printf().

```
1:    linclude <stdio.h>
2:    int main()
3:    {
4:       printf("4s","hello  world\n");
5:
6:       char *phrase = "Hello again!\n";
7:       printf("%s",phrase);
8:
9:       int x = 5;
10:      printf("%d\n",x);
11:
!2:      char *phraseTwo = "Here's some values: *";
13:      char *phraseThree = " and also these: ";
14:      int y = 7, z = 35;
15:      long longVar = 98456;
16:      float floatVar =  8.8;
17:
18:      printf("%s %d %d %s %ld %f\n",phraseTwo,y,z,phraseThree,1ongVar,f1oatVar)·
19:
20:      char *phraseFour = "Formatted: ";
21:      printf("%s %5d %10d  %10.5f\n",phraseFour.y,z,f1oatVar);
22:   return 0;
23:   }
```

```
:|jj1Lžfe>  Пel1₀ WOrld
            Hello again!
            5
            Here's some values: 7 35  and also these: 98456 8.800000
            Formatted:      7       35     8.800000
```

ЗД.Ш^_> Prva Printf 0 naredba u liniji 4 koristi standardni oblik: izraz printf koga sledi string pod navodnicima, sa specifikatorom za konverziju (u ovom slučaju %s), iza koga sledi vrednost koju treba uneti u specifikator za konverziju.

Oznaka %s ukazuje da se radi o stringu i da je njegova vrednost, u ovom slučaju, string pod navodnicima "Hel1o, wor1d."

Druga printf() naredba je slična prvoj, ali ovoga puta je iskorišćen char pointer, umesto stringa pod navodnicima, na istom mestu u printf() naredbi.

Treća printf() naredba u liniji 10 koristi specifikator za celobrojnu konverziju i za njegovu vrednost celobrojne promenljive x. Četvrta printf() naredba u liniji 18 je dosta složenija. U njoj se nalazi šest konkatovanih vrednosti. Za svaku je obezbeden specifikator za konverziju, a vrednosti su razdvojene zapetama.

Na kraju, u liniji 21, specifikatori za formatizaciju su iskorišćeni za odredivanje širine i preciznosti. Kao što možete da vidite, sve ovo je jednostavnije od korišćenja manipulatora.

Kao što je ranije rečeno, ograničenja su ovde, medutim, u tome, što ne postoji provera tipa i što printf ne može biti deklarisana kao prijateljska, ili funkcija član klase. Stoga, ako želite da prikazujete različite podatke clanove klase, moraćete da napravite nezavisnu printf() naredbu za svaki podatak ponaosob.

Ulazi i izlazi iz datoteke

Stream-ovi obezbeduju jedinstven rad sa podacima koji dolaze sa tastature, ili sa hard diska, ili odlaze na ekran, ili hard disk. U svakom slučaju, možete koristiti Insert operatore i operatore za ekstrakciju, ili neke druge odgovarajuće funkcije, ili manipulatore. Da biste otvorili, ili zatvorili datoteku, kreiraćete ifstream i ofstream objekte, kao što će biti opisano u sledećih nekoliko odeljaka.

Ofstream

Objekti koji se koriste za čitanje, ili upisivanje u datoteke se nazivaju ofstream objekti. Oni su izvedeni iz iostream objekata, koje ste već koristitli.

Da biste počeli upisivanje u datoteku, neophodno je da, najpre, kreirate ofstream objekat i da, zatim, dodelite taj objekat odredenoj datoteci na Vašem disku. Da biste koristili ofstream objekte, neophodno je da uključite fstream, f u program.

yJAPOMIMA^ Pošto fstream.**h** uključuje iostrem**.h,** nije potrebno da eksplicitno uključite i ostream.**h.**

Stanja uslova

Objekti iostream održavaju flag-ove koji nas obaveštavaju o stanju našeg ulaza, ili izlaza. Svaki od njih možete proveriti, korišćenjem logičkih funkcija eof(), bad(), fail() i good(). Funkcija eof() vraća True, ako je iostream objekat naišao na EOF, kraj datoteke. Funkcija bad() vraća True, ako ste pokušali neispravnu operaciju. Funkcija fail() vraća True svaki put kada je bad() vratio True, ili operacija nije uspela. Na kraju, funkcija good() će vratiti True svaki put kada sve tri preostale funkcije vrate False.

Otvaranje datoteka za ulaz i izlaz

Da biste datoteku myfile.cpp otvorili ofstream objektom, deklarišite kopiju ofstream objekta i prosledite ime datoteke kao parametar:

```
ofstream fout("myfile.cpp");
```


Otvaranje ove datoteke za ulaz funkcioniše na potpuno isti način, osim što koristi ifstream objekat:

```
ifstream fin("myfile.cpp");
```

Primetićete da su fout i fin imena koja ste dodelili; ovde je fout iskorišćeno, kako bi odrazilo svoju sličnost sa cout, i fin, i cin.

Još jedna važna stream funkcija koja, Vam je neophodna, je cl ose (). Svaki file stream objekat koji kreirate otvara datoteku, bilo za čitanje, bilo za pisanje (ili za obe namene). Veoma je važno da zatvorite datoteku sa Close(), pošto završite njeno čitanje, ili pisanje; ovim ste se obezbedili da datoteka neće biti oštećena i da će podaci koje ste pisali biti upisani na disk.

Listing 16.16: Otvaranje datoteka za citanje i pisanje.

```
1:  linclude <fstream.h>
2:  int main()
3:  {
4:     char fileName[80];
5:     char buffer[255];     // za korisnički ulaz
6:     cout « "File name: ";
7:     cin » fileName;
8:
9:     ofstream fout(fileName);  // otvara za pisanje
10    fout « "This line written directly to the file...\n";
11    cout « "Enter text for the file: ";
12    cin.ignore(l,'\n');  // uklanja novi red posle imena datoteke
13;   cin.getline(buffer,255);  // uzima korisnički ulaz
14:   fout « buffer « "\n";    // i zapisuje ga u datoteku
15:
16:   fout.close();             // zatvara datoteku, spremna je za novo otvaranje
17:
18:   ifstream fin(fileName);     // novo otvaranje za čitanje
19:   cout « "Here's the contents of the file:\n";
20:   char ch;
21:   while (fin.get(ch))
22:      cout « ch;
23:
24:   cout « "\n***End of file contents.***\n";
25:
26:   fin.close();            // uvek donosi urednost
    return 0;
27: }
```

```
WEEjfefr      """e: testl
Enter text for the file: This text is written to the file-
Here s the contents of the file:
This line written directly to the file...
This text is written to the file!

**End of file contents.***
```

f I'MjHiffc^ U liniji 4 postavljen je bafer, nezavisan od imena datoteke, a u liniji 5 postavljen je bafer, nezavisan od ulaza korisnika. Od korisnika je traženo da unese ime datoteke u liniji 6 i to je upisano u bafer f i 1 eName. U liniji 9 kreiran je objekat ofstream, pod imenom fout, koji je dodeljen novom imenu datoteke. Ovim je otvorena datoteka; ako je već postojala, njen sadržaj je odbačen.

U liniji 10 string je upisan direktno u datoteku, dok je u liniji 11 od korisnika traženo da unese podatke. Karakter newline, koji je ostavio korisnik u imenu datoteke, "pojeden" je u liniji 12 i podaci koje je korisnik uneo su smešteni u buffer u liniji 13. Taj ulaz je upisan u datoteku, zajedno sa newl i ne karakterom u liniji 14 i zatim je datoteka zatvorena u liniji 15.

U liniji 17 datoteka je ponovo otvorena, ovoga puta u `input` modu, i njen sadržaj je pročitan, karakter, po karakter, u linijama 20 i 21.

Izmena podrazumevanog ponasanja ofstream pri otvaranju

Podrazumevane) ponašanje pri otvaranju datoteke je da se kreira datoteka, ako ona ne postoji, ili da se datoteka isprazni, ako već postoji. Ako ne želite da koristite ovo podrazumevano ponašanje, potrebno je da eksplicitno obezbedite argument konstruktoru Vašeg ofstream objekta.

Ispravni argumenti su:

- `ios::app` - dodavanje na kraj postojeće datoteke, umesto njenog pražnjenja
- `ios:-.ate` - pomera Vaš na kraj datoteke, ali Vi ne možete dodavati podatke bilo gde u datoteku
- `ios::trunc` - podrazumevani argument, koji prouzrokuje pražnjenje postojeće datoteke
- `ios::nocreate` - ako datoteka ne postoji, otvaranje neće uspeti
- `ios::norepl ace` - ako datoteka već postoji, otvaranje neće uspeti.

Primetićete da je `app` skraćenica za `apend` (dodavanje), `ate` za `end` i `trunc` za `truncate`. Listing 16.17 ilustruje koriscenje dodavanja, tako što će ponovo otvoriti datoteku iz listinga 16.16 i dodati slogove u nju.

Listing 16.17: Nadovezivanje na kraj datoteke.

```
1:    linclude <fstream.h>
2:    int main()    // vraća 1 za grešku
3:    {
4:       char fileNane[80];
5:       char buffer[255];
6:       cout << "Please re-enter the file name: ":
7:       cm >> fi leName;
```



```
9-       ifstream fin(fi1eName);
         "if (fin)                // već postoji?
11:      {
12:         cout « "Current file contents:\n";
13:         char ch;
14:         while (fin.get(ch))
15:            cout « ch;
!6:         cout « "\n***Enc of file contents.***\n"
17:      ,
18:      fin.close();
2C:      cout « "\n0pemng " « fileName « " in append mode...\n";

22:      ofstream fout(fi1eName,ios::app);
23:      if (Jfout)
24:      {
25:         cout « "Unable to open " « fileName « " for appending.\n";
26:         return(1);
27:      }
28:
29:      cout « "\nEnter text for the file: ";
30:      cin.ignore(1,*\n');
31:      cin.getline(buffer,255);
32:      fout « buffer « "\n";
33:      fout.close();
34:
35:      fin.open(fileName);  // ponovno dodeljivanje postojećem fin objektu'
36:      if (!fin)
37:      I
38:         cout « "Unable to open " « fileName « " for reading.\n";
39:         return(I);
40:      }
41:      cout « "\nHere's the contents of the file:\n";
42:      char ch;
43:      while (fin.get(ch))
44:         cout « ch;
45:      cout « "\n***End of file contents.***\n";
46:      fin.close();
47:      return 0;
48:   }
```

```
IIIIIIjJI^  P1ease re-enter the file name: testl
               Current file contents:
               This line written directly to the file...
               This text is written to the file!

               ***End of file contents.***

               Opening testl in append mode...
```

```
Enter text for the file: More text for the file!

Here's the contents of the file:
This line written directly to the file...
This text is written to the file!
More text for the file!

***End of file contents.***
```

jJIMUAfcfc Korisnik je ponovo upitan za ime datoteke. Ovoga puta, ulazni stream za datoteku je kreiran u liniji 9. Testiranje datoteke je uradeno u liniji 10 I, ako datoteka postoji, njen sadržaj je prikazan u linijama 12-16. Primetićete da je `if (fin)` sinonim za `if (fin.goodO)`.

Ulazna datoteka je, zatim, zatvorena i ista datoteka je ponovo otvorena, ali ovoga puta u append modu, u liniji 22. Posle ovog otvaranja (i bilo kojeg drugog), datoteka je testirana, kako bismo se uverili da li je otvorena na odgovarajući način.

Primetićete da je if (!fout) isto kao i testiranje if (`fout.fail()`). Od korisnika je, zatim, traženo da unese tekst i datoteka je ponovo zatvorena u liniji 33.

Na kraju, kao i u listingu 16.16, datoteka je ponovo otvorena u modu za čitanje; medutim, ovoga puta, `fin` nije moralo da bude ponovo deklarisano. Ono je samo ponovo dodeljeno istom imenu datoteke. Ponovno je testirano otvaranje datoteke u liniji 36 i, ukoliko je uspelo, sadržaj datoteke je prikazan na ekranu i datoteka je konačno zatvorena.

PAIII1 j|5?f Testirajte svako otvaranje datoteke, kako biste se uverili da je ona uspešno otvorena.

Koristite postojeće i `fstream` i ofstream objekte.

Zarvorite sve f stream objekte, kada završite rad sa njima.

Nemojte pokušavati da zarvorite, ili promenite dodeljivanje ci n-u, ili `cout`-u.

Binarne protiv tekstualnih datoteka

Neki operativni sistemi, kao što je DOS, razlikuju tekstualne i binarne datoteke. Tekstualne datoteke smeštaju sve kao tekst (kao što ste i pretpostavili), tako da se veliki brojevi, kao, na primer, 54,325, smeštaju kao stringovi numerici ("5", "4", "3", "2", "5"). Ovo može biti vrlo neefiksno, ali ima svoju prednost, što tekst može biti učitan korišćenjem jednostavnih programa, kao što su programi DOS tipa.

Da bi pomogao file sistemu da uoči razliku između tekstualnih i binarnih datoteka, C+4- obezbeduje `ios::binary` flag. U većini sistema ovaj flag se ignoriše, s obzirom da su svi podaci smešteni u binarnom formatu. U nekim drugim sistemima flag ios::binary je nelegalan i neće proći kompilaciju.

Binarne datoteke ne samo da mogu da čuvaju integer-e i stringove, nego i cele strukture podataka. Možete upisati sve podatke odjednom, korišćenjem write() metoda fstream-a.

Ako koristite write(), podatke možete povratiti uz pomoć read(). Svaka od ovih funkcija očekuje pointer na karakter; međutim, Vi možete dodeliti adresu na Vašu klasu, da bude pointer na karakter.

Drugi argument za ove funkcije je broj karaktera koji treba da se upiše, a možete ga odrediti korišćenjem sizeof (). Primetićete da je ono što će biti upisano jesu samo podaci, a ne i metodi. Takode, ono što će biti ponovo učitano su samo podaci. U listingu 16.18 ilustrovano je upisivanje sadržaja klase u datoteku.

Listing 16.18: Zapisivanje klase u datoteku.

```
1:  linclude <fstream.h>
2:
3:  class Animal
4:  {
5:  public:
6:    Animal(int weight, long days):itsWeight(weight),itsNumberDaysAlive(days){}
7:    čAnimalOO
8:
9:    int GetWeightQconst { return itsWeight; }
10:   void SetWeight(int weight) { itsWeight = weight; }
11:
12:   long GetDaysAliveQconst { return itsNumberDaysAlive; }
13:   void SetDaysAlive(long days) { itsNumberDaysAlive = days; }
14:
15: private:
16:   int itsWeight;
17:   long itsNumberDaysAlive;
18: };
19:
20: int main()   // vraća 1 za grešku
21: {
22:   char fileName[80];
23:   char buffer[255];
24:
25:   cout « "Please enter the file name: ";
26:   cin » fileName;
27:   ofstream fout(fi 1 eName, ios: -.binary);
28:   if (!fout)
29:   {
30:     cout « "Unable to open " « fileName « " for writing.\n";
31:     return(1);
32:   }
33:
34:   Animal Bear(50,100);
```

nastavlja se

Listing 16.18: Zapisivanje klase u datoteku. *nastavak*

```
fout.write((char*)  &Bear,sizeof  Bear);

fout.closeO;

ifstream   fin(fileName,ios::binary);
if  (!fin)
   cout « "Unable to open " « fileName « " for reading.\n";
   return(l);
}

Animal  BearTwo(l,1);

cout « "BearTwo weight:  " « BearTwo.GetWeight() « endl;
cout « "BearTwo days:  " « BearTwo.GetDaysAlive() « endl;

fin.read((char*) &BearTwo, sizeof BearTwo);

cout « "BearTwo weight:  " « BearTwo.GetWeight() « endl;
cout « "BearTwo days:  " « BearTwo.GetDaysAl ive() « endl;
fin.close();
return 0;
```

```
И^ШИj>  Please enter the file name: Animals
        BearTwo weight:  1
        BearTwo days: 1
        BearTwo weight:  50
        BearTwo days: 100
```

*IIIJIIIIII^* U linijama 3-18 deklarisana je klasa `Animal`. U linijama 22-32 kreirana je datoteka i otvorena za izlaz u binarnom modu. Kreirana je, u liniji 34, životinja čija je težina 50 i koja je stara 100 dana.

Datoteka je zatvorena u liniji 37 i ponovo otvorena za čitanje u binarnom modu u liniji 39. Druga životinja je kreirana u liniji 46, sa težinom jedan i starošću samo jedan dan. Podaci iz datoteke su učitani u novi `Animal` objekat u liniji 51, pri čemu su obrisani postojeći podaci i zamenjeni podacima iz datoteke.

Obrada komandne linije

Mnogi operativni sistemi, kao što su DOS i UNIX, omogućavaju konsniku da prosledi parametre svojim programima, pri njihovom startovanju. Ovi parametri se nazivaju opcije komandne linije i obično su odvojeni praznim mestima. Na primer:

```
SomeProgram Paraml Param2 Parar3
```


Ovi parametri se ne prosleduju direktno u `main()`. Umesto toga, `main()` funkcija svakog programa dobija dva parametra. Prvi je celobrojni i predstavlja ukupan broj argumenata u komandnoj liniji. U ovo je uračunato i samo ime programa, tako da svaki program ima najmanje jedan parametar. Prethodno prikazana komandna linija ima četiri parametra (ime `SomeProgram`, plus tri parametra čine ukupno četiri arguments komandne linije).

Drugi parametar koji se prosleduje main-u je niz pointera na karakter string. S obzirom da je ime niza konstantni pointer na prvi element niza, možete deklarisati ovaj argument da bude pointer na pointer na char, pointer na niz `char`-ova, ili niz nizova `char`-ova.

Uobičajeno je da se prvi argument naziva argc (argument count - broj argumenata), ali ga možete zvati kako god želite. Drugi argument se, obično, naziva argv (argument vector - vektor na argumente), ali i ovo je samo konvencija.

Takođe je uobičajeno da proverite argc, kako biste se uverili da ste dobili očekivani broj argumenata, a da argv koristite kako biste pristupili samom stringu. Primetićete da su `argv [0]`, kao ime programa, i `argv [1]`, kao prvi parametar programa, predstavljeni kao string. Ako Vaš program uzima dva broja kao argumente, potrebno je da izvršite translaciju tih brojeva u stringove. U Danu 21 videćete kako se koriste standardne bibliotečke konverzije. U listingu 16.19 ilustrovano je kako se koriste argumenti iz komandne linije.

Listing 16.19: Koriscenje argumenata komandne linije.

```
linclude <iostream.h>
int main(int argc,  char **argv)
{
   cout « "Received " « argc « " arguments...\n";
   for (int i=0; i<argc; i++)
      cout « "argument " « i « ": " « argv[i]  « endl;
return 0;
}
```

```
TestProgram  Teach Yourself C++ In 21 Days
Received 7 arguments...
argumnet 0:  TestProgram.exe
argument 1:  Teach
argument 2:  Yourself
argument 3:  C++
argument 4:  In
argument 5:  21
argument 6:  Days
```

*IIIIIEJ^** Funkcija main() deklariše dva argumenta: argc je integer, koji sadrži "~ broj argumenata iz komandne linije, dok je argv pointer na niz stringova. Svaki string u nizu na koji je ukazano sa argv je argument komandne linije. Primetićete da argv, jednostavno, može biti deklarisano kao char `*argv` [], ili `char`

argv [] []. Ovo je pitanje stila programiranja, kako ćete deklarisati argv; kao što je u ovom programu deklarisano kao pointer na pointer, niz ofset-a koji su korišćeni za pristup individualnom stringu.

U liniji 4 argc je iskorišćeno, da bi se prikazao broj argumenata komandne linije: ukupno sedam, računajući i samo ime programa.

U linijama 5 i 6 prikazan je svaki argument komandne linije, prosledivanjem stringova terminiranih sa NULL funkciji cout, uz pomoć indeksiranja u niz stringova.

Najčešće koriscenje argumenata komandne linije je ilustrovano unifikacijom listinga 16.18, tako da preuzima ime datoteke kao argument komandne linije. Ovaj listing ne uključuje deklaraciju klasa koje su nepromenjene.

Listing 16.20: Koriscenje argumenata komandne linije.

```
1:    linclude <fstream.h>
2:    int main(int argc, char *argv[])    // vraća 1 za grešku
3:    {
4:       if (argc != 2)
5:       {
6:          cout « "Usage: " « argv[0] « " <filename>" « endl;
7:          return(1);
8:       }
9:
10:      ofstream fout(argv[l],ios::binary);
11:      if (!fout)
12:      {
13:         cout « "Unable to open " « argv[l] « " for writing.\n";
14:         return(1);
15:      }
16:
17:      Animal Bear(50,100);
18:      fout.write((char*) &Bear,sizeof Bear);
19:
20:      fout.close();
21:
22:      ifstream fin(argv[l],ios::binary);
23:      if (!fin)
24:      {
25:         cout « "Unable to open " « argv[l] « " for reading.\n";
26:         return(l);
27:      }
28:
29:      Animal BearTwo(l,1);
30:
31:      cout « "BearTwo weight: " « BearTwo.GetWeight() « endl;
32:      cout « "BearTwo days: " « BearTwo.GetDaysAlive() « endl;
33:
34:      fin.read((char*) &BearTwo, sizeof BearTwo);
35:
```



```
cout « "BearTwo weight:   « BearTwo.GetWeightO  « endl;
cout « "BearTwo days: "  < BearTwo.GetDaysAlive0 « endl;
fin.close();
return 0;
```

```
BearTwo weight: 1
BearTwo days: 1
BearTwo weight: 50
BearTwo days: 100
```

Deklaracija klase Animal je ista kao u listingu 16.18, pa je izostavljena iz ovog primera. Ovoga puta, umesto da pitamo korisnika za ime datoteke, koristili smo argumente iz komandne linije. U liniji 2 deklarisana je main(), koja uzima dva parametra: ukupan broj argumenata komandne linije i pointer na stream niz argumenata komandne linije.

U liniji 4-8 program proverava da li je primio odgovarajući broj argumenata (tačno dva). Ako korisnik nije dostavio ime datoteke, prikazaće se poruka o grešci:

```
Usage TestProgram <filename>
```

Zatim će program završiti sa radom. Primetićete da korišćenjem argv [0], umesto "zakucavanja" imena programa, možete izvršiti kompilaciju ovog programa pod bilo kojim imenom i korišćenjem na ovaj način sve će funkcionisati automatski.

U liniji 10 program pokušava da otvori datoteku, pod imenom koje mu je dostavljeno za binarni izlaz. Ne postoji razlog da se kopira ime datoteke u lokalni privremeni bafer. Ono se može koristiti direktnim pristupom argv [1].

Ova tehnika je ponovljena u liniji 22, kada je ista datoteka ponovo otvorena za ulaz i iskorišćena je naredbama za obradu grešaka, kada nismo bili u mogućnosti da otvorimo datoteku u linijama 13 ili 25.

Rezime

Danas smo upoznali strimove, kao i globale objekte cout i cin. Cilj objekata istream i ostream je da izvrše enkapsulaciju rada sa drajverima uređaja i baferovanjem ulaza i izlaza.

Postoje četiri standardna stream objekta, koji se kreiraju u svakom programu: cout, cin, cerr i clog. Svaki od njih može biti redirektovan u većini operativnih sistema. Istream objekat cin se koristi za ulaz i njegova najčešća uloga je za overload operatora za ekstrakciju (»). Ostream objekat cout se koristi za izlaz i njegova najčešća uloga je za overload operatora Insert («).

Svaki od ovih objekata ima nekoliko funkcija cianova, kao što su get() i put(). S obzirom da opšte forme svake od ovih metoda vraćaju referencu na stream objekat, jednostavno je izvršiti konkatenaciju svakog od ovih operatora i funkcija.

.stream objekata može biti promenjeno, korišćenjem manipulatora. Oni mogu da postave karakteristike formatizacije i prikaza i različite druge atribute stream objekta.

U/I operacije nad datotekama mogu biti izvršene, korišćenjem fstream klasa, koje se izvode iz stream klasa dodatno, i da bi obezbedili Insert operatore i operatore za ekstrakciju, ovi objekti, takode, podržavaju read() i write0 za smeštanje i dtanje velikih binarnih objekata.

Pitanja i odgovori

P Kako ćete znati kada da koristite Insert operatore i operatore za ekstrakciju, a kada druge funkcije clanove stream klasa?

0 Uopšteno, govoreći, jednostavnije je koristiti `Insert` operatore i operatore za ekstrakciju i to je poželjnije kada je njihovo ponašanje adekvatno. U onim neobičnim situacijama kada ovi operatori ne završavaju posao, na primer, kada se čita string koji se sastoji od reči, mogu se koristiti druge funkcije.

P Koja je razlika izmedu cerr i clog?

O cerr nije baferisana. Bilo šta što je upisano u cerr se trenutno prikazuje. Ovo je sasvim u redu za greške koje treba upisati na ekran, ali može oduzeti previse od performansi kada se upisuje na disk clog baferiše svoj izlaz I, stoga, može biti mnogo efikasniji.

P Za{to su strimovi kreirani kada printf() funkcioniše sasvim dobro?

O Pri ntf () ne podržava jak sistem tipizacije C+ + i takođe ne podržava korisnički definisane klase.

P Zašto biste ikada koristili putback?

O Kada se jedna operacija za čitanje koristi da bi se odredilo da li je karakter ispravan, druga operacija čitanja (pretpostavimo, iz drugog objekta) zahteva da karakter bude u baferu. Ovo se često koristi kada prosledujete datoteku, na primer, C+ + kompajler može da koristi `putback()`.

P Kada se koristi IgnoreO?

O Uobičajena upotreba je posle `get()`. S obzirom da `get()` ostavlja karakter za terminaciju u baferu, nije neuobičajeno da odmah posle poziva `get()` pozovete `Ignore (1, "\n")`. Ovo se često koristi u prosledivanju.

P Moji prijatelji koriste printf() u svojim C+ + programima. Da li to mogu i ja?

O Naravno, dobićete na jednostavnosti, ali ćete "platiti" žrtvovanjem koje je vezano za sigurnost tipova.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje predenog materijala i vežbe, koje Vam obezbeduju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigumi da razumete odgovore, pre nego što predete na sledeće poglavlje.

Pitanja

1. Sta je operator za Insert i Čemu on služi?
2. Sta je operator za ekstrakciju i čemu on služi?
3. Koje su tri forme `cin.getO` i koja je razlika između njih?
4. Koja je razlika između `cin.read()` i `cin.getlineO`?
5. Koja je podrazumevana širina za prikaz `long integer-a`, kada se koristi Insert operator?
6. Sta vraća `insert` operator?
7. Koje parametre uzima konstruktor za ofstream objekat?
8. Šta radi argument `ios::ate`?

Vezbe

1. Napišite program koji piše na četiri standardna `iostream` objekta: `cin`, `cout`, `cerr` i `clog`.
2. Napišite program koji od korisnika traži da unese svoje ime i prezime i zatim ih prikazuje na ekranu.
3. Prepravite listing 16.9 da radi isto, ali bez korišćenja `putback()`, ili `ignore()`.
4. Napisite program koji uzima ime datoteke kao parametar i otvara datoteku za čitanje. Pročitajte sve karaktere iz datoteke, prikažite samo slova i punktuaciju na ekranu. Ignorišite sve neprintabilne karaktere, a, zatim zatvorite datoteku.
5. Napišite program koji prikazuje njegove argumente iz komandne linije u obrnutom redosledu, a ne prikazuje ime programa.

Dan 17

Pretprocesor

Najveći deo onoga što pišete u Vašim datotekama izvornog koda je C+ +, njih zatim, interpretira kompajler i vraća u Vaš program. Medutim, pre nego što se kompajler startuje, startuje se pretprocesor, koji Vam omogućava izvršavanje uslovne kompilacije. Danas ćete naučiti

- šta je uslovna kompilacija i kako se njome upravlja
- kako se pišu makroi, uz pomoć pretprocesora
- kako se koristi pretprocesor u pronalaženju bagova.

Pretprocesor i kompajler

Svaki put kada startujete Vaš kompajler, prvi sa radom kreće pretprocesor. On traži pretpocesorske instrukcije, od kojih svaka započinje simbolom "taraba" (#). Efekat svaka od ovih instrukcija je izmena teksta u izvornom kodu. Rezultat je novodobijena datoteka izvornog koda, privremena datoteka koju Vi normalno ne vidite, ali možete dati naredbu kompajleru da je sačuva, tako da možete da je ispitate, ako to želite.

Kompajler ne čita Vašu originalnu datoteku izvornog koda; on čita pretprocesorski izlaz i kompajlira tu datoteku. Naredbom #include efekat ove operacije biće odmah vidljiv. Na ovaj način se pretprocesor instruira da pronade datoteku, čije ime sledi #i nel ude naredbu i da je učita u neku pomoćnu datoteku na toj lokaciji. To je isto kao da ste ovu kompletnu datoteku ukucali direktno u izvorni kod i, u trenutku kada ga kompajler potraži, ta uključena datoteka će već tamo i biti.

Pogled na privremenu datoteku

Gotovo svaki kompajler poseduje prekidač, koji možete postaviti bilo u integrisanom razvojnom okruženju (IDE), bilo u komandnoj liniji, čime ćete instruisati kompajler da sačuva privremenu datoteku. Pogledajte uputstvo za svoj kompajler, kako biste pronašli odgovarajuće prekidaČe, koji se mogu postaviti, ako želite da ispitate ovu datoteku.

Koriscenje #define

Komanda `#define` definiše zamenu stringa. Ako napišete

```
#define BIG 512
```

Vi ste instruirali pretkompajler da string BIG zameni stringom 512 gde god ga bude pronašao. Ovde se ne radi o stringu koji je "u duhu" C++. Karakteri 512 su zamenjeni u Vašem izvornom kodu svuda gde je *token* BIG uočen. Token je string karaktera, koji mogu biti upotrebljeni svuda gde string, ili konstanta, ili neki drugi skup cifara može da se upotrebi. Prema tome, ako napišete

```
Idefine BIG 512
int myArray [BIG];
```

Pomoćna datoteka koju je proizveo pretkompajler će izgledati ovako:

```
int myArray [512];
```

Primetićete da je naredba `#define` nestala. Sve naredbe pretkompajlera su uklonjene iz pomoćne datoteke i neće se uopšte pojaviti u finalnom izvornom kodu.

Koriscenje #define za konstante

Jedan od načina korišćenja `#define` je kao zamena za konstante. Medutim, ovo gotovo nikada nije dobra ideja, budući da `#define` vrši samo zamenu stringa, bez bilo kakve provere tipa. Kao što je već objašnjeno u odeljku o konstantama, postoji sijaset prednosti za korišćenje ključne reči `const`, umesto `#define`.

Koriscenje #define za testiranje

Drugi način korišćenja `#define` je jednostavno deklarisanje da je odrecteni karakter string definisan. Stoga ćete napisati

```
#define BIG
```

Kasnije ćete mod da istestirate da li je `BIG` bio definisan `i` da, u skladu sa tim, `ipre`-duzmete akciju. Naredbe pretkompajlera za testiranje da li je string (bio) definisan su `#ifdef` i `#ifndef`. Iza obe mora da sledi komanda `#endif`, pre završetka bloka (pre sledeće zatvorene zagrade).

`#ifdef` će dati TRUE, ako je string koji se testira već definisan. Stoga, Vi možete napisati

```
lifdef DEBUG
cout « "Debug defined";
#endif
```

Kada pretkompajler pročita `#ifdef`, on će proveriti tabelu koju je napravio, kako bi video da li ste definisali DEBUG. Ako je tako, `#ifdef` će dati TRUE i sve što postoji do sledećeg `#else`, ili `#endif` biće upisano u pomoćnu datoteku za kompajliranje. Ako proizvede FALSE, izmedu `#ifdef DEBUG` i `#endif` ništa neće biti upisano u pomoćnu datoteku; biće tako kao da se nikada nije ni nalazilo na prvom mestu u izvornom kodu.

Primetićete da je `#ifndef` logički suprotna `#ifdef`; `#ifndef` će proizvesti TRUE, ukoliko string, do tog trenutka, nije definisan u datoteci.

Koriscenje #else komande pretkompajlera

Kao što ste mogli da pretpostavite, termin `#else` se može umetnuti ili izmedu `#ifdef`, ili (između) `#ifndef` i naredbe `#endif`. U listingu 17.1 je prikazano kako se ovi termini upotrebljavaju.

Listing 17.1: Koriscenje #define.

```
1:    #define DemoVersion
2:    #define DOS_VERSION 5
3:    linclude <iostream.h>
4:

      int main()


      cout « "Checking on the definitions of DemoVersion, DOS_VERSION and
      ->WINDOWS_VERSION...\n";

      lifdef DemoVersion
         cout « "DemoVersion defined.\n";
      lelse
         cout « "DemoVersion not defined.\n";
      #endif

      lifndef DOS_VERSION
         cout « "DOS_VERSION not defined!\n";
      #else
         cout « "DOS_VERSION defined as: " « DOS_VERSION « endl;
      #endif

      lifdef WINDOWS_VERSION
         cout « "WINDOWS_VERSION defined!\n";
```

nastavlja se

Listing 17.1: Koriscenje #def i ne. *nastavak*

```
25:    #else
26:       cout « "WINDOWS_VERSION was not defined.\n";
27:    #endif
28:
29:       cout « "Done.\n";
30:     return 0;
31:   }
```

```
Checking on the definitions of DemoVersion, DOS_VERSION and
WINDOWS_VERSION...\n";
DemoVersion defined.
DOSJERSION defined as: 5
WINDOWSJERSION was not defined.

Done.
```

U 1 1 2 definisane su DemoVersion i DOS_VERSION, sa DOS_VERSION koja je definisana stringom 5. U liniji 11 testirana je definicija DemoVersion i pošto je DemoVersion definisana (iako bez vrednosti), test je tačan i string je prikazan u u liniji 12.

U liniji 17 se testira da li je DOS_VERSION definisana, ili ne. Postoje DOS_VERSION definisana, test ne prolazi i izvršavanje se nastavlja, počev od linije 20. Ovde je string 5 zamenjen za reč DOS_VERSION; kompajler ovo vidi kao:

```
cout « "DOS_VERSION defined as: " « 5 « endl;
```

Primetićete da prva reč DOS_VERSION nije zamenjena, zato ito se nalazi pod navodnicima. Medutim, druga DOS_VERSION je zamenjena i zato kompajler vidi 5, kao da ste tamo ukucali 5.

Konačno, u liniji 23 program radi test za WINDOWS_VERSION. Pošto niste definisali WINDOWSJ/ERSION, test neće proći i u liniji 24 će se prikazati poruka.

Uključivanje i isključivanje zaštite

Vi ćete biti u situaciji da kreirate projekte sa mnogo razlidtih datoteka. Verovatno ćete organizovati svoje direktorijume tako da svaka klasa ima sopstvenu datoteku zaglavlja (HPP), sa deklaracijom klase i sopstvenom datotekom za implementaciju (CPP) i izvornim kodom za metode klase.

Vaša main() funkcija će biti smeštena u sopstvenu CPP datoteku i sve CPP datoteke će biti kompajlirane u OBJ datoteke, koje će, zatim, uz pomoć linkera, biti povezane u poseban program.

Pošto će Vaši programi koristiti metode iz različitih klasa, u svaku datoteku biće uključeno vise datoteka zaglavlja. Takode, datotekama zaglavlja je često potrebno da uključuju jedna drugu. Na primer, datoteka zaglavlja za deklaraciju izvedene klase mora da ukljud datoteku zaglavlja svoje bazne klase.

Zamislite da je klasa Animal deklarisana u datoteci ANIMAL.HPP. Klasa Dog (koja se izvodi iz Animal) mora da uključi datoteku ANIMAL.HPP u DOG.HPP, ili Dog neće moći da se izvede iz Animal a. Zaglavlje Cat takođe uključuje ANIMAL.HPP zbog istog razloga.

Ako kreirate metod koji koristi i Cat i Dog, pretiće Vam opasnost da dvaput uključite ANIMAL.HPP. Ovo će generisati grešku u kompajliranju, jer nije dozvoljeno da se clasa (Animal) deklariše dvaput, čak iako su deklaracije identične. Ovaj problem ćete rešiti uključivanjem zaštita. Na vrhu Vaše ANIMAL datoteke zaglavlja ispisaćete sledeće linije:

```
#ifndef ANIMAL_HPP
Idefine ANIMAL_HPP
                    // eela datoteka dolazi ovde
#endif
```

što će reći da sada definišite termin ANIMAL_HPP, ukoliko to već niste uradili. Izmedu naredbe #define i #endif smešten je kompletan sadržaj datoteke.

Prvi put kada Vaš program bude ukljudo ovu datoteku, on će pročitati prvu liniju i test će proizvesti TRUE; to znači, da Vi još uvek niste definisali ANIMAL_HPP. Prema tome, krenite i definišite je, a, zatim, uključite i kompletnu datoteku.

Kada Va§ program drugi put bude ukljudo datoteku ANIMAL.HPP, on će prodtati prvu liniju i proizvesti FALSE; ANIMAL.HPP je (već) definisana. Zato on preskače u sledeću #else (koja ne postoji), ili sledeću #endif (na kraju datoteke). Stoga, on preskače dtav sadržaj datoteke, tako da klasa neće biti dvaput deklarisana.

Postojed (aktuelni) naziv defmisanog simbola (ANIMAL_HPP) nije važan, mada je uobičajeno da se za ime datoteke koriste velika slova sa tačkom (.) zamenjenom u (_).

Definisanje u komandnoj liniji

Gotovo svi C 4-4- kompajleri će Vam omogudti koriscenje #define vrednosti bilo iz komandne linije, bilo iz integrisanog razvojnog okruženja. Stoga, imate mogućnost da izostavite linije 1 i 2 iz Listinga 17.1 i da definišete DemoVersion i BetaTestVersion iz komandne linije za neke kompilacije, ali ne i za sve ostale.

Nije ništa neobično, ako stavite specijalan kod za debagovanje, koji će biti okružen #ifdef DEBUG i #endif. Ovo će omogudti da svi kodovi za debagovanje budu Iako uklonjeni iz izvornog koda kada budete vršili kompajliranje finalne verzije; jednostavno, nemojte definisati termin DEBUG.

Poništavanje definicija

Ako imate ime koje ste definisali i želite da ga iskljudte iz Vašeg koda, upotrebite #undef. Ova komanda radi kao "antipod" #define. Listing 17.2 ilustruje njeno koriscenje.

Listing 17.2: Koriscenje lundef.

```
1:  #define DemoVersion
2:  #define DOSJERSION 5
3:  linclude <iostream.h>
4:
5:
6:  int main()
7:  {
8:
9:  cout « "Checking on the definitions of DemoVersion, D0S_VERSI0N and
    -WINDOWSJ/ERSION...\n";
10:
11: lifdef DemoVersion
12:    cout « "DemoVersion defined.\n";
13: lelse
14:    cout « "DemoVersion not defined.\n";
15: lendif
16:
17: lifndefDOSJERSION
18:    cout « "DOSJERSION not defined!\n";
19: lelse
20:    cout « "DOSJERSION defined as: " « DOSJERSION « endl;
21: lendif
22:
23: lifdefWINDOWSJERSION
24:    cout « "WINDOWSJERSION defined!\n";
25: lelse
26:    cout « "WINDOWSJERSION was not defined.\n";
27: lendif
28:
29: lundef DOSJERSION
30:
31:  lifdef DemoVersion
32:    cout « "DemoVersion defined.\n";
33:  #else
34:    cout « "DemoVersion not defined.\n";
35:  lendif
36:
37:  #ifndef DOSJERSION
38:    cout « "DOSJERSION not defined!\n";
39:  lelse
40:    cout « "DOSJERSION defined as: " « DOSJERSION « endl;
41:  lendif
42:
43:  lifdefWINDOWSJERSION
44:    cout « "WINDOWSJERSION defined!\n";
45:  lelse
46:    cout « "WINDOWS_VERSION was not defined.\n";
47:  lendif
```



```
cout « "DoneAn"
return 0;
}
```

```
Checking on the definitions of DemoVersion, DOSJERSION and
WINDOWSJERSION...\n";
DemoVersion defined.
DOSJERSION defined as: 5
WINDOWSJERSION was not defined.
DemoVersion defined.
DOSJERSION not defined!
WINDOWSJERSION was not defined.
Done.
```

Listing 17.2 je isti kao i listing 17.1, osim u liniji 29, kada je pozvana #undef DOSJERSION. Na ovaj način se uklanja definicija termina DOSJERSION, bez menjanja ostalih definisanih teimina (u ovom slučaju, DemoVersion). Preostali deo listinga samo ponavlja izlaze. Testovi za DemoVersion i WINDOWSJERSION se ponašaju kao što su se ponašali i ranije, ali test za DOSJERSION sada proizvodi TRUE. U ovom drugom slučaju, D0S_VERSI0N ne postoji kao definisan termin.

Uslovna kompilacija

Kombinovanjem `#define`, ili definicija komande linije sa `#ifdef`, `#else` i `#ifndef`, bićete u mogućnosti da napište program za kompajliranje razlicitih kodova, u zavisnosti od toga šta je već `#defined`. Na ovaj način se može kreirati skup izvornog koda, čije će se kompajliranje (iz)vršiti na dvema različitim platformama, kao što su DOS i Windows.

Drugi uobičajeni slučaj upotrebe ove tehnike je kod uslovne kompilacije u nekim kodovima, koja zavisi od toga da li je `debug` definisan, što ćete moći da vidite za nekoliko trenutaka.

<J|- **PAzm** jj> Koristite uslovnu kompilaciju kada imate potrebu da kreirate vise od jedne verzije svog koda u isto vreme.

Nemojte dozvoliti da Vaši uslovi postanu suviše složeni za održavanje.

Koristite `#undef` kad god je to moguće, kako biste u Vašem kodu izbegli "zalutale" definicije.

Koristite uključivanje zaštita!

Makro funkcije

Naredba `ldefine` se, takođe, može koristiti za kreiranje makro funkcija. Makro funkcija je simbol, kreiran uz pomoć `#define`, koji traži argument, na vrlo sličan način kako to radi funkcija. Pretprocesor će zameniti string za zamenu bilo kojim datim argumentom. Na primer, možete definisati makro TWICE kao

```
Idefine TWICE(x) ( (x) * 2 )
```

1, zatim, u svoj kod uneti

```
TWICE(4)
```

Kompletan string TWICE će biti uklonjen i vrednost 8 će biti supstituisana (zamenjena). Kada pretkompajler bude naišao na 4, on će zameniti ((4) * 2) i proizvesti 4 * 2, ili 8.

Makro može da ima vise od jednog parametra i svaki od njih može se ponovo upotrebiti u tekstu u kome se vrši zamena. Dva uobičajena makroa su MAX i MIN:

```
Idefine MAX(x.y) ( (x) > (y) ? (x) : (y) )
Idefine MIN(x.y) ( (x) < (y) ? (x) : (y) )
```

Primetićete da se u definiciji makro funkcije otvorene zagrade za listu parametara moraju nalaziti odmah iza imena makroa, bez praznog prostora. Pretprocesor Vam neće "progledati kroz prste" za prazna mesta, kao što će to učiniti kompajler.

Ako ste napisali

```
Idefine MAX (x.y) ( (x) > (y) ? (x) : (y) )
```

a, zatim, pokušali da ovako upotrebite MAX

```
int x = 5, y = 7, z;
z = MAX(x.y);
```

privremeni kod će izgledati ovako

```
int x = 5, y = 7, z;
z • (x,y) ( (x) > (y) ? (x) : (y) ) (x.y)
```

Biće izvršena jednostavna zamena teksta, umesto pozivanja makro funkcije. Stoga će token MAX biti zamenjen sa (x,y) ((x) > (y) ? (x): (y)), a zatim će to biti praćeno sa (x,y), koje sledi MAX.

Medutim, uklanjanjem praznog prostora izmedu MAX i (x, y), pomoćni kod će postati

```
int x - 5, y = 7 , z
z = 7;
```

Čemu služe sve te zagrade?

Možda ćete se zapitati čemu toliko zagrada u većini makroa, koji su do sada prikazani. Pretprocesor ne zahteva da se argumenti u stringu za supstituciju (zamenu) stavljaju u zagrade, ali one pomažu da se izbegnu neželjeni sporedni efekti prilikom prosledivanja komplikovanih vrednosti u makro. Na primer, ako definišete MAX kao

```
Idefine MAX(x.y) x > y ? x : y
```


I r o S f l (mU), r dnoSti_5/7, makro Će raditi kao št0 ie l nameravao. Ali, ako mu prosfedite komphkovarnji izraz, dobićete rezultate koje niste planirali da dobijete, kao štoje prikazano u listingu 17.3.

Listing 17.3: Koriscenje zagrada u makroima.

```
1:    // Listing 17.3 Makro razvoj
2:    linclude <iostream.h>
3:
4:    Idefine CUBE(a) ( (a) * (a) * (a) )
5:    Idefine THREE(a) a * a ', a
6:
7:    int main()
8:    {
9:       long x = 5;
10:      long y = CUBE(x);
11:      long z = THREE(x);
12:
13:        cout « "y: " « y « endl;
14:      cout « "z: " « z « endl;
15:
16:      long a = 5, b = 7;
17:      y = CUBE(a+b);
18:      z = THREE(a+b);
19:
20:      cout « "y: " « y « endl;
21:      cout « "z: " « z « endl;
22:   return 0;
23:   }
```

```
W y: 125
   z: 125
      1728
      82
```

^'.'Mu/:^ U liniji 4 definisan je makro CUBE sa argumentom x koji, se stavlja u zagrade svaki put kada se upotrebi. U liniji 5 definisan je makro THREE, bez zagrada.

Prilikom prvog korišćenja makroa, vrednost 5 je data kao parametar i oba makroa dobro rade. CUBE (5) se proširuje u (5) * (5) * (5), što daje 125 i THREE (5) se proširuje u 5 * 5* 5, što daje 125.

Prilikom drugog korišćenja, u linijama 16-18 parametar je 5 + 7. U ovom slučaju, CUBE (5 + 7) daje (5+7) * (5+7) * (5+7))

što daje

```
(12) * (12) * (12) )
```

sto, zatim, daje 1.728. Medutim, THREE (5 4- 7) daje

```
5 + 7 * 5 + 7 * 5 + 7
```

Pošto množenje ima viši nivo od sabiranja, ovo prerasta u

```
5 + (7 * 5) + (7 * 5) + 7
```

što daje

```
5 + (35) + (35) + 7
```

što na kraju daje 82.

Makroi protiv funkcija i templejta

C++ makroe "pritiskaju" četiri problema. Prvi je što mogu biti zbunjujući, ako postanu veliki, pošto makroi moraju biti definisani u jednoj liniji. Ovu liniju možete proširiti upotrebom obrnute kose crte (backslash) (\), ali veliki makroi brzo postaju teški za održavanje.

Drugi problem je što se makroi sire unutar linije svaki put kada se upotrebe, što će reći da, ako se makro upotrebi 12 puta, ta izmena će se u Vašem programu pre pojaviti 12 puta, nego što će se poziv neke funkcije pojaviti samo jednom. Sa druge strane, one su, obično, mnogo brže od poziva funkcija, s obzirom da je izbegnuto preobimno koriscenje resursa, koje bi funkcije zahtevale.

Činjenica da se C++ makroi proširuju unutar linije vodi u treći problem-makro se vise ne pojavljuje unutar izvornog koda koji koristi kompajler i stoga je nedostupan većini debagera. Ovim je debagovanje makroa postalo vrlo komplikovano.

Međutim, četvrti problem je i najveći: makroi ne održavaju sigurnost tipova. S obzirom da je uobičajeno da apsolutno bilo koji argument može biti upotrebljen u radu sa makroima, ovo u velikoj meri slabi tipiziranje u C++ i predstavlja pravo prokletstvo za C++ programere. Medutim, postoje načini za prevazilaženje ovih problema, koji će biti prikazani u Danu 19, "Templejti."

Inline funkcije

Često je moguće deklarisati inline funkciju, umesto makroa. Na primer, listing 17.4 kreira CUBE funkciju, koja radi isto što i CUBE makro iz listinga 17.3, ali ovoga puta tipovi podataka su očuvani.

Listing 17.4: Koriscenje inline funkcija, umesto makroa

```
linclude <iostream.h>

inline unsigned long Square(unsigned long a)
inline unsigned long Cube(unsigned long a)
    { return a * a * a; }
int  main()
```



```
8:      unsigned long x=l ;
9:      for (;;)
10:     {
11:        cout « "Enter a number (0 to quit):
12:        cin » x;
13:        if (x == 0)
14:           break;
15:        cout « "You entered: " « x;
16:        cout « ".  Square(" « x « "): ";
17:        cout  « Square(x);
18:        cout« ". Cube(" _« x « "): ";
L9:        cout « Cube(x) « "." « endl;

      return 0;
```

```
Enter a number (0 to quit): 1
You entered: 1.  Square(l): 1. Cube(l): 1.
Enter a number (0 to quit): 2
You entered: 2.  Square(2): 4. Cube(2): 8.
Enter a number (0 to quit): 3
You entered: 3.  Square(3): 9. Cube(3): 27.
Enter a number (0 to quit): 4
You entered: 4.  Square(4): 16. Cube(4): 64.
Enter a number (0 to quit): 5
You entered: 5.  Square(5): 25. Cube(5): 125.
Enter a number (0 to quit): 6
You entered: 6.  Square(6): 36. Cube(6): 216.
Enter a number (0 to quit): 0
```

lu.UM.]^K U linijama 3 i 4 definisane su dve inline funkcije: Square () i Cube(). Svaka je definisana kao inline funkcija, pa će, kao i makro funkcije, biti proširene u jednom prostoru za svaki poziv, neće biti prekobrojnog pozivanja funkcija. Da Vas podsetimo, inline širenje znači da će sadržaj funkcije biti smešten u kodu, bez obzira odakle je funkcija pozvana (na primer, u liniji 16). U liniji 16 pozvana je funkcija Square, kao i funkcija Cube. Pošto su ovo inline funkcije, desiće se isto kao da su ove linije napisane na sledeći način:

```
16:        cout « ".  Square(" « x « "): "  « x * x « ". Cube(" « x « "): " «
x * x * x «
"." « endl;
```

Manipulacija stringovima

Pretprocesor obezbeduje dva specijalna operatora za manipulisanje stringovima u makroima. Operator za str.ngovanje (#) zamenjuje string pod navodnicima b.lo čime sto sledi operator za stringovanje. Operator za konkatenaciju povezuje dva stringa u jeoan.

Stringovanje

Operator za stringovanje smešta u navodnike bilo koji karakter koji sledi operator, sve do sledećeg praznog prostora. Stoga, ako napišete

```
#define WRITESTRING(x) cout « #x
```

i, zatim, pozovete

```
WRITESTRIN6(This is a string);
```

kompajler će ovo pretvoriti u

```
cout « "This is a string";
```

Primetićete daje string `"This is a string"` smešten u navodnike, kako je `cout` zahtevao.

Konkatenacija

Operator za konkatenaciju Vam omogućava da povežete vise izraza u jednu reč. Nova reč je, u stvari, token, koji se može koristiti kao ime klase, ime promenljive, i kao pozicija u nizu, ili u bilo kojoj drugoj formi (situaciji), gde se serija slova može pojaviti.

Pretpostavimo, na trenutak, da imate pet funkcija `fOnePrint, fTwoPrint, fThreePrint, fFourPrint` i `fFivePrint`. Takode, možete deklarisati:

```
#define fPRINT(x) f ll x ## Print
```

i zatim koristiti `f PRINT (Two)`, da biste generisali `fTwoPrint`, i `f PRINT (Three)` za generisanje `fThreePrint-a`

U zaključku Nedelje 2 razvijena je klasa `PartsList`. Ova lista može da podržava samo objekte tipa `List`. Pretpostavimo da će ovo funkcionisati, a Vi biste želeli da možete da napravite liste životinja, kola, kompjutera, itd.

Jedan pristup bi bio da kreirate `Animal List, CarList, ComputerList` i tako dalje, kopirajući postojeće delove programa. Ovo bi ubrzo postaoVaša noćna mora, s obzirom da bi bilo kakva izmena u jednoj prouzrokovala izmene u svim ostalim listama.

Alternativa je koriscenje makroa i operatora za konkatenaciju. Na primer, mogli biste da defmišete

```
Idefine Listof(Type)  class Type##List \
{ \
public: \
Type##List(){} \
private:          \
int itsLength; \
};
```


Ova primer je veoma uprošćen, `ali` ideja je da se smeste svi neophodni metodi i podaci. Kada budete spremni da kreirate `Animal List`, napisaćete

```
Listof(Animal)
```

i to će biti pretvoreno u deklaraciju klase `Animal List`. Postoji nekoliko problema u ovom pristupu, a koji će biti prodiskutovani u Danu 19, "Templejti".

Predefinisani makroi

Mnogi kompajleri predefinišu veliki broj korisnih makroa, kao što su _DATE_, _TIME_, _LINE_, _FILE_. Svako od ovih imena je ograničeno dvema donjim crtama, da bi se izbegli konflikti sa sličnim imenima koje ste koristili u svojim programima.

Kada pretkompajler primeti jedan od ovih makroa, on će izvršiti odgovarajuću zamenu. Za _DATE_ biće izvršena zamena tekućim datumom. _TIME_ će biti zamenjeno tekućim vremenom, dok će _LINE_ i _FI LE_ biti zamenjeni u izvornom programu brojem linije, odnosno imenom datoteke, respektivno. Primetćete da će ove zamene biti izvršene u trenutku kompilacije, a ne u trenutku izvršenja. Ako od programa zatražite da prikaže _DATE_, Vi nećete dobiti tekući datum, nego datum kada je program poslednji put kompajliran. Na ovaj način, makroi postaju vrlo korisni za debagovanje programa.

assert()

Većina kompajlera nudi assert() makro. Makro assert() vraća vrednost TRUE, ako njegov parametar proizvodi TRUE, i izvodi neke "akcije", ako on proizvodi FALSE. Većina kompajlera će prekinuti program, ako assert() ne uspe; drugi će, pak, proslediti izuzetak (pogledajte Dan 20, "Izuzeci i obrada grešaka").

Najsnažnija osobina makroa assert () je da ga kompajler neće ugraditi u kod, ako nije definisan DEBUG. Ovo je velika pomoć tokom razvoja i kada se konačni proizvod isporučuje ne dolazi do gubitka performansi, ili uvećanja izvršne verzije programa.

Umesto da zavisite od assert (), koji Vam je obezbedio kompajler, imate pravo da napišete sopstveni makro assert(). U listingu 17.5 prikazan je jednostavan assert (), kao i njegovo koriscenje.

Listing 17.5: Jednostavan assertQ **makro.**

```
// Listing 17.5 ASSERT
Idefine DEBUG
linclude <iostream.h>

#ifndef DEBUG
   Idefine ASSERT(x)
lelse
```

nastavlja se

Listing 17.5: Jednostavqn ossertQ makro. *nastavak*

```
8:      Idefine ASSERT(x) \
9:              if (! (x)) \
10:             { \
11:                cout « "ERROR!! Assert " « Ix « " failed\n"
12:                cout « " on line M «____LINE____« "\n"; \
13:                cout « " in file " «____FILE___« "\n";  \
14:             }
15:  lendif
16:
17:
18:  int  main()
19:  {
20:     int x = 5;
21:     cout « "First assert: \n";
22:     ASSERT(x==5);
23:     cout « "\nSecond assert: \n";
24:     ASSERT(x != 5);
25      cout « M\nDone.\n";
26     return 0;
27    }
```

```
First assert:


Second assert:
ERROR!! Assert x !=5 failed
on line 24
in file testI704.cpp
Done.
```

U liniji 2 definisan je izraz DEBUG-uobičajeno je da se ovo izvrši iz komandne linije u trenutku kompilacije. Stoga ga možete uključiti, ili isključiti.

U linijama 8-14 definisan je makro `assert()`. Ovo se, obično, radi u datotekama zaglavlja i zaglavlje (ASSERT.HPP) će biti uključeno u sve Vase datoteke za implementaciju.

U liniji 5 testiran je izraz DEBUG. Ako on nije definisan, `assert` () je definisan tako da ne kreira nikakav kod. Ako je DEBUG definisan, primeniće se funkcionalnost, koja je definisana u linijama 8-14.

Sam `assert` () je jedna dugačka naredba, podeljena unutar sedam linija koda. U liniji 9 testiran je prosledeni parametar i kada on proizvede FALSE pozivaju se naredbe u linijama 11-13, koje prikazuju poruku o grešci. Ako je prosledena vrednost prouzrokovala TRUE, neće biti sprovedena nikakva akcija.

Debagovanje sa assert()

Kada pišete Vaš program, često ćete biti u situaciji da ste u dubini duše sigurni da je nešto tačno: funkcija ima odgovarajuću vrednost, pointer je ispravan i tako dalje. Priroda bagova je takva da, ono za šta Vi mislite da je tačno, pod nekim uslovima nije. Na primer, Vi znate da je neki pointer ispravan, ali Vaš program i dalje "pada". Assert () Vam može pomoći da pronađete ovaj tip bagova, ali samo ako imate naviku da koristite assert () u svom kodu. Svaki put kada dodelite, ili prosledite pointer, kao parametar, ili vraćenu vrednost funkcije, uverite se da je pointer ispravan. Svaki put kada Vaš kod zavisi od odredene vrednosti, koja se nalazi u promenljivoj, proverite sa assert () da li je to tačno.

Ne postoje lose strane učestalog korišćenja `assert()`; on će biti uklonjen iz koda kada isključite debagovanje. On, takođe, omogućava dobru internu dokumentaciju, podsećajućii čitača da je tačnan kod u nastavku programa.

assert () protiv izuzetaka

U Danu 20 naučićete kako da radite sa izuzecima, koji će Vam pomoći pri obradi grešaka. Važno je napomenuti da `assert` () nije namenjen za podršku greškama u fazi izvršenja, kao što su neispravni podaci, prekoračenje memorije, nemogućnost otvaranja datoteke i tako dalje. On je namenjen samo za "hvatanje" grešaka u programiranju. Stoga, ako se `assert` () javi, znaćete da imate bag u svom kodu.

Ovo je kritično, s obzirom da, kada isporudte kod kupcima, kopije `assert` () će biti uklonjene. Ne možete zavisiti od `assert()`, kako biste rešili probleme u fazi izvršavanja, budući da `assert()` neće biti prisutan. Uobičajena greška je da se `assert` () koristi za testiranje vraćene vrednosti pri dodeljivanju memorije:

```
Animal *pCat = new Cat;
Assert(pCat);    // loša upotreba Asserta
pCat->SomeFunction();
```

Ovo je klasična greška u programiranju. Svaki put kada programer startuje program, na raspolaganju mu je dovoljno memorije i `assert` () se neće javiti. Na kraju krajeva, programer startuje program sa velikom količinom dodatnog RAM-a, kako bi ubrzao kompajler, debager, itd. Programer posle toga isporučuje program i javni korisnik, koji ima malo memorije, uspeva da dode samo do dela programa i pozivi za new ne uspevaju i vraćaju NULL. Medutim, `assert` () vise nije u kodu; vise ništa ne može indicirati da pointer ukazuje na NULL. U trenutku kada se dostigne naredba `pCat ->SomeFunction()` program će "pući."

Dobijanje NULL-a u trenutku dodeljivanja memorije nije greška u programiranju. Ipak, ovo je jedna specijalna situacija. Vaš program bi morao da nađe mogućnost da se izvuče iz ove situacije. Zapamtite: cela `assert` () naredba će nestati, kada se ukloni DEBUG. Izuzeci su detaljno opisani u Danu 20.

Propratni efekfi

Nije neuobičajeno da se pojave bagovi samo posle uklanjanja kopija assert(). Ovo se, gotovo uvek, dogada usled nenamerne zavisnosti programa od propratnih efekata, koji su ostvareni u assert(), ili u nekom drugom kodu, koji služi isključivo za debagovanje.

Na primer, ako napisete

```
ASSERT (x = 5)
```

kada ste nameravali da proverite da li je x = = 5, kreiraćete izuzetno "nestašan" bag.

Recimo, da ste baš pre ovog assert() pozvali funkciju koja je postavila da je x jednako 0. Ovim assert() mislili ste da proveravate da li je x jednako 5; činjenica je da ste vi, zapravo postavili da je x jednako 5. Test će vratiti TRUE, pošto je x = 5, i ne samo da će postaviti x na 5, nego će i vratiti vrednost 5, a s obzirom da je 5 različito od nule, vratiće vrednost TRUE.

Kada prosledite assert() naredbu, x će stvarno biti 5 (upravo ste ga tako i postavili!). Vaš program će funkcionisati. Kada budete spremni da ga isporučite, Vi ćete isključiti debagovanje. U torn trenutku nestaće sve naredbe assert() i vrednost x vise neće biti postavljana na 5. S obzirom da je pre ovoga vrednost x postavljena na nulu, ona će i ostati nula i Vaš program neće funkcionisati.

Isfrustrirani, vraćate debagovanje i gle: na čaroban način bag će nestati! Ovo je daleko zabavnije posmatrati, nego sa time "živeti", i zato budite vrlo pažljivi sa propratnim efektima nastalim u kodu za debagovanje. Ako primetite bag koji se pojavljuje samo kada je isključeno debagovanje, pažljivo pregledajte Vaš kod za debagovanje "širom otvorenih očiju", ne biste li uočili propratne efekte.

Invarijante klasa

Većina klasa ima neke uslove, koji uvek moraju da budu tačni, čak i kada završite sa funkcijama clanovima klase. Ove *invarijante* su sine qua non Vase klase. Na primer, verovatno je tačno da Vaš CIRCLE objekat nikada ne bi trebalo da ima prečnik dužine 0, ili da Vaš Animal uvek mora da ima godina vise od nula, a manje od 100.

Bilo bi veoma zgodno deklarisati metod InvariantsO, koja vraća TRUE samo ako su svi uslovi zadovoljeni. Vi tada možete postaviti assert (InvariantsO) na početak i na kraj svakog metoda klase. Izuzetak bi mogao da bude kada InvariantsO ne bi očekivao da vrati True pre nego što Vaš konstruktor bude startovan, ili pošto Vaš destruktor završi sa radom. U listingu 17.6 demonstrirano je koriscenje metoda Invariants u krajnje jednostavnoj klasi.

Listing 17.6: Korišćenie InvariantsO •

```
*\
-H
*
```

```
0:    Idefine DEBUG
1:  Idefine SHOWJNVARI ANTS
2:    linclude <iostream.h>
3:    linclude <string.h>
4:
5:    lifndef DEBUG
6:    Idefine ASSERT(x)
7:    Ielse
8:    Idefine ASSERT(x) \
9:                if (I (x)) \
10:                { \
11:                   cout « "ERROR!! Assert " « Ix « " failed\n"; \
12:                   cout « " on line " «____LINE____« "\n"; \
13:                   cout « " in file " «    FILE    « "\n"-   \
14:                }                          ~      -
15:   Iendif
16:
17:
18:   const int FALSE = 0;
19:   const int TRUE = 1;
20:   typedef int B00L;
21:
22:
23:   class String
24:   {
25:      public:
26:        // konstruktori
27:        StringO;
28:        String(const char *const);
29:        String(const String &);
30:        -StringO;
31:
32:        char & operator[](int offset);
33:        char operators(int offset) const;
34:
35:        String & operator (const String &);
36:        int GetLen()const { return itsLen; }
37:        const char * GetStringO const { return itsString; }
38:        B00L InvariantsO const;
39:
40:     private:
41:        String (int);          // privatni konstruktor
42:        char * itsString;
43^        // unsigned short itsLen;
44:        int itsLen;
45:   };
46:
```

nastavljase

nastavak

Listing 17.6: Koriscenje InvariantsO •

```
47    // podrazumevani konstruktor kreira string od 0 bajtova
48    String::String()
49    {
50       itsString = new char[l];
51       itsString[0] = '\0';
52       itsLen=0;
53       ASSERT(InvariantsO);
54
55
56    // privatni (pomoćni) konstruktor, koji se koristi samo od
57    // metoda klase za kreiranje novog stringa
58    // tražene veličine. Puni se sa null.
59    String::String(int len)
60    {
61       itsString = new char[len+l];
62       for (int i = 0; i<=len; i++)
63          itsString[i] = '\0';
64       itsLen=len;
65       ASSERT(InvariantsO);
66
67
68    // Konvertuje niz karaktera u String
69    String::String(const char * const cString)
70    {
71       itsLen = strlen(cString);
72       itsString = new char[itsLen+1];
73       for (int i = 0; i<itsLen; i++)
74          itsString[i] = cString[i];
75       itsString[itsLen]='\0';
76       ASSERT(InvariantsO);
77    }
78
79    // konstruktor kopije
80    String::String (const String & rhs)
81    {
82       itsLen=rhs.GetLen();
83       itsString = new char[itsLen+1];
84       for (int i = 0; i<itsLen;i++)
85          itsString[i] = rhs[i];
86       itsString[itsLen] = '\0';
87       ASSERT(Invariants())s
88
89
90    // destruktor, oslobada alociranu memoriju
91    String::-String ()
92
93       ASSERT(InvariantsO);
94       delete [] itsString;
```



```
      itsLen = 0;


// operator jednako, oslobađa postoje}u memoriju
// a onda kopira string i veličinu
 String& String::operator=(const String & rhs)
 {
    ASSERT(InvariantsO);
    if (this == &rhs)
       return *this;
    delete [] itsString;
    itsLen=rhs.GetLen();
    itsString = new char[itsLen+1];
    for (int i = 0; i<itsLen;i++)
       itsString[i] = rhs[i];
    itsString[itsLen] = '\0';
    ASSERT(InvariantsO);
    return *this;


// nekonstantan operator pomaka, vra}a
// referencu na karakter tako da se on može
//  promeniti!
char & String::operator[](int offset)
{
   ASSERT(InvariantsO);
   if (offset > itsLen)
      return itsString[itsLen-1];
   el se
      return itsString[offset];
   ASSERT(Invariants());


// konstantan operator pomaka koji se koristi
// za konstantne objekte (pogledajte konstruktor kopije!)
char String::operator[](int offset) const
{
   ASSERT(Invariants());
   if (offset > itsLen)
      return itsString[itsLen-1];
   el se
      return itsString[offset];
   ASSERT(InvariantsO);



BOOL String::InvariantsO const
(
lifdef SHOW INVARIANTS
```

nastavlja se

Listing 17.6: Koriifenje InvariantsO.

```
144:       cout « " String OK ";
145:    #endif
146:       return ( (itsLen && itsString) dd
147:         (!itsLen && !itsString) );
148:    }
149:
150:    class Animal
151:    {
152:    public:
153:       Animal():itsAge(l),itsName("John Q. Animal")
154:          (ASSERT(Invari ants());)
155:        Animal (int, const Strings;
156:       -Animal (){}
157:       int GetAgeO {   ASSERT(Invariants()); return itsAge;}
158:       void SetAge(int Age)
159:       {
160:            ASSERT(InvariantsO);
161:           itsAge = Age;
162:            ASSERT(InvariantsO);
163:       }
164:        String&  GetNameO
165:       {
166:             ASSERT(InvariantsO);
167:            return itsName;
168:       }
169:       void  SetName(const  String&  name)
170:             {
171:              ASSERT(InvariantsO);
172:              itsName  -  name;
173:              ASSERT(InvariantsO);
174:       }
175:       BOOL  InvariantsO;
176:    private:
177:       int  itsAge;
178:       String  itsName;
179:    };
180:
181:     Animal::Animal(int age, const String* name):
182:    itsAge(age),
183:    itsName(name)
184:    {
185:        ASSERT(InvariantsO);
186:    }
187:
188:    BOOL   Animal::Invariants()
189:    {
190:    lifdef SHOWJNVARIANTS
191:       cout « " Animal OK ";
192:    lendif
193:       return (itsAge > 0 && itsName.GetLen());
194:
195:
196:    int main()
197:    {
198:       Animal sparky(5,"Sparky");
199:       cout « "\n" « sparky.GetNameO .GetStringO
200:       cout « sparky.GetAgeO « " years old.";
201:       sparky.SetAge(8);
202:       cout « "\n" « sparky.GetNameO .GetStringO
203:       cout « sparky.GetAgeO « " years old.";
204:       return 0;
205:    }
```

```
String OK  String OK  String OK  String OK  String OK
String OK  String OK  Animal OK  String OK  Animal OK
Sparky is  Animal OK 5 years old. Animal OK  Animal OK
Animal OK  Sparky is  Animal OK 8 years old. String OK
```

U linijama 6-16 definisan je makro `assert()`. Ako je definisan DEBUG, ovo će prikazati poruku o grešci, kada `assert()` makro proizvede FALSE.

U liniji 38 je deklarisana funkcija `InvariantsO` String klase, koja je definisana u linijama 141-148. Konstruktor je deklarisan u linijama 48-54, a u liniji 53, pošto su svi objekti konstruisani, pozvan je `InvariantsO`, kako bi potvrdio ispravnost konstrukcije.

Ova šema je ponovljena i za druge konstruktore, a destruktor je pozvao `InvariantsO` tek pošto je uništio objekat. Preostale funkcije klase pozivaju `InvariantsO` pre preuzimanja bilo kakve akcije, kao i po završetku tih akcija.

Ovim su afirmisani i potvrdeni osnovni principi C + + . Funkcije članovi, koji nisu konstruktori, ili destruktori, trebalo bi da rade sa ispravnim objektima i te objekte bi trebalo da ostave u ispravnom stanju.

U liniji 175 klasa `Animal` deklariše sopstveni `InvariantsO` metod koji je implementiran u linijama 188-194. Primetite u linijama 154, 157, 160 i 162 da inline funkcije mogu da pozovu `InvariantsO` metod.

Štampanje privremenih vrednosti

Uz tvrdnju da je nešto tačno, na osnovu korišćenja `assert()` makroa, možete poželeti da prikažete tekuće vrednosti pointera, promenljivih i stringova. Ovo može biti veoma korisno za proveru Vaših pretpostavki o radu Vašeg programa i kod lociranja bagova u petlji. U listingu 17.7 prikazana je ova ideja.

Listing 17.7: Prikazivanje vrednosti u DEBUG modu

```
1   // Listing 17.7 - Štampanje vrednosti u OEBUG režimu.
2   linclude <iostream.h>
3   Idefine DEBUG
4
5   lifndef DEBUG
6   Idefine PRINT(x)
7   lelse
8   Idefine PRINT(x) \
9      cout « Ix « ":\t" « x « endl;
10  lendif
11
12  enum BOOL { FALSE, TRUE } ;
13
14  int main()
15  {
16     intx=5;
17     long y = 738981;
18     PRINT(x);
19     for (int i = 0; i < x; i++)
20     {
21        PRINT(i);
22
23
24     PRINT (y);
25     PRINT("Hi.");
26     int *px = &x;
27     PRINT(px);
28     PRINT (*px);
29   return 0;
30
```

```
x:      5
        0
        1
        2
        3
I:      4
y-      73898
"Hi."   Hi.
px:       0x2100 (Vi ćete možda dobiti neku drugu vrednost)
*px:
```

Makro u linijama 5-10 obezbeduje prikaz tekućih vrednosti prosledemh parametara. Primetite da je prvo dostavljena cout-u stingovana verzija parametra; stoga, ako prosledite x, cout će primiti "x'\ Zatim, cout će primiti string u navodnicima ":\t", koji će prikazati dvotačku i tab karakter. Treće, cout primiti vrednost parametra (x) i na kraju endl, koje će ispisati novu liniju, i lzvrsiti flush bafera.

Nivoi debagovanja

U velikim kompleksnim projektima poželećete da imate veću kontrolu od jednostavnog uključivanja i isključivanja DEBUG-a. Možete definisati nivoe debagovanja i testirati na tim nivoima, tako što ćete odlučivati koje makroe da uključite, a koje da isključite.

Da biste definisali nivo, jednostavno iza Idefine DEBUG naredbe dodajte broj. S obzirom da možete imati proizvoljan broj nivoa, preporučili bismo Vam da se, ipak, zadržite na uobičajena četiri: HIGH, MEDIUM, LOW i NONE. U listingu 17.8 ilustrovano je kako se ovo može ostvariti, korišćenjem String i Animal klasa iz listinga 17.6. Definicije svih metoda, osim InvariantsO. su radi uštede prostora, izostavljene, ali su i neizmenjene u odnosu na listing 17.6.

yNAPOMENA's Da biste kompajlirali ovaj kod, kopirajte linije 43 do 136 iz listinga 17.6 između linija 64 i 65 ovog listinga.

Listing 17.8: Nivoi dibagiranja.

```
o   enum LEVEL { NONE, LOW, MEDIUM, HIGH };
I   const int FALSE = 0;
2   const int TRUE = 1;
3   typedef int B00L;
4
5    Idefine DEBUGLEVEL HIGH
6
7    linclude <iostream.h>
8    linclude <string.h>
9
10   lif DEBUGLEVEL < LOW  // mora biti srednji ili visok
11   Idefine ASSERT(x)
12   lelse
13   Idefine ASSERT(x) \
14       if (! (x)) \
15       { \
16          cout « "ERROR!! Assert " « Ix « " failed\n"; \
17          cout « " on line 4 «____LINE____« "\n"; \
18          cout « " in file " «____FILE___« "\n";  \
19       }
20   lendi f
21
22   lif DEBUGLEVEL < MEDIUM
23   Idefine EVAL(x)
24   lelse
25   Idefine EVAL(x) \
26     cout « Ix « ":\t" « x « endl;
27   lendif
28
29   lif DEBUGLEVEL < HIGH
```

nastavlja se

Listing 17.8: Nivoi dibagiranja.

```
#define PRINT(x)
lelse
ldefine PRINT(x) \
  cout « x « endl;
lendif


class String
{
   public:
     // konstruktori
     StringO;
     String(const char *const);
     String(const String &);
     -StringO;

     char & operators(int offset);
     char operators(int offset) const;

     String & operator= (const String &);
     int Getl_en()const { return itsLen; }
     const char * GetStringO const
      { return itsString; )
     BOOL InvariantsO const;

   private:
     String (int);          // privatni konstruktor
     char * itsString;
     unsigned short itsLen;
};

BOOL String::InvariantsO const
{
    PRINT("(String Invariants Checked)");
    return ( (BOOL) (itsLen && itsString) đđ
        (MtsLen && MtsString) );
}

class Animal
{
public:
   Animal():itsAge(l),itsName("John Q. Animal")
       (ASSERT(InvariantsO);}

   Animal(int, const Strings);
   -Animal(){}

   int GetAgeO
```



```
       {
           ASSERT(InvariantsO);
           return itsAge;
       }

    void SetAge(int Age)
       {
           ASSERT(InvariantsO);
           itsAge = Age;
           ASSERT(InvariantsO);
       }
    String& GetNameO
       {
           ASSERT(InvariantsO);
           return itsName;
       }

    void SetName(const String& name)
       {
           ASSERT(InvariantsO);
           itsName = name;
           ASSERT(InvariantsO);
       }

    BOOL InvariantsO;
private:
    int itsAge;
    String itsName;
};

BOOL Animal::Invariants()
{
    PRINT("(Animal Invariants Checked)");
    return (itsAge > 0 && itsName.GetLen());
}

int main()
{
    const int AGE = 5;
    EVAL(AGE);
    Animal sparky(AGE,"Sparky");
    cout « "\n" « sparky.GetNameO .GetStringO ;
    cout « " is ";
    cout « sparky.GetAgeO « " years old.";
    sparky.SetAge(8);
    cout « "\n" « sparky.GetNameO .GetStringO ;
    cout « " is
    cout « sparky.GetAgeO « " years old.";
    return 0;
}
```

```
AGE:    5
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)
(String  Invariants  Checked)

Sparky  is  (Animal  Invariants  Checked)
5  Years  old.  (Animal  Invariants  Checked)
(Animal  Invariants  Checked)
(Animal  Invariants  Checked)

Sparky  is  (Animal  Invariants  Checked)
8  years  old.  (String  Invariants  Checked)
(String  Invariants  Checked)

// pokrenut ponovo sa DEBUG = MEDIUM

AGE:    5
Sparky  is  5  years  old.
Sparky  is  8  years  old.
```

J£23Ejfc> U linijama 10-20 definisan je makro assert (), koji će biti odsečen, ako ' ^ je DEBUGLEVEL manji od LOW (što će red, DEBUGLEVEL je NONE). Ako je uključeno debagovanje, makro assert () će funkcionisati. U liniji 23 EVAL je deklarisano tako da bude isključeno, ako je DEBUGLEVEL manji od MEDIUM; ako je DEBUGLEVEL NONE, ili LOW, EVAL će biti isključen.

Na kraju, u linijama 29-34 deklarisan je makro PRINT, koji će biti isključen, ako je DEBUGLEVEL manji od HIGH. Makro PRINT će se koristiti samo u slučajevima kada je DEBUGLEVEL podignut na HIGH; možete eliminisati ovaj makro, tako što ćete postaviti DEBUGLEVEL na MEDIUM, a pri torn ćete i dalje zadržati makroe EVAL i assert ().
Makro PRINT se koristi unutar InvariantsO metoda za prikazivanje poruka. Makro EVAL se koristi u liniji 117, kako bi izračunao tekuću vrednost celobrojne konstante AGE.

PAzrn *II*> Koristite velika slova za imena makroa. Ovo je uobičajena konven cija i drugi programeri bi bili zbunjeni, ako to ne biste činili.

Nemojte dozvoliti svojim makroima da imaju propratne efekte. Nemojte povećavati promenljive, ili dodeljvrednosti unutar makroa.

Postavite sve argumente u zagrade unutar makro funkcija.

Rezime

Danas ste naučili vise detalja o radu sa pretprocesorom. Svaki put kada startujete kompajler, prethodno se startuje pretprocesor, koji vrši translaciju Vaših pretprocesorskih direktiva, kao što su ldefine i lifdef.

Pretprocesor vrši zamenu teksta, mada, zajedno sa korišćenjem makroa, ovo može postati vrlo složeno. Korišćenjem lifdef, lelse i lifndef, možete završiti uslovnu kompilaciju i kompajliranje odredenih naredbi, pod jednim uslovima, ili drugog skupa naredbi, pod drugim uslovima. Ovo može pomoći pri pisanju programa za vise od jedne platforme, a često se koristi za uslovno ukljudvanje informacija za debagovanje.

Makro funkcije obezbeđuju složenu zamenu teksta, koja se bazira na argumentima prosleđenim makrou u trenutku kompilacije. Vrlo je važno staviti svaki argument u makrou u zagrade, kako biste bili sigurni da će se izvršiti odgovarajuća zamena.

Makro funkcije manje su važne u C + + -u, nego što su bile u C-u. C + + obezbeduje veliki broj osobina jezika, kao što su const promenljive i templejti, koji predstavljaju superiornu alternativu korišćenju pretprocesora.

Pitanja i odgovori

P Ako C++ nudi bolje alternative od pretprocesora, zašto je ova opcija i dalje na raspolaganju?

O Najpre, C + + je vertikalno kompatibilan sa C-om i, stoga , svi važni delovi C-a moraju biti podržani u C++-u. Drugo, postoje neki slučajevi korišćenja pretprocesora koji su i dalje u čestoj upotrebi u C + + - U , kao što je uključivanje zaštite.

P Zašto koristiti makro funkcije, kada se mogu koristiti regularne funkcije?

O Makro funkcije su inline i koriste se kao zamena za stalno "prepucavanje" istih komandi sa manjim varijacijama. Još jednom da kažemo da, templejti, ipak, nude bolje alternative.

P Kako da znate kada da koristite makro, a kada inline funkcije?

O Cesto, to i nije mnogo važno. Koristite šta Vam je jednostavnije. Medutim, makroi nude zamenu karaktera, stringovanje i konkatenaciju. Ništa Vam od ovoga nije na raspolaganju kada koristite funkcije.

P Koja je alternativa korišćenju pretprocesora za prikazivanje meduvrednosti tokom debagovanja?

0 Najbolja alternativa je koriscenje watch naredbi unutar debagera. Za informacije o watch naredbama konsultujte svoju dokumentaciju za kompajler, ili debager.

P **Kako da odlučite kada da koristite assert (), a kada da napravite izuzetak?**

O U situacijama kada Vaše testiranje može biti tačno, čak i da ne dobijete grešku, koristite izuzetke. Ako je jedini razlog, kada je ova situacija u pitanju, koristite assert().

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Pitanja

1. Štaje uključivanje zaštite?
2. Kako ćete objasniti svom kompajleru da prikaže sadržaj pomoćne datoteke, da biste prikazali efekte pretprocesora?
3. Koja je razlika izmedu Idefine debug 0 i lundef debug?
4. Navedite četiri predefinisana makroa.
5. Zašto ne možete pozvati InvariantsO u prvoj liniji svog konstruktora?

Vežbe

1. Napišite naredbe za uključivanje zaštite za datoteku zaglavlja STRING.H.
2. Napišite assert () makro koji, prikazuje poruku o grešci, datoteku i broj linije, ako je nivo debagovanja 2, samo poruku (bez datoteke i broja linije), ako je nivo **1,** i ništa, ako je nivo nula.
3. Napišite makro DPrint koji proverava da li je definisan DEBUG i, ako jeste, prikažite vrednost koja je prosledena kao parametar.
4. Napišite funkciju, koja prikazuje poruku o grešci. Funkcija bi trebalo da prikaže broj linije i ime datoteke gde se greška desila. Imajte u vidu da se broj linije i ime datoteke prosleduju ovoj funkciji.
5. Kako biste nazvali prethodnu funkciju za obradu greške?
6. Napištie assert() makro, koji koristi funkciju za obradu greške iz Vežbe 4, i napišite drajver program, koji poziva taj assert () makro.

Dan 18

Objektno-orijentisana analiza i dizajn

Lako može da se dogodi da, usredsredeni na sintaksu C+ + , izgubite iz vida kako se i zašto koriste ove tehnike u izgradnji programa. Danas ćete naučiti:

- kako da analizirate probleme iz objektno-orijentisane perspektive
- kako da dizajnirate program iz objektno-orijentisane perspektive
- kako da dizajnirate programe koje ćete mod ponovo da koristite i proširujete.

Razvojni ciklus

Mnogo knjiga je napisano na temu razvojnog ciklusa. Neki predlažu metod "vodopada", u kome dizajneri određuju šta program treba da radi, projektanti odreduju kako će program biti "izgraden", koje će klase koristiti i tako dalje, i, na kraju, dolaze programeri, koji implementiraju dizajn i arhitekturu. Vremenom, arhitektura i dizajn su prepušteni programerima, i to u celini. Sve što programer treba da uradi je da implementira zahtevanu funkcionalnost.

Cak i ako metod vodopada dobro funkcioniše, on je loš metod za pisanje dobrih programa. Kada programer dode na red da uradi svoj posao, on obavezno nailazi na probleme šta je napisano do sada i šta je ostalo da se uradi. I dok je, sa jedne strane, tačno da su dobri C + + programi dizajnirani do sitnih detalja, pre nego što je napisana ijedna linija koda, sa druge strane ne bismo nikako mogli prihvatiti tvrdnju da će taj dizajn ostati nepromenjen do završetka ciklusa.

Količina dizajniranog "materijala", koja mora biti završena pre nego što započne programiranje, je u funkciji veličine programa. Izuzetno složeni programi, koji uključuju desetine programera i njihov višemesečni rad, pre će zahtevati potpunije artikulisanu arhitekturu, nego što će to biti potrebno kratkim rutinama (napisao ih je za jedan dan programer).

Ovo poglavlje će se, u prvom redu, baviti dizajnom velikih, složenih programa, koji će biti proširivani i menjani tokom vise godina. Mnogi programed uživaju da rade na "oštroj ivici" tehnologije; oni teže da napišu programe, čija kompleksnost doseže do krajnjih mogućnosti korišćenih alata i granica razumevanja. Na mnoge načine C++ je dizajniran da proširi složenost, koju programer, ili tim programera može da održava.

U ovom poglavlju ispitaćemo veći broj dizajnerskih problema iz objektno-orijentisane perspektive. Cilj će biti da upoznamo proces analize i da razumemo kako se primenjuje sintaksa C++ u implementaciji tog dizajna.

Simulacija alarmnog sistema

Simulacija je kompjuterski model nekog sistema iz realnog sveta. Postoji vise razloga za izgradnju simulacije, ali uspešan dizajn mora da krene od razumevanja samih pitanja, na koja će, nadajmo se, simulacija uspeti da odgovori.

Na samom početku, ispitajmo sledeći problem: od Vas je traženo da simulirate alarmni sistem jedne kuće. Kuća je kolonijalnog tipa, sa centralnim holom, četiri spavaće sobe, suterenom i garažom ispod kuće.

U prizemlju postoje sledeći prozori: tri u kuhinji, četiri u trpezariji, jedan u malom kupatilu, po dva u dnevnoj i primaćoj sobi i dva mala odmah pored vrata. Sve četiri spavaće sobe se nalaze na spratu i svaka ima po dva prozora, osim glavne spavaće sobe, koja ima četiri. Postoje, takođe, i dva kupatila sa po jednim prozorom. Na kraju, u suterenu postoje Četiri poluprozora i jedan prozor u garaži.

Kuća, normalno, ima ulazna vrata. Uz to, kuhinja ima klizna staklena vrata, a garaža ima dvoja vrata za kola i jedna vrata za jednostavniji pristup suterenu. U bašti (iza kuće) postoje vrata za podrum.

Sva vrata i prozori su pokriveni alarmnim sistemom, pa se dugmići za "uzbunu" nalaze na svakom telefonu i odmah pored kreveta. Dvorište je takođe "pokriveno" alarmnim sistemom, ali je kalibracija pažljivo izvršena, kako alarm ne bi aktivirale male životinje, ili ptice.

U suterenu je smešten centralni alarmni sistem, koji proizvodi upozoravajući zvuk kada se alarm uključi. Ako alarm ne bude isključen u okviru podešenog vremenskog perioda (intervala), biće pozvana policija. U slučaju da je aktivirano dugme za uzbunu, policija će biti trenutno pozvana.

jip
ш Alarm je takode povezan sa detektorima vatre i dima i sprinkler sistemom; sam alarmni sistem ima ugrađenu toleranciju na greške, poseduje sopstvene baterije za neprekidno napajanje i smešten je u kutiju koja je zaštićena od požara.

Preliminarni dizajn

Na koja pitanja bi simulacija trebalo da odgovori? Na primer, možda ćete želeti da simulacija odgovori koliko dugo senzor može biti polomljen, pre nego što to neko primeti ili, da li je moguće savladati alarme na prozorima, a da o tome policija ne bude obaveštena.

Kada shvatite svrhu simulacije, znaćete koji deo realnog sistema program mora da modelira. Kada ovo budete dobro razumeli, biće mnogo jednostavnije dizajnirati sam program.

Koji objekti postoje?

Jedan od načina rešavanja ovog problema je da se odvojite od poslova "vezanih" za korisnički interfejs i da svu pažnju usmerite samo na komponente u "problemskom prostoru." Prva aproksimacija objektno-orijentisanog dizajna može biti lista objekata koje želite da simulirate i da ispitate šta ti objekti "znaju" i šta "rade."

^VJJyjIJш) *Problemski prostor* je skup problema, koje Vaš program pokušava da reši. *Prostor rešenja* je skup mogućih rešenja problema.

Na primer, jasno je da imate senzore različitih tipova: centralni alarmi sistem, dugmiće, žice i telefone. Nadalje je jasno da morate da simulirate sobe, verovatno i spratove, kao i grupe ljudi, kao što su vlasnici i policija.

Senzori se mogu podeliti na detektore kretanja, zvuka, dima i tako dalje. Sve su ovo tipovi senzora i ne postoji takva stvar koja bi mogla da predstavlja senzor sam za sebe. Ovo je dobra indikacija da je senzor apstraktni tip podataka (ADT).

Kao ADT, klasa senzor će obezbediti kompletan interfejs za sve tipove senzora, a svaki izvedeni tip će obezbediti implementaciju. Klijenti različitih senzora će ih koristiti ravnopravno, bez potrebe da znaju išta o tipu senzora, a svaki od njih će raditi ono što je potrebno, u zavisnosti od svog realnog tipa.

Da biste kreirali dobar ADT, potrebno je da kompletno razumete šta, a ne kako senzori rade. Na primer, da li su senzori pasivni, ili aktivni uredaji? Da li čekaju da se neki element ugreje, da se žica prekine, ili da se neki deo istopi? Da li testiraju svoje okruženje? Pretpostavimo da neki senzori mogu da imaju samo binarna stanja (alarmno, ili redovno stanje). Ali drugi mogu imati vise analognih stanja (na primer, koja je trenutna temperatura). Interfejs za apstraktni tip podataka bi trebalo da bude što kompletniji, kako bi podržao sve anticipirane potrebe izvedenih klasa.

Drugi objekti

Dizajniranje se nastavlja u sledećem pravcu: pronalaženjem drugih klasa koje će biti zahtevane, kako bi se zadovoljila specifikacija. Na primer, ako se ukaže potreba za vodenjem dnevnika, verovatno će biti potreban tajmer. Da li će tajmer prozivati senzore ili će svaki senzor, periodično, sam davati izveštaj?

Korisnik će imati potrebu za podešavanjem, isključivanjem, ili programiranjem sistema i tada će biti neophodna neka vrsta terminala. Možete poželeti da odvojite objekte u Vašoj simulaciji od samog programa za alarm.

Koje klase postoje?

Kada rešite ovaj problem, počećete da dizajnirate svoje klase. Na primer, već imate indikaciju da će se HeatSensor izvesti iz klase Sensor. Ako Sensor periodično daje izveštaje, on se, takođe, može izvesti putem višestrukog nasledivanja iz tajmera, ili može imati tajmer kao promenljivog člana.

HeatSensor će, verovatno, imati funkcije clanove, kao što su CurrentTempQ i SetTempLimitO, i naslediti funkcije, kao sto je SoundAlarm(), iz svoje bazne klase Sensor. Često korišćen termin u oblasti objektno-orijentisanog dizajniranja je enkapsulacija. Možete zamisliti dizajn u kome alarmni sistem ima postavljenu maksimalnu temperaturu (MaxTemp). Alarmni sistem pita toplotni senzor za trenutnu temperaturu, upoređuje je sa maksimalnom temperaturom i aktivira alarm ako je previse toplo. Neki će se složiti, da se ovim narušava princip enkapsulacije. Verovatno bi bilo bolje kada alarmni sistem ne bi znao, ili vodio računa o detaljima temperaturne analize. To bi trebalo da dešava u toplotnom senzoru (HeatSensor).

Bez obzira da li ćete se Vi složiti sa ovim argumentom ili ne, to je vrsta odluke na koju treba da se usredsredite tokom analize problema. Da bismo nastavili ovu analizu, trebalo bi da se složimo da samo senzor i Log objekat treba da znaju sve detalje kako je voden dnevnik senzora; alarm objekat ne bi trebalo da o tome vodi računa.

Dobra enkapsulacija se ogleda u tome da svaka klasa ima koherentan i kompletan skup svojih odgovornosti i da ni jedna druga klasa nema istu takvu odgovornost. Ako je senzor odgovoran za notiranje trenutne temperature, ni jedna druga klasa ne bi trebalo da ima tu odgovornost.

S druge strane, druge klase mogu da pomognu da se ostvari željena funkcionalnost. Na primer, dok odgovornost klase Sensor može da bude da konstatuje i upiše trenutnu temperaturnu vrednost, takode je moguće implementirati tu odgovornost, putem delegacije, Log objektu, koji bi se bavio upisom i snimanjem podataka.

Održavanjem fine podele odgovornosti, Vaš program će postati jednostavniji za proširivanje i održavanje. Kada odlučite da zamenite alarmni sistem unapredenim modelom, njegov interfejs za log i senzore će biti prirodan i dobro definisan. Izmene u alarmnom sistemu ne bi trebalo da obuhvate klase Sensora, niti obrnuto.

Da li je neophodno da HeatSensor ima funkciju ReportAlarmO? Svim senzorima će biti neophodna mogućnost da prijave alarm. Ovo je dobra indikacija da ReportAlarmO bude virtuelna metoda klase senzor i da senzor treba da bude apstraktna bazna klasa. Moguće je da će se HeatSensor povezati na uopšteniji ReportAlarmO metod senzor klase; funkcija će samo popuniti detalje za koje je isključivo ona kvalifikovana da ih obezbedi.

Kako se prijavljuju alarmi?

Kada Vaš senzor prijavi stanje alarma, biće potrebno da se obezbedi odredena količina informacija objektu, koji će pozvati policiju i aktivnost upisati u dnevnik. Bilo bi dobro da kreirate klasu Condition, čiji će konstruktori će preuzimati veći broj merenja. U zavisnosti od složenosti merenja, ovo mogu biti objekti, ili jednostavne skalarne vrednosti, kao što su integer-i.

Moguće je da se Condition objekti proslede centralnom Alarm objektu, ili da Condi tion objekat bude potklasiran u alarm objekte, koji sami znaju kako da izvedu operacije u hitnim slučajevima. Pretpostavimo da ne postoji centralni objekat; umesto njega, postoje senzori, koji znaju kako se kreiraju Condition objekti. Neki Condi ti on objekti će znati kako da samostalno vode dnevnik, a drugi će znati kako da kontaktiraju policiju.

Dobro dizajniran, dogadajima voden sistem nema potrebu za centralnim koordinatorom. Možete zamisliti da senzori nezavisno primaju i šalju poruke jedni drugima, postavljaju parametre, vrše očitavanja i nadgledaju kuću. Kada se desi izvestan dogadaj, kreira se Al arm objekat, koji upisuje problem u dnevnik (slanjem poruke Log objektu) i preuzima odgovarajuću akciju.

Petlje događaja

Da biste simulirali ovakav, dogadajima voden sistem, Vaš program mora da kreira petlju dogadaja. Ona je, obično, beskonačna, kao, na primer, whi le(1), koja preuzima poruke od operativnog sistema (klikove mišem, pritiske na tastaturu) i obraduje ih jednu, po jednu i zatim se vraća u petlju, dok se ne zadovolji kriterijum za izlaz. U listingu 18.1 prikazana je rudimentarna petlja dogadaja.

Listing 18.1: Jednostavno petlja dogadaja

```
1:    // Listing 18.1
2:
3:    linclude <iostream.h>
4:
5:    class Condition
6:    {
7:    public:
8:       Condition() { }
```

nastavljase

Listing 18.1: Jednostovna petlja dogadaja

```
9:        virtual -Condition() {}
10:       virtual void Log() = 0;
11:   (
12:
13:    class Normal : public Condition
14:    {15:    public:
16:       Normal () { Log(); }
17:       virtual -Normal() {}
18:       virtual void Log() { cout « "Logging normal conditions.. An
19:    {
20:
21:    class Error : public Condition
22:    {23:    public:
24:       Error() {LogO;}
25:       virtual -Error() {}
26:       virtual void LogO { cout « "Logging error!\n"; }
27:    {
28:
29:    class Alarm : public Condition
30:    {
31:    public:
32:       AlarmO;
33:       virtual   -AlarmO {}
34:       virtual void WarnQ { cout « "Warning!\n"; }
35:       virtual void LogO { cout « "General Alarm log\n"; }
36:       virtual void Call() = 0;
37:
38:    {
39:
40:    Alarm: :Alarm()
41:    {
42:        Log();
43:       Warn();
44:    }
45:    class FireAlarm : public Alarm
46:    {
47:    public:
48:       FireAlarm() {LogO ;{
49:       virtual -FireAlarm() {}
50:       virtual void Call() { cout« "Calling Fire Dept.!\n"; }
51:       virtual void LogO { cout « "Logging fire call.\n"; }
52:    {
53:
54:     int main()
55:    {
56:       int input;
57:       int okay = 1;
58:       Condition * pCondition;
59:       while (okay)
60:       {
61:          cout « "(0)Quit (l)Normal (2)Fire:
62:          cin » input;
63:          okay = input;
64:          switch (input)
65:          {
66:             case 0: break;
67:             case 1:
68:                pCondition = new Normal;
69:                delete pCondition;
70:                break;
71:             case 2:
72:                pCondition = new FireAlarm;
73:                delete pCondition;
74:                break;
75:             default:
76:                pCondition = new Error;
77:                delete pCondition;
78:                okay = 0;
79:                break;
80:          }
81:       }
82:    return 0;
83:
```

```
(0)Quit (l)Normal (2)Fire: 1
Logging normal conditions...
(0)Quit (l)Normal (2)Fire: 2
General Alarm log
Warning!
Logging fire cal 1.
```

J 3 S t > Jednostavna petlja je kreirana u linijama 59-80 i omogućava korisniku da unese simulaciju normalnog izveštaja sa senzora, ili da prijavi požar. Primetiće da je efekat ovog izveštaja da kreira Condition objekat, čiji konstruktor kreira različite izveštaje.

Pozivanje virtuelne funkcije člana iz konstruktora, može dovesti do zbunjujućih rezultata, ako ne znate redosled konstruisanja objekata. Na primer, kada je objekat FireAlarm kreiran u liniji 72, redosled konstrukcije je Condition, Alarm, FireAlarm. Alarm konstruktor poziva Log (dnevnik), ali to je Alarmov Log(), a ne FireAlarmov koji je pozvan, s obzirom da je Log() deklarisan kao virtuelan. Ovo je stoga što u trenutku rada alarmovog konstruktora nije postojao FireAlarm objekat. Kasnije, kada je FireAlarm konstruisan, njegov konstruktor je pozvao Log() i tada je pozvan FireAlarm::Log().

PostMaster

Postoji još jedan problem, sa kojim ćete se susretati u Vašoj objektno-orijentisanoj analizi. Pretpostavimo da ste se zaposlili u Acme Software, Inc., da ste započeli novi softverski projekat i zaposlili tim C+ + programera, koji će izvršiti implementaciju Vašeg programa. Jim Grandiose, potpredsednik firme, je Vaš novi šef. On je od Vas tražio da dizajnirate, izgradite i napravite PostMaster, rutinu za citanje elektronske pošte, koja stiže od različitih nepovezanih e-mail provajdera. Potencijalni kupci su zaposleni ljudi koji koriste vise od jednog e-mail proizvoda, na primer, Interchange, CompuServe, Prodigy, America Online, Delphi, Internet Mail, Lotus Notes, AppleMail cc:Mail i tako dalje.

Kupac bi trebalo da "nauči" PostMaster kako da pozove, ili se na drugi način poveže sa bilo kojim od e-mail provajdera. PostMaster bi tada trebalo da preuzme poštu i da je prikaže u uniformnom obliku, omogućavajući korisniku da poštu organizuje, piše odgovore, prosleduje poruke kroz razne servise i tako dalje.

PostMasterProfessional koji bi bio razvijen kao verzija 2 PostMastera je već prihvaćen. U njemu je dodat Administrative Assistant mod radi omogućavanja korisniku da zabrani drugim ljudima čitanje delova, ili kompletne pošte, da podržava rutinsku korespondenciju i tako dalje. Takođe, postoje nagađanja u odeljenju za marketing da bi mogla biti pridodata komponenta veštačke inteligencije, koja bi omogudla PostMasteru da presortira poštu,i dodeli joj prioritete, i to na osnovu ključnih red po predmetu i sadržaju, kao i prema asocijacijama.

Govorilo se i o drugim unapredenjima, kao, na primer, o mogućnosti podrške ne samo elektronskoj pošti, već i diskusionim grupama, kao što su Interchange diskusije, CompuServe forumi, Internet news grupe i tako dalje. Jasno je da Acme polaže velike nade u PostMaster, a Vi ste, naravno, pritisnuti vremenskim rokovima, tu da ga izvedete na tržište, koliko ste mogli da vidite, sa praktično neograničenim budžetom.

Dvaput men, jednom seci

Uredili ste svoju kancelariju i složili opremu i sada je Vas prvi poslovni zadatak da nabavite dobru specifikaciju proizvoda. Pošto ste ispitali tržište, odlučili ste da predložite da razvoj bude koncentrisan na jednokorisničke platforme i mogli ste da se odludte između DOS-a, UNIX-a, Macintosh-a, Windows-a, WIndowsNT i OS/2.

Imali ste mnogo "bolnih" sastanaka sa potpresednikom Jimom Grandiosem, kome je postalo jasno da ne postoji pravi izbor. Stoga ste Vi odludli da odvojite korisnički interfejs od komunikacionog dela i dela koji se bavi bazom podataka. Da biste što bolje upoznali problematikom, odludli ste da prvo pišete za DOS, zatim za Win32, Mac, pa tek onda za UNIX i OS/2.

Ova jednostavna odluka je dovela do enormnog grananja Vašeg objekta. Brzo je postalo jasno da će Vam biti potrebne biblioteke klasa, pa, čak, i serije biblioteka za podršku i upravljanje memorijom za različite korisničke interfejsove, a, verovatno, i za komunikaciju, kao i komponente baze podataka.

Potpresednik Grandiose je ubeden da jedan projekat živi, ili umire, u zavisnosti od toga da li postoji osoba sa jasnom vizijom, pa Vas je, stoga, zamolio da izvršite inicijalnu analizu strukture i dizajna, pre nego što zaposlite bilo kojeg programera.

Zavadi, pa vladaj!

Brzo je postalo jasno da Vi, u stvari, treba da rešite vise od jednog problema. Podelili ste projekat na sledeće glavne potprojekte:

1. Komunikacija: mogućnost da softver pozove e-mail provajder putem modema, ili da se poveže preko mreže.
2. Baza podataka: mogućnost da se smeste podaci i mogućnost da se ponovo dobiju sa diska.
3. E-mail: mogućnost da se dtaju razlidti e-mail formati i da se pišu nove poruke za bilo koji sistem.
4. Editovanje: obezbedenje kvalitetnog editora za kreiranje i manipulaciju poruka.
5. Platforme: razlidti korisnički interfejsovi za različite platforme (DOS, Windows).
6. Ekstenzivnost: planovi za rast i unapredenje.
7. Organizacija i planiranje vremena: upravljanje razlidtim ljudima u razvoju i njihovim zavisnostima u kodu. Svaka grupa mora da izradi i objavi plan rada, koji će, zatim, uskladiti sa ostalima. Rukovodeći kadar i služba marketinga moraju da znaju kada će proizvod biti spreman.

Odludli ste da zaposlite rukovodioca, koji će se baviti stavkom 7. Zatim ste zaposlili glavnog projektanta, koji bi Vam pomogao u analizi i dizajnu, a i u organizaciji i implementaciji preostalih delova. Ovi glavni projektanti će organizovati sledeće timove:

1. Za komunikacija: odgovorni za modemsku i mrežnu komunikaciju. Oni će se "boriti" sa paketima, strimovima i bitovima, pre nego sa samim e-mail porukama.
2. Za format poruka: odgovorni za konvertovanje poruka dobijenih od bilo kojeg e-mail provajdera u kanoničan oblik (PostMaster standard i nazad). Njihov posao je i, da njihove poruke upisuju na disk i da ih sa diska dtaju kada to bude potrebno.
3. Za editorie poruka: odgovorni za kompletan korisnički interfejs proizvoda na svim platformama. Njihov posao je da obezbede interfejs između svih delova proizvoda, koji neće dovesti do dupliranja koda, kada proizvod prede na novu platformu.

Format poruke

Odlučili ste da se prvo usredsredite na format poruke, dok ste po strani ostavili komunikaciju i korisnički interfejs. Ovo će uslediti kada budete potpunije razumeli ono čime se bavite. Nema mnogo smisla brinuti o tome kako predstaviti informacije korisniku, dok u potpunosti ne shvatite o kakvim se uopšte informacijama radi.

Ispitivanjem različitih e-mail formata, shvatili ste da oni imaju mnogo zajedničkih stvari, umesto razlika. Svaka e-mail poruka ima tačku u kojoj se nalazi izvor, tačku u kojoj se nalazi odredište i datum kreiranja. Skoro sve ovakve poruke imaju liniju za naslov, ili subject, kao i telo, koje se može sastojati od jednostavnog teksta, formatiranog teksta, grafike, ili, čak, muzike, ili drugih zabavnih "dodataka." Većina e-mail servisa podržava relaciono "prikačinjanje" datoteka, tako da korisnici mogu da šalju programe, ili bilo šta drugo.

Potvrdili ste svoju raniju odluku da ćete svaku poruku pročitati iz njenog originalnog formata i smestiti je u PostMaster format. Na ovaj način, imaćete samo jedan format sloga i pisanje i čitanje sa diska će biti uprošćeno. Takođe ste odludli da odvojite informacije iz zaglavlja (pošiljalac, primalac, datum, subject i tako dalje) od tela poruke. Korisnik će često poželeti da pretraži samo zaglavlja, bez dtanja sadržaja svih poruka. Može doći trenutak kada će korisnici želeti da preuzmu samo zaglavlja od provajdera, bez preuzimanja teksta, ali ste, za sada, odludli da verzija 1 PostMastera uvek preuzima jednu poruku, pošto je ne mora prikazivati korisniku.

Inicijalni dizajn klasa

Analiza poruka odvela Vas je u dizajniranje klase Message. Zbog namere da program proširujete porukama koje nisu e-mail tipa, klasu EmailMessage ste izveli iz apstaktne klase Message. Iz klase EmailMessage ste, zatim, izveli PostMasterMessage, InterchangeMessage, CISMessage, ProdigyMessage i tako dalje.

Poruke su prirodan izvor za objekte u programima za obradu e-mail poruka, ali pronalaženje svih ispravnih objekata u kompleksnom sistemu je najved izazov objektno-orijentisanog programiranja. U nekim prilikama, kao što je to slučaj sa porukama, primarni objekti izgledaju kao da će "iskodti" iznad nivoa Vašeg razumevanja problema. Medutim, mnogo češće će biti potrebno da dugo i dobro razmislite, šta pokušavate da uradite, kako biste pronašli odgovarajući objekat.

Nemojte biti razočarani. Većina dizajnerskih poduhvata nije savršena, kada se radi prvi put. Dobra polazna tačka je opis samog problema. Napravite listu svih imenica i glagola, koje koristite kada opisujete projekat. Imenice su dobri kandidati za objekte, dok glagoli mogu biti metodei tih objekata (ili mogu biti objekti sa sopstvenim pravima). Ovaj metod Vam ne garantuje da neće doći do greške, ali je dobra tehnika koju možete koristiti kada podnjete sa radom na novom dizajnu.

Ovo je bio lakši deo. Sada prelazimo na sledeća pitanja. Mora li zaglavlje poruke, kao klasa, biti razdvojeno od tela? Ako je odgovor da, da li su im potrebne paralelne hijerarhije CompuServeBody i CompuServerHeader, kao i **Prodi**gyBody i **Prodi**gyHeader? Paralelne hijerarhije su često znak upozorenja za loš dizajn. Česta je greška u objektno-orijentisanom dizajnu imati skup objekata u jednoj hijerarhiji i odredeni skup "menadžera" tih objekata u drugoj. Problemi da se ove hijerarhije održe ažurne i medusobno sinhronizovane su klasična noćna mora u održavanju.

Ne postoje čvrsta i jasna pravila. Naravno, postoje situacije kada su paralelne hijerarhije najbolje rešenje za određeni problem. Ako primetite da se Vaš dizajn kreće u ovom pravcu, potrebno je da o problemu ponovo razmislite; može se desiti da postoji mnogo elegantnije rešenje.

Kada poruke stižu od e-mail provajdera, one neće biti obavezno podeljene na zaglavlje i telo; vedna će biti jedan veliki niz podataka, koji Vaš program mora da reorganizuje. Pretpostavimo da Vaša hijerarhija treba direktno da reflektuje ovu ideju.

Druga refleksija zadataka koje imamo vodi nas, da pokušamo da prikažemo listu osobina ovih poruka, dok jednim okom posmatramo smeštanje podataka na odgovarajućem nivou apstrakcija. Spisak osobina Vaših objekata je dobar nadn kako za pronalaženje, tako i za "istresanje "drugih objekata koji Vam mogu zatrebati.

E-mail poruke se moraju negde smestiti, kao i brojevi telefona, adrese i tako dalje. To smeštanje jasno zahteva da bude visoko u hijerarhiji. Da li e-mail poruke obavezno moraju da dele baznu klasu sa telefonskim brojevima, ili adresama?

Korene hijerarhije protiv hijerarhija bez korena

Postoje dva pristupa realizaciji hijerarhije nasledivanja: možete imati sve, ili uglavnom sve Vaše klase koje izviru iz zajedničke korene klase, ili možete imati vise od jedne hijerarhije nasledivanja. Prednost zajedničke korene klase je, da ćete često izbed visestruko nasledivanje, a mana je da će, u vedni situacija, implementacija biti filtrirana u baznu klasu.

> Skup klasa je sa *korenom,* ako sve klase dele zajednički čvor. Hijerarhija *bez korena* dni da sve klase ne dele zajedničku baznu klasu.

Pošto znate da će Vaš proizvod biti razvijan na vise platformi i pošto je visestruko nasledivanje složeno i nisu ga dobro podržali svih kompajleri na razlidtim platformama, Vasa prva odluka je da koristite korenu hijerarhiju i jednostruko nasledivanje. Odludli ste da identifikujete ona mesta gde bi se eventualno moglo korisiti visestruko nasledivanje i dizajnirali ste tako što ste podelili hijerarhiju i dodali visestruko nasledivanje, ali ste ovaj posao, ipak, ostavili za kasnije. Odludli ste da date prefiks imenima svih Vaših internih klasa, i to malo slovo p, tako da brzo možete prepoznati koja je klasa Vaša, a koja je iz neke druge biblioteke. U Danu 21, "Sta dalje," učićete o davanju imena koja neće biti u skladu sa ovim što je rečeno, ali, za sada, ovo će dobro funkcionisati.

S^pp* Vaša korena klasa će biti pObject; virtuelno, svaka klasa koju kreirate će nastati iz ovog objekta. On će biti krajnje jednostavan; samo oni podaci koje apsolutno svaka stavka deli sa svima drugima će se pojaviti u ovoj klasi.

Ako želite korenu hijerarhiju, poželećete da korenoj klasi dodelite generičko ime (kao što je pObject) i nekoliko mogućnosti. Svrha korenog objekta je, da on kreira kolekciju od svih svojih naslednika i da im se predstavi kao kopija pObjecta. Loša strana korene hijerarhije je, što filtrira interfejs u korenu klasu. Platićete cenu tako što ćete filtrirati ove interfejsove u koreni objekat, a drugi naslednici će imati interfejsove koji nisu odgovarajući za njihov dizajn. Jedino dobro rešenje ovog problema u jednostrukom nasledivanju je koriscenje templejta, o kojima će biti red narednoj lekciji.

Mogud kandidati za vrh hijerarhije su pStored i pWired.; pStored objekat je sačuvan na disku u razlidtim trenucima (na primer, kada se program ne koristi), a pWired objekat se šalje mrežom, ili modemom. Pošto skoro svi Vaši objekti treba da budu smešteni na disk, ima smisla smestiti ovu funkcionalnost na vrh. Pošto svi objekti koji se šalju modemom moraju biti upisani, ali ne moraju svi upisani objektni biti poslati kroz žicu, ima smisla pWired izvesti iz pStored.

Svaka izvedena klasa dobija celokupno svoje znanje (podatke) i funkcionalnost (metode) od svoje bazne klase i svaka može da doda diskretne dodatne mogućnosti. Stoga, pWired može da doda razlidte metode, ali svi oni moraju da budu u službi dodavanja mogućnosti za prenos podataka modemom.

Moguće je da su svi pWired objekti sačuvani, ili da su svi pStored objekti pWired, ili da ni jedna od ovih izjava nije tačna. Ako samo neki od pWired objekata pStored i samo neki pStored objekat je pWired, biće nephodno da koristite visestruko nasledivanje, ili da "ishakerišete" problem. Potencijalno "hakerisanje" u ovoj situaciji bi bila mogućnost da nasledite, na primer, Wired iz Stored i, zatim, za one objekte koji se šalju modemom, ali se nikad ne upisuju, da napravite metode za upisivanje koji ne rade ništa, ili vraćaju grešku.

U stvari, Vi ste shvatili da se neki objekti koji se smeštaju, jasno, ne šalju žicom. Na primer, telefonski imenik. Medutim, svi oni objekti koji koji se šalju žicom se upisuju i stoga je Vaša hijerarhija nasledivanja do ovog trenutka onakva kakva je prikazana na slici 18.1.

Dizajniranje interfejsa

Na ovom nivou dizajniranja Vašeg proizvoda vrlo je važno izbeći bavljenje problemom implementacije. Vi želite da svu svoju energiju prvo usmerite na dizajniranje čistog interfejsa, koji povezuje klase, a da, zatim, opišete šta je potrebno podacima i metodama svake od klasa.

Često je poželjno imati solidno znanje o baznim klasama, pre nego što pokušate da dizajnirate izvedene klase, tako da ste odludli da u centar Vašeg interesovanj? postavite pObject, pStored i pWired.

Slika 18.1
Inicijalna hijerarhija nasledivanja

Korena klasa, pObject, imaće samo one podatke i metode, koji su zajednički za ceo Vaš sistem. Možda bi svaki objekat trebalo da ima jedinstven identifikacioni broj. Možete kreirati pID (PostMaster ID) i udniti ga danom pObject-a; ali prvo se morate zapitati da li je bilo kojem objektu, koji nije sačuvan i nije "ožičen" (wi red), potreban ovakav jedan broj. Naravno, ovo dalje nameće pitanje da li postoje bilo kakvi objekti koji nisu sačuvani, a deo su ove hijerarhije.

Ako takvih objekata nema, možda ćete poželeti da pogledate spajanje pObject i pStored u jednu klasu; na kraju krajeva, ako su svi objekti sačuvani, u čemu je svrha diferencijacije? Razmišljajući na ovakav nadn, shvatićete da možda postoje neki objekti, kao što su adresni objekti koje bi bilo korisno izvesti iz p0bject-a, ali koji nikada ne bi bili sami za sebe sačuvani; ako bi bili sačuvani, oni bi bili deo nekog drugog objekta.

To znad da Vam može biti od koristi ako u ovom trenutku imate zasebnu pObject klasu. Zamislite da postoji knjiga sa adresama, u kojoj je smeštena kolekcija pAddress objekata, i mada ni jedna pAddress nikada neće biti smeštena sama za sebe, biće korisno da svaka od njih ima sopstveni jedinstveni identifikacioni broj. Kao eksperiment, dodelite pID p0bject-u, što znad da bi pObject, u najmanju ruku, trebalo da izgleda ovako:

```
class pObject
{
public:
pObject ();
čpObject();
pID GetID()const;
void SetID();
private:
pID  itsID;
}
```

Postoji niz stvari koje bi trebalo da uzmete u obzir kada je reč o deklarisanju ove klase. Najpre, ova klasa nije deklarisana da se izvodi iz bilo koje druge; ovo je Vaša korena klasa. Drugo, ne postoje nikakve namere da se prikaže implementacija, pa čak ni kada je reč o metodima, kao što je, na primer, Get ID(), za koje je vrlo verova-tno da imaju inline implementaciju.

Kao treće, const metodi su već identifikovani; ali ovo je deo interfejsa, a ne implementacije. Na kraju, obuhvaćen je i novi tip podataka: pID. Time što ćete definisati pID kao tip, pre nego što ćete upotrebiti, na primer, unsigned long, Vaš dizajn će, umnogome, dobiti na fleksibilnosti.

Ako se stvar preokrene tako da Vam vise nije potreban unsigned long, ili ako taj unsigned long nije dovoljno veliki, imate mogućnost da izmenite pID. Ova izmena će se odraziti na sve slučajeve u kojima je upotrebljen pID, tako da nećete morati da pratite i editujete svaku datoteku koja ima pID.

Od sada ćete mod da koristite typedef za deklarisanje pID kao ULONG, koji ćete, zatim, deklarisati kao unsigned long. Ovim se nameće pitanje: Kuda idu ove deklaracije.

Kada programirate veliki projekat, neophodno je izvršiti celovit dizajn datoteka. Standardni pristup, koji ćete, inače, primeniti u ovom projektu, podrazumeva da se svaka klasa pojavljuje u sopstvenoj datoteci zaglavlja i da se implementacija metoda klase pojavljuje u pridruženoj CPP datoteci. Stoga, imaćete jednu datoteku pod imenom OBJECT.HPP i drugu pod imenom OBJECT.CPP. Vi ćete prihvatiti da imate i ostale datoteke, kao što su MSG.HPP i MSG.CPP, sa deklarisanjem pMessage i implementacijom njenih metoda, respektivno.

yMAPOMEMJ^ Pitanje sa kojim ćete se susretati tokom čitavog procesa dizajniranja Vašeg programa je koje se rutine mogu kupiti, a koje ćete sami morati da napišete? Vrlo je verovatno da ćete mod da iskoristite postojeće komercijalne biblioteke za rešavanje nekih, ili, možda, i svih Vaših komunikacionih problema. Troškovi licenciranja, kao i ostali netehnički problemi, moraju, takode, biti razmatrani.

U velikom broju slučajeva je prednost ako kupite jednu ovakvu biblioteku i usmerite svu svoju energiju na odredeni program, umesto da "izmišljate rupu na saksiji". Možda ćete, čak, poželeti i da razmotrite mogućnost kupovine biblioteka koje nisu posebno namenjene za korišćenje u radu sa C+ +, ako one mogu da Vam pruže osnovnu funkcionalnost koju biste, u suprotnom, morali sami da izmislite. Ovo bi moglo da Vam pomogne da posao završite u roku.

Građenje prototipa

Za veliki projekat, kao što je PostMaster, malo je verovatno da će Vaš prvobitni dizajn biti kompletan i savršen. I vrlo jednostavno ćete biti savladani nizom problema, a pokušaj da kreirate sve klase i kompletirate njihove interfejsove pre pisanja makarijedne linije radne verzije koda je pravi recept za uništenje.

Postoji mnogo dobrih razloga da svoj dizajn ispitate na protopipu - brzopoteznoj i ne baš mnogo "dstoj" radnoj verziji Vaše zamisli. Medutim, postoji veliki broj različitih prototipova i svaki od njih zahteva nešto drugo.

Prototip dizajna korisničkog interfejsa Vam pruža mogućnost da istestirate kako će Vaš proizvod izgledati u očima potencijalnih korisnika i kakav će osećaj kod njih izazvati.

Prototip funkcionalnosti može biti dizajniran bez interfejsa za krajnjeg korisnika, ali omogućava korisniku da isproba razne osobine, kao što su prosledivanje poruka i prikadnjanje (attaching) datoteka, bez brige o krajnjem interfejsu.

Konačno, *prototip arhitekture* može biti dizajniran tako da Vam da pruži mogućnost za razvijanje manjih verzija programa i za procenu, koliko dobro se Vaše dizajnerske odluke uklapaju u sam projekat.

Imperativ koji se nameće podrazumeva da ciljevi izrade Vaših prototipova budu jasni. Da li proveravate korisnički interfejs, da li eksperimentišete sa funkcionalnošću, ili radite na izradi scale modela za Vaš finalni proizvod? Dobar prototip arhitekture "izgradiće" slab prototip korisničkog interfejsa i obratno.

Pravilo 80/80

Dobro pravilo kaže da dizajniranje u ovoj fazi treba "uraditi" za one stvari koje 80% ljudi želi da radi 80 odsto vremena, a da preostalih 20 odsto treba ostaviti po strani i ne razmišljati o njima. Preostali delovi tog dizajna pojaviće se pre, ili kasnije, ali "srž" Vašeg dizajna bi trebalo da bude usmerena na pravilo "80/80."

Imajud ovo u vidu, možete odludti da počnete sa dizajniranjem glavnih klasa, a da sekundarne klase ostavite po strani. Kasnije, kada identifikujete vise klasa koje imaju sličan dizajn, ali samo sa malim razlikama, možete izabrati jednu reprezentativnu klasu i usmeriti svoju pažnju na nju, ostavljajud dizajn i implementaciju njenih bliskih "rodaka" po strani.

s|**NAPOMEN**/pr Postoji još jedno pravilo. To je pravilo 80/20, koje kaže da će prvih 20 odsto Vašeg programa zahtevati 80 odsto Vašeg vremena za kodiranje i da će preostalih 80 odsto Vašeg programa uzeti novih 80 odsto Vašeg vremena.

Dizajniranje klase PostMasterMessage

Imajud u vidu prethodna pravila, odludli ste da usmerite pažnju na PostMasterMessage. Ova klasa je ponajviše pod Vašom direktnom kontrolom.

Kao deo njenog interfejsa, PostMasterMessage će imati potrebu da komunicira sa drugim tipovima pod Vašom direktnom kontrolom.

Kao deo njenog interfejsa, **PostMasterMessage će** imati potrebu da komunicira sa **drugim tipovima poruka.** Pretpostavili ste **da ćete biti u** mogućnosti **da razumete poruke drugih provajdera i da ćete dobiti specifikacije formata za** njihove poruke, **ali** za sada **možete** samo **nagađati šta će** biti poslato Vašem kompjuteru kada budete koristili njihove usluge.

U svakom slučaju, poznato Vam jeda **će** svaka **PostMasterMessage** imati pošiljaoca, primaoca, datum i temu, baš kao i telo poruke, a, verovatno, i priključene datoteke. Ovo govori da **će** Vam biti **potrebne** pristupne **i** metodi **za** svaki od ovih atributa, kao i metodi koje će Vam saopštiti veličinu priključene datoteke, veličinu poruka, itd.

Neki od servisa sa kojima ćete se povezati koriste tekst sa instrukcijama za formatizaciju, kojima se odreduju font, veličina karaktera i atributi, kao što su bold i italic. Drugi servisi ne podržavaju ove atribute i mogu, ali i ne moraju, koristiti sopstvene seme osobina za upravljanje tekstom. Vašoj klasi će biti potrebni metodi za konverziju, za pretvaranje formatizovanog teksta u običan ASCII, kao i za pretvaranje različitih formata u **Post Master Format.**

Interfejs aplikacionog programa

Application Program Interface (API) je skup dokumenata i rutina za koriscenje servisa. Većina e-mail provajdera će Vam dati API, tako da će PostMaster mod da iskoristi prednosti njihovih naprednih funkcija, kao što je, na primer, formatizovani tekst.PostMaster će, takođe, želeti da objavi svoj API, tako da drugi provajderi mogu da planiraju rad sa PostMasterom u budućnosti.

Vaša **PostMasterMessage** klasa će želeti da ima dobro dizajniran javni interfejs, a funkcije za konverziju će biti glavna komponenta PostMasterovog API-ja. U listingu 18.2. prikazano je kako u ovom trenutku izgleda **PostMasterMessage** Interface.

Listing 18.2: Interfejs PostMasterMessagea.

```
1:    class PostMasterMessage : public MailMessage
2:    {
3:    public:
4:       PostMasterMessage();
5:       PostMasterMessage!
6:            pAddress Sender,
7:            pAddress Recipient,
8:            pString Subject,
9:            pDate creationDate);
10:
11:      // ovde dolaze drugi konstruktori
12:      // ne zaboravite da uključite konstruktor kopije
13:      // kao i konstruktor sa skiadištenja
14:      // i konstruktor iz žičanog formata
15:      // Takođe uključite konstruktore iz drugih formata
16:      -PostMasterMessageO;
```



```
17       pAddress& GetSender() const;
18       void SetSender(pAddress&);
19       // druge metode pristupa
20
21       // ovde dolaze operatori, uključujući operator jednako
22       // i rutine za konverziju za pretvaranje PostMaster poruke
23       // u poruke drugih formata.
24
25    private:
26       pAddress itsSender;
27       pAddress itsRecipient;
28       pString  itsSubject;
29       pDate itsCreationDate;
30       pDate itsLastModDate;
31       pDate itsReceiptDate;
32       pDate itsFirstReadDate;
33       pDate itsLastReadDate;
34    {
```

Nema.

SSSJE* Klasa PostMasterMessage je definisana tako da se izvodi iz **MailMessage.** Biće obezbeden ved deo konstruktora, koji će omogudti kreiranje klase **PostMasterMessage i** drugih tipova poruka.

Prihvaćen je i ved broj pristupnih metoda za dtanje i postavljanje različitih podataka danova, kao i operatora za pretvaranje dela poruka i cele poruke u druge formate poruka. Odludli ste da ove poruke smeštate na disk, a da ih dtate kroz "žicu" (modem, ili mrežu), tako da su Vam i za ovu namenu potrebni pristupni metodi.

Programiranje u velikim grupama

Čak i ova preliminarna arhitektura je dovoljna da pokaže šta će biti potrebno da razlidte grupe za razvoj urade. Grupa za komunikaciju može da započne svoj rad, tako što će dogovoriti interfejs sa grupom koja "radi" na formatu poruke.

Grupa koja radi na formatu poruke će verovatno uspostaviti opšti interfrejs za **Message** klase koji je već započet, a zatim će svoju pažnju usmeriti na pitanja" vezana" za upis i dtanje podataka sa diska. Kada budu razumeli ovaj disk interfejs, članovi grupe će biti u dobroj poziciji da se dogovaraju o interfejsu za komunikaciju.

Grupa za izradu editora poruka će pokušati da kreira editore sa dobrim znanjem **Message** klase, ali ovo je velika dizajnerska greška. Članovi ove grupe takođe, moraju da se dogovaraju najprostijem interfejsu **Message** klase i, prema tome trebalo bi da znaju sasvim malo o internoj strukturi poruka.

Nadolazeća pitanja vezana za dizajn

U toku razvoja projekta, stalno ćete se suočavati sa osnovnim problemom u dizajniranju: U koju klasu bi trebalo smestiti odredeni set funkcionalnosti, ili informacija. Da li ova funkcija treba da pripada klasi Message, ili klasi Address? Da li editor treba da upisuje ove informacije, ili to treba da radi sama poruka?

Vaša klasa bi trebalo da funkcioniše na principu "samo ono sto je neophodno da zna", slično tajnim agentima. Ona ne bi trebalo da deli ni trunku svoga znanja vise od onoga što je apsolutno neophodno.

Odluke u dizajniranju

Kako budete napredovali sa programom, sve vise ćete se suočavati sa stotinama pitanja o dizajnu. Ona mogu biti rangirana od globalnih pitanja ("šta želimo da ovo radi"), do mnogo odredenijih, ("kako da napravimo da ovo radi").

S obzirom da detalji Vase implementacije neće biti završeni, dok ne isporudte kod, i budući da će se interfejsovi menjati tokom Vašeg rada, morate se uveriti da ste Vaš dizajn dobro razumeli u ranijim procesima. Imperativ je da znate šta pokušavate na napravite, pre nego što napišete kod. Najčešći uzrok kraha softvera je taj, da nije bilo dovoljno argumenata u trenutku kada je on pravljen.

Odluke, odluke

Da biste stekli osećaj šta je proces dizajniranja, prostudirajte pitanje: "Šta će biti u meniju?" Za PostMaster prva stavka će, verovatno, biti "nova e-mail poruka", a to odmah postavlja novo dizajnersko pitanje: "Šta će se dogoditi kada korisnik pritisne New Message?" Da li će se kreirati editor, koji će kreirati e-mail poruku, ili će se prvo kreirati nova e-mail poruka, koja će pozvati editor?

Komanda sa kojom radite je "nova e-mail poruka." Stoga se dni da je kreiranje nove e-mail poruke prvo što treba uraditi. Ali, šta će se dogoditi ako korisnik pritisne Cancel, pošto započne sa pisanjem poruke? Čini se, da bi bilo čistije i jednostavmje prvo kreirati editor i dozvoliti mu da on kreira (i poseduje) novu poruku.

Problem sa ovim pristupom je, da će editor morati da se ponaša drugačije u slučajevima kada se poruka kreira, nego kada se ispravlja postojeća.

Ako se poruka prvi put kreira, ko će je kreirati? Da li će je kreirati kod mem komande? Ako je tako, da li će meni takođe naložiti poruci da se edituje, ili će ovo biti deo konstruktor metode poruke?

Na prvi pogled, čini se da ima smisla da ovo uradi konstuktor; na kraju krajeva, svaki put kada kreirate poruku verovatno ćete želeti i da je editujete. Medutim, ovo nije dobra ideja. Prvo, moguća je sledeće situacija: kreirali ste automatsku poruku, upućenu sistem-operatoru koja se ne edituje. Drugo, još važnije, posao konstruktora je da kreira objekat i ne bi trebalo da bude ni vise ni manje od toga. Jednom kada je kreirana e-mail poruka, posao konstruktora je završen. Dodavanjem poziva za Edit metod, promenili bismo osnovnu ulogu konstuktora i doveli do toga da bi e-mail poruke postale "ranjive" greškama editora.

Što je još gore, Edit metod bi pozivao drugu klasu, editor, što bi dovelo do pozivanja njegovog konstruktora. Pošto editor nije bazna klasa poruke i ne nalazi se unutar poruke, ne bi bilo ispravno da konstrukcija poruke zavisi od uspešnog kreiranja editora.

Na kraju, ne biste želeli da uopšte pozovete editor, ako poruka ne može da bude uspešno kreirana; pogotovo ako uspešno kreiranje poruke iz ovog scenarija zavisi od pozivanja editora. Potrebno je da potpuno izadete iz konstuktora za poruke, pre poziva editora.

РАЗШ

Pofražite objekte koji prirodno nastaju u Vašem dizajnu.

Vršite redizajniranje kako se Vaše razumevanje problemskog prostora povećava.

Nemojte deliri vise informacija izmedu klasa nego što je neophodno.

Potražite mogućnosti koje će Vam omogudti da iskoristiti prednosti polimorfizama u CH- +.

Rad sa drajver programima

Jedan pristup za ispitivanje dizajna je kreiranje drajver programa u ranoj fazi. Na primer, drajver program za PostMaster može da ponudi jednostavan meni, koji će kreirati PostMasterMessage objekte, manipulisati njima i na drugi način ispitivati deo Vašeg dizajna.

^^^Pb *Drajver program* je funkcija koja postoji samo za demonstraciju 1 testiranje drugih funkcija.

U listingu 18.3 prikazana je robusnija definicija klase PostMasterMessage, kao i jednostavan drajver program.

i'"Stjng 18.3: Test program za PostMasterMessage.

```
#include <iostream.h>
#include <string.h>

typedef unsigned long pDate;
enum SERVICE
  { PostMaster, Interchange, CompuServe, Prodigy, AOL, Internet
class String

9       public:
10:        // konstruktori
11:        StringO;
```

nastavlja se

Listing 18.3: Test program za PostMasterMessage. *nastavak*

```
12          String(const char *const);
13          String(const String &);
14          -StringO;
15
16          // preklopljeni operatori
17          char & operatord(int offset);
18          char operator[](int offset) const;
19          String operator+(const String&);
20          void operator+=(const String&);
21          String & operator= (const String &);
22          friend ostream& operator«
23             ( ostream& theStream,String& theString);
24          // Opšte metode pristupa
25          int GetLen()const { return itsLen; }
26          const char * GetStringO const { return itsString; }
27          // static int ConstructorCount;
28       private:
29          String (int);          // privatni konstruktor
30          char * itsString;
31          unsigned short itsLen;
32
33    {
34
35    // podrazumevani konstruktor kreira string od 0 bajtova
36    String: :String()
37    {
38       itsString = new char[l];
39       itsString[0] = '\0';
40       itsLen=0;
41       // cout « "\tDefault string constructors";
42       // ConstructorCount++;
43    }
44
45    // privatni (pomočni) konstruktor, koji se koristi samo od strane
46    // metoda klasa za kreiranje novog stringa
47    // tražene veličine. Puni se sa null.
48    String::String(int  len)
49    {
50       itsString = new char[len+l];
51       for (int i = 0; i<=1en; i++)
52          itsString[l] = '\0';
53       itsLen=len;
54       // cout « "\tString(int) constructors";
55       // ConstructorCount++;
56
57
58    // Konvertuje niz karaktera u String
59    String::String(const char * const cString)
```



```
    itsLen = strlen(cString);
    itsString = new char[itsLen+1];
    for (int i - 0; i<itsLen; i++)
       itsString[i] = cString[i];
    itsString[itsLen]='\0';
    // cout « "\tString(char*) constructors"
    // ConstructorCount++;
}

// konstruktor kopije
String::String (const String & rhs)
{
    itsLen=rhs.GetLen();
    itsString = new char[itsLen+l];
    for (int i = 0; i<itsLen;i++)
       itsString[i] = rhs[i];
    itsString[itsLen] = '\0';
    // cout « "\tString(String&) constructor\n"
    // ConstructorCount++;


// destruktor, oslobada alociranu memoriju
String: :-String()

    delete [] itsString;
    itsLen = 0;
    // cout « "\tString destructor\n";
}

// operator jednako, oslobađa postojeću memoriju
// a onda kopira string i veličinu
String& String::operator=(const String & rhs)
{
    if (this == &rhs)
       return *this;
    delete []itsString;
    itsLen=rhs.GetLen();
    itsString = new char[itsLen+1];
    for (int i = 0; i<itsLen;i++)
       itsString[i] = rhs[i];
    itsString[itsLen] = '\0';
    return *this;
    // cout « "\tString operator=\n";


// nekonstantni operator pomaka, vraća
// referencu na karakter tako da se on može
//  promeniti!
```

nastavljase

nastavak

Listing 18.3: Test program za PostMasterMessage.

```
109: char & String::operatorn(int offset)
110: {
111:     if (offset > itsLen)
112:         return itsString[itsLen-1];
113:     else
114:         return itsString[offset];
115: }
116:
117: // konstantni operator pomaka koji se koristi
118: // za konstantne objekte (pogledati konstruktor kopije!)
119: char String::operator[](int offset) const
120: {
121:     if (offset > itsLen)
122:         return itsString[itsLen-l];
123:     else
124:         return itsString[offset];
125: }
126:
127: // kreira novi string dodajuću tekuči
128: // string rhs stringu
129: String String::operator+(const Strings rhs)
130: {
131:     int totalLen = itsLen + rhs.GetLenQ;
132:     int i,j;
133:     String temp(totalLen);
134:     for ( i = 0; i<itsLen; i++)
135:         temp[i] = itsString[i];
136:     for ( j = 0; j<rhs.GetLen(); j++, i++)
137:         temp[i] = rhs[j];
138:     temp[totalLen]S='\0';
139:     return temp;
140: }
141:
142: void String::operator+=(const String& rhs)
143: {
144:     unsigned short rhsLen = rhs.GetLenQ;
145:     unsigned short totalLen = itsLen + rhsLen;
146:     String temp(totalLen);
147:     for (int i = 0; i<itsLen; i++)
148:         temp[i] = itsString[i];
149:     for (int j = 0; j<rhs.GetLen(); j++, i++)
150:         temp[i] = rhs[i-itsLen];
151:     temp[totalLen]='\0';
152:     *this = temp;
153: }
154:
155: // int String::ConstructorCount = 0;
156:
```


```
157: ostream& operator«( ostreamS theStream,Strings theString)
158: {
159:     theStream « theString.GetStringO;
160:     return theStream;
161:
162:
163: class pAddress
164: {
165: public:
166:     pAddress(SERVICEtheService,
167:              const Strings theAddress,
168:              const Strings theDisplay):
169:       itsService(theService),
170:       itsAddressString(theAddress),
171:       itsDisplayString(theDisplay)
172:       {}
173:     // pAddress(String, String);
174:     // pAddress();
175:     // pAddress (const pAddressS);
176:     -pAddress(){}
177:     friend ostreamS operator«( ostreamS theStream, pAddressS theAddress);
178:     Strings GetDisplayStringO { return itsDisplayString; }
179: private:
180:     SERVICE itsService;
181:     String itsAddressString;
182:     String itsDisplayString;
183:
184:
185: ostreamS operator«( ostreamS theStream, pAddressS theAddress)
186:
187:     theStream « theAddress.GetDisplayString();
188:     return theStream;
189:
190:
191: class PostMasterMessage
192: {
193: public:
194:   // PostMasterMessageO;
195:
196:   PostMasterMessage(constpAddressSSender,
197:                     const pAddressS Recipient,
198:                     const Strings Subject,
199:                     const pDateS creationDate);
200:
201:   // ovde dolaze drugi konstruktori
202:   // ne zaboravite da uključite konstruktor kopijeS
203:   // kao i konstruktor sa skiadišta
204:   // i konstruktor iz žičanog formata
205:   // Takođe uključite konstruktore iz drugih formata
```

nastavlja se

Listing 18.3: Test prograrn za PostMasterMessage^

nastavak

```
206:    ~PostMasterMessage()0
207:
208:    void Edit(); // poziva editor za ovu poruku
209:
210:    pAddressS GetSenderQ const { return itsSender; }
211:    pAddressS GetRecipient() const { return itsRecipient; }
212:    Strings GetSubjectO const { return itsSubject; }
213:    //  void SetSender(pAddress& );
214:    // druge metode pristupa
215:
216:    // ovde dolaze operatori
217:    // i rutine za konverziju za pretvaranje PostMaster poruka
218:    // u poruke drugih formata.
219:
220:  private:
221:     pAddress itsSender;
222:     pAddress itsRecipient;
223:     String   itsSubject;
224:     pDate  itsCreationOate;
225:     pDate itsLastModOate;
226:     pDate  itsReceiptDate;
227:     pDate  itsFirstReadDate;
228:     pDate  itsLastReadDate;
229:
230:
231:  PostMasterMessage::PostMasterMessage(
232:        const pAddressS Sender,
233:        const pAddressS Recipient,
234:        const Strings Subject,
235:        const pDateS creationDate):
236:        itsSender(Sender),
237:        itsRecipient(Recipient),
238:        itsSubject(Subject),
239:        itsCreationDate(creationDate),
240:        itsLastModDate(creationDate),
241:        itsFirstReadDate(O),
242:        itsLastReadDate(O)
243:    {
244:      cout « "Post Master Message created. \n"
245:
246:
247:    void PostMasterMessage::Edit()
248:     {
249:       cout « "PostMasterMessage edit function called\n";
250:     }
251
252
253:  int  main()
254
255:     pAddress Sender(PostMaster, "jliberty@PostMaster", "Jesse Liberty");
256:     pAddress Recipient(PostMaster, "sl@PostMaster","Stacey Liberty");
257:     PostMasterMessage PostMessage(Sender, Recipient, "Saying Hello", 0);
258:     cout « "Message review... \n";
259:     cout « "From:\t\t" « PostMessage.GetSender()  « endl;
260:     cout « "To:\t\t" « PostMessage.GetRecipientO  « endl;
261:     cout « "Subject:\t" « PostMessage.GetSubjectO  « endl;
262:   return 0;
263:  }
```

liiPOZORiwcl Ako dobijete grešku "can't convert*, uklonite ključnu reč const iz linija 210-212.

```
ЈДНЕД^ Post Master Message created.
       Message review...
       From:          Jesse Liberty
       To:            Stacey Liberty
       Subject:       Saying Hello
```

fVMNofcfr U liniji 5 pDate je definisan kao unsigned long. Nije neuobičajeno da se datumi smeštaju kao long integer-i (obično je to broj sekundi koji je protekao od datuma 01.01.1900). U ovom programu time je samo zauzeto mesto; možete očekivati da će se pDate pretvoriti u klasu.

U liniji 5 definisana je konstanta SERVICE, koja treba da omogući Address objektima da prate sopstveni tip adrese, kao što je PostMaster, CompuServe i tako dalje.

Linije 7-161 predstavljaju interfejs i implementaciju Stringa, a to su iste linije kao i one koje ste videli u prethodnim poglavljima. Klasa String je iskorišćena za vise promenljivih cianova u svim Message klasama i u raznim drugim klasama, koje koriste poruke. Celovita i "robusna" String klasa će imati značajnu ulogu u kompletiranju Vaših Message klasa.

U linijama 162-183 deklarisana je klasa pAddress. Ona predstavlja samo osnovnu funkcionalanost ove klase i možete očekivati da će se ona proširiti kada budete bolje razumeli Vaš program. Ovi objekti predstavljaju suštinske komponente svake poruke: adrese pošiljaoca i primaoca. Potpuno funkcionalan pAddress objekat moći će da podržava prosledivanje poruka, odgovora na poruke i tako dalje.

Posao objekta pAddress je da prati prikaz stringa, kao i interno rutiranje stringova u sopstvenim servisima. Jedno pitanje za Vaš dizajn je da li bi trebalo da postoji pAddress objekat, ili bi trebalo da on bude potklasa svakog tipa servisa? Za sada, servis je praćen kao konstanta, koja je promenljiva clan svakog pAddress objekta.

U linijama 191-229 prikazan je interfejs klase PostMasterMessage. U ovom primeru ta klasa je samostalna, ali vrlo brzo želećete da ovaj deo napravite iz hijerarhije nasledivanja. Kada napravite da se ona nasleduje iz Message, neke promenljive članovi će biti prebačeni u baznu klasu, a neke funkcije članovi će izvršiti override metoda bazne klase.

Nekoliko drugih konstruktora pristupnih funkcija i drugih funkcija članova biće neophodni da bi ova klasa postala potpuno funkcionalna. Primetiće da je ono što ovaj listing ilustruje to, da klasa ne mora biti 100 odsto kompletirana, pre nego što napišete jednostavan drajver program za testiranje Vašeg pristupa i dizajna.

U linijama 247-250 funkcija Edi t () je kreirana sa taman toliko detalja koji ukazuju, gde će funkcionalnost biti smeštena kada ova klasa bude u potpunosti operacionalna.

Linije 253-263 predstavljaju drajver program. Trenutno, ovaj program ne radi ništa vise od ispitivanja nekoliko pristupnih funkcija i overload-a operatora«. Bez obzira na to, on Vam pruža polaznu tačku za eksperimentisanje sa PostMasterMessage i okvir, u granicama u kojima možete modifikovati ove klase i ispitivati rezultate tih izmena.

Rezime

Danas ste naučili, kako povezati veliki broj elemenata sintakse C + + i primeniti ih u objektno-orijentisanoj analizi, dizajnu i programiranju. Razvojni ciklus nije dat u linearnoj progresiji, od čiste analize preko dizajna, sa kulminacijom u programiranju; on je, pre bi se moglo reći, cikličan. Prvom fazom je obuhvaćena tipična analiza problema, a rezultati koji su ovom analizom dobijeni predstavljaju osnovu za preliminarni dizajn.

Kada je preliminarni dizajn završen, programiranje može da počne; ali lekcije naučene u fazi programiranja su povratne u analizu i dizajn. I kako programiranje napreduje, kreće testiranje i debagovanje. Ciklus se nastavlja i nikada se stvarno ne završava, čak i kada se dostignu kritične tačke, u kojima je proizvod spreman za isporuku.

Kada neki krupan problem analiziramo iz objektno-orijentisane perspektive, delovi problema koji uzajamno utiču jedni na druge vrlo često predstavljaju objekte preliminarnog dizajna. Dizajner budnim okom prati proces, u nadi da će uspeti da izvrši enkapsulaciju kritičnih dogadaja u objekte, kad god se za to ukaže mogućnost.

Dizajniranje hijerhije klase je obavezno i fundamentalni odnos između delova koji su u interakciji mora da postoji. Za preliminarni dizajn se, naravno, ne misli da je konačan i funkcionalnost će se premeštati sa objekta na objekat, sve dok dizajn dovoljno ne "očvrsne".

Osnovni cilj objektno-orijentisane analize je da sakrije sto vise podataka i onogo što je implementirano i da izgradi objekte koji imaju dobro definisan interfejs. Klijenti koji koriste Vaš objekat nemaju potrebu da znaju detalje implementacije da bi izvršili svoje obaveze.

Pitanja i odgovori

Po čemu se objektno-orijentisana analiza i dizajn, u osnovi, razlikuju od ostalih pristupa?

0 Pre nego što poftiu da se bave razvojem ovih objektno-orijentisanih tehnika, analitičari i programed bi trebalo da programe "razmotre" kao funkcije koje se izvršavaju nad podacima. Objektno-orijentisano programiranje "vidi" integrisane podatke i funkcionalnost kao celine, koje imaju i znanje (podaci) i mogućnosti (funkcije). Proceduralni programi, pak, svu pažnju usmeravaju na funkcije i kako se one izvršavaju nad podacma. I tako su, može se reći, Pascal i C programi kolekcije procedura, a C+ + programi kolekcije klasa.

Da li je konačno objektno-orijentisano programiranje "srebrno tane" **koje će rešiti sve probleme u** programiranju?

0 Ne, ovo nikada nije bio cilj objektno-orijentisanog programiranja. Medutim, kada su u pitanju veliki i složeni problemi, objektno-orijentisana analiza, dizajn i programiranje su u stanju da programera "opskrbe" alatima, kao dragocenu u rešavanju svih tih složenosti i problema, na načine koji su ranije bili potpuno nemogući.

P **Da li je** C+ + savršeni **jezik za objektno**-orijentisano programiranje?

0 C + + ima brojne prednosti i mane, u poredenju sa alternativnim jezicima za objektno-orijentisano programiranje; ali on zato ima jednu ubitačnu prednost, koja je daleko iznad svih ostalih. On je najpopularniji jezik za objektno-orijentisano programiranje na svetu. Iskreno govoreći, većina programera se ne rešava da programira u C + + - u pre iscrpne analize alternativnih objektno-orijentisanih programskih jezika; oni idu tamo gde je akcija; a u 90-tim tu akciju ćete dobiti sa C + +! Za to postoji mnogo dobih razloga; C + + Vam nudi mnogo vise od onoga što se nalazi u ovoj knjizi i ja sam uveren da ćete je pročitati, jer za C+ + kao razvojni jezik se odlučilo zaista mnogo korporacija.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz

1. Kakva je razlika između objektno-orijentisanog programiranja i proceduralnog programiranja?
2. Na šta se odnosi sintagma "event-driven" (dogadajima voden)?
3. Nabrojte faze razvojnog ciklusa?
4. Sta je te korena hijerarhija?
5. Šta je drajver program?
6. Šta je to enkapsulacija?

Vežbe

1. Pretpostavimo da treba da simulirate raskrsnicu Avenije Massachusetts i Pete ulice - dve tipično dvosmerne ulice, na kojima su postavljeni semafori i pešački prelazi. Smisao ove simulacije je da odredite da li vreme na saobraćajnom znaku (semaforu) obezbeduje lagano odvijanje saobraćaja.

   Koje vrste objekata bi trebalo modelirati za potrebe simulacije? Koje bi klase trebalo da postoje u ovoj simulaciji?

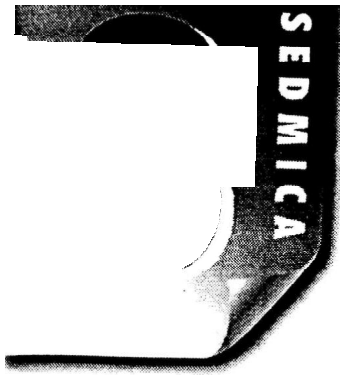
2. Pretpostavimo da se raskrsnica iz Vežbe 1 nalazila u predgradu Bostona, koji, složićete se, ima "najneprijateljskije" ulice u Sjedinjenim Državama. U svako vreme, tim ulicama se kreću tri vrste bostonskih vozača:

   lokalni, koji prolaze kroz raskrsnicu, iako se na semaforu upalilo crveno svetlo, turisti, koji voze lagano i obazrivo (kolima rent-a-car-a, što je vrlo tipično), i taksisti, kod kojih se može zapaziti široka paleta stilova vožnje, u zavisnosti od vrste putnika u vozilu.

   Takođe, Bostonom se kreću dve vrste pešaka: lokalni, koji prelaze ulicu gde god im se svidi i gotovo nikad ne koriste dugme (na semaforu) za prelaz pešaka preko ulice, i turisti, koji uvek koriste pomenuto dugme i prelaze preko ulice samo na zeleno (svetlo). Na kraju, Boston ima i bicikliste koji nikada ne obraćaju pažnju na crveno svetlo. Kako će ovi uslovi uticati na model?

3. Zamoljeni ste da dizajnirate plan rada grupe. Softver Vam dozvoljava (omogućava) da ogranizujete sastanke između pojedinaca, ili grupa i da rezervišete odredeni broj sala za konferencije. Identifikujte glavne podsisteme.

4. Dizajnirajte i prikažite interfejs za klase u delu za rezervaciju soba, iz programa o kome je bilo reči u Vežbi 3.



Templejti

U Danu 17, "Pretprocesor", videli ste kako se koriste makroi za kreiranje različitih lista, uz pomoć operacija za konkatenaciju. Makroi imaju veći broj problema koji su ispravljeni u templejtima.

Danas ćete učiti:

- šta su templejti i kako se koriste
- zašto su templejti bolja alternativa od makroa
- kako se kreiraju templejti klasa
- kako se kreiraju templejti funkcija

Sta su templejti?

Na kraju druge nedelje videli ste kako se pravi PartsLi st objekat i kako se on koristi za kreiranje PartsCataloga. Ako želite da izgradite PartsList objekat, da biste dobili listu mačaka, naići ćete na problem. PartsList poznaje samo delove za kola, ili avione.

Da biste rešili ovaj problem, možete kreirati List baznu klasu i iz nje izvesti PartsList i CatsList klase. Tada možete iskopirati veći deo PartsList klase u deklaraciju nove Cats Li st klase. Sledeće nedelje, kada poželite da napravite listu Car objekata, moraćete da napravite novu klasu i da opet izvršite kopiranje (Cut & Paste) delova deklaracije.

Nije ni potrebno naglašavati da ovo i nije neko zadovoljavajuće rešenje. Vremenom, klasa **List** i njene izvedene klase će biti potrebno proširiti. Provera da li su sve izmene sprovedene u sve odgovarajuće klase može postati noćna mora.

U Danu 17 kratko je demonstriran jedan pristup sa parametarizovanim listama korišćenjem makroa i konkatenacije imena. Iako nam makroi štede vreme koje bi bilo potrebno za kopiranje celih, ili delova deklaracija, oni imaju jedan veliki nedostatak. Kao i sve drugo što se nalazi u pretprocesoru, ni makroi Vam ne mogu garantovati sigumost u radu sa tipovima podataka. Templejti nude bolje metode. Oni su integrisani deo jezika, vode računa o tipovima podataka i vrlo su fleksibilni.

Parametarizovani tipovi

Templejti Vam omogućavaju da naučite kompajler kako da napravi listu stvari bilo kojeg tipa, umesto da kreira set lista specifičnog tipa - PartsList za listu delova, CatList za listu mačaka, itd. Jedini atribut po kome se oni razlikuju je tip stvari u listi. Sa templejtima taj tip postaje parametar za definiciju klase.

Uobičajena komponenta u praktično svim C++ bibliotekama je niz klasa. Kao što ste videli sa Lists, teško je i neefikasno kreirati jedan niz klasa za integer-e, drugi za double integer-e i još jedan za niz životinja. Templejti Vam omogućavaju da deklarišete parametarizovan niz klasa i da, zatim, specificirate koji tip objekata će svaka instanca niza sadržati.

| Д р ^ j *Instantinacija* je akt kreiranja specifičnog tipa iz templejta. Individualne klase se nazivaju instance templejta. *Parametarizovani templejti* Vam omogućavaju da kreirate opštu klasu i da prosledite tipove kao parametre toj klasi, kako biste izgradili specifičnu instancu.

finicija templejta

Deklarišite parametarizovan Array objekat (templejt za niz), tako što ćete napisati

```
1: template <class T>     // deklariše šablon i parametar
2: class Array            // klasa koja se parametarizuje
3: {
4:    public:
5:       ArrayO;
6:    // ovde dolazi puna deklaracija klase
7: };
```

Ključna reč Template se koristi na početku svake deklaracije i definicije templejt klase. Parametri za templejt su smešteni iza ključne reči Template. Parametri su stvari koje će se menjati za svaku instancu. Na primer, u prethodno prikazanom templejt nizu tip objekata koji je smešten u nizu će se menjati. Jedna instanca može da smesti niz integer-a, dok druga može da smesti niz Animal sa.

U ovom primeru korišćena je ključna reč **Class,** koju je sledio identifikator T. Ključna reč **Class** inicira da je taj parametar **tip.** Identifikator T se koristi **u** preostalom delu templejta, kako bi objasnio parametarizovani **tip.** Jedna instanca ove klase će zameniti **Int,** gde god se **T** pojavi, a druga će zameniti **Cat.**

Da biste deklarisali Int, ili Cat instancu parametarizovane klase Array, potrebno je da napišete

```
Array<int> anIntArray;
Array<Cat> aCatArray;
```

Objekat anIntArray je jedan tip *niza integer-a,* objekat aCatArea je *tipa niz mačaka.* Sada možete koristiti tip Array<Int>, gde god biste normalno koristili taj tip - kao vraćenu vrednost iz funkcije, kao parametar za funkciju i tako dalje. U listingu 19.1 prikazane su potpune deklaracije skraćenog Array templejta.

уИАРОМШ^ Listing 19.1 nije kompletan program

Listing **19.1:** Šablon klase Array.

```
1  Listing 19.1 Šablon klase koja predstavlja niz
2      #include <iostream.h>
3      const int DefaultSize = 10;
4
5      template <class T>  // deklarise šablon i parametar
6      class Array            // klasa koja se parametarizuje
7      {
8      publi c:
9:        // konstruktori
10        Array(int itsSize = DefaultSize);
11        Array(const Array &rhs);
12        -ArrayO { delete [] pType; }
13
14        // operatori
15        Array& operator(const ArrayS);
16        T& operator[] (int offSet) { return pType[offSet]; }
17
18        // metode pristupa
19        int getSize() { return itsSize; }
20
21     private:
22        T *pType;
23        int   itsSize;
24
```

JEШ|j> Nema izlaza. Ovo je nepotpun program.

- **ᵘwte*** Definicija templejta počinje u liniji 5 ključnom rečju Tempi ate, koju sledi parametar. U ovom slučaju parametar će biti tip za ključnu reč Cl ass, a identifikator T se koristi za predstavljanje parametarizovanog tipa.

Od linije 6, pa sve do kraja templejta u liniji 24, ostatak deklaracije je sličan bilo kojoj drugoj deklaraciji klase. Jedina razlika je u tome da, gde god bi se normalno pojavio tip objekta, umesto njega stoji identifikator **T.** Na primer, za operator [] bi se očekivalo da vraća referencu na jedan objekat niza, a, u stvari, on je definisan da vraća referencu na T.

Kada je deklarisana instanca integer niza, operator= za taj niz će vratiti referencu na integer. Kada je deklarisana instanca niza Animal, tada će operator= za niz Animal vratiti referencu na Animal.

Koriscenje imena

Unutar deklaracije klase reč `Array` se može koristiti bez dodatnih kvalifikacija. U svim drugim delovima programa ova klase će biti referencirana sa `Array<T>`. Na primer, ako niste napisali konstruktor unutar deklaracije klase, moraćete da napišete

```
template <class T>
Array<T>::Array(int  size):
itsSize = size
{
pType = new T[size];
for (int i = 0; i<size; i++)
pType[i] = 0;
}
```

Deklaracija u prvoj liniji ovoga koda je obavezna, kako bi identifikovala tip (`Class T`). Ime templejta je `Array<T>`, a ime funkcije je `Array (int size)`.

Preostali deo funkcije je isti kao kada bi se koristila funkcija bez templejta. Uobičajen i poželjan metod je da izvršite proveru rada klase i njenih funkcija sa jednostavnim deklaracijama, pre nego što ih uključite u templejte.

Implementacija templejta

Potpuna implementacija Array templejt klase zahteva implementaciju konstruktora, operatora = i tako dalje. Listing 19.2 omogućava jednostavan drajver program za proučavanje ove templejt klase.

^HAPOMEHAy Neki storiji kompajleri ne podržavaju templejte. Templejti su, medutim, deo osnovnog C++ standarda. Svi glavni proizvođači kompajlera su se složili da podržavaju templejte u svojim sledecim verzijama kompajlera, ako to do sada nisu uradili. Ako imate neki stariji kompajler, verovatno nećete biti u mogućnosti da kompajlirate i izvršavate vežbe iz ovog poglavlja. Ipak, pročitajte detaljno celo ovo poglavlje i vratite se na njega kada poboljšate Vaš kompajler.

Listing 19.2: Implementacija šablonskog niza.

```
1    linclude <iostream.h>
2:
3.   const int DefaultSize = 10;
4:
5:   // deklarise se jednostavna klasa Animal tako da možemo
6:   // kreirati niz životinja
7:
8    class Animal
9:   {
10   public:
11      Animal(int);
12      Animal();
13      -Animal()  {}
14      int GetWeightO const { return itsWeight; }
15      void DisplayO const { cout « itsWeight; }
16   pri vate:
17      int itsWeight;
18   };
19
20   Animal::Animal(int   weight):
21   itsWeight(weight)
22
23
24   Animal::Animal ():
25   itsWeight(O)
26   {}
27
28
29   template <class T>  // deklarise šablon i parametar
30   class Array            // klasa koja se parametarizuje
31   {
32   public:
33      //  konstruktori
34      Array(int itsSize = DefaultSize);
35      Array(const Array &rhs);
36:     -ArrayO { delete [] pType; }
37:
38:     //  operatori
39:     Array& operator=(const Array&);
40:     T& operators(int offSet) { return pType[offSet]; }
41:     const T& operator[](int offSet) const
42:         { return pType[offSet]; }
43:     // metode pristupa
44:     int GetSize() const { return itsSize; }
45:
46:  private:
47:     T *pType;
```

nastavlja se

Listing 19.2: Implementacija šablonskog niza. *nastavak*

```
48        int   itsSize;
49
50
51     // implementacije slede...
52
53     // implementira se konstruktor
54     template <class T>
55     Array<T>::Array(int size = DefaultSize):
56     itsSize(size)
57     {
58        pType = new T[size];
59        for (int i = 0; i<size;
60           pType[i] = 0;
61     }
62
63     // konstruktor kopije
64     template <class T>
65     Array<T>::Array(const Array &rhs)
66     {
67        itsSize = rhs.GetSize();
68        pType = new T[itsSize];
69        for (int i = 0; i<itsSize; i++)
70           pType[i] = rhs[i];
71:    }
72
73     // operator=
74     template <class T>
75     Array<T>& Array<T>::operator=(const Array &rhs)
76     {
77        if (this == &rhs)
78           return *this;
79        delete [] pType;
80        itsSize = rhs.GetSize();
81        pType - new T[itsSize];
82        for (int i = 0; i<itsSize; i++)
83           pType[i] = rhs[i];
84        return *this;
85
86
87     // demonstracioni program
88     int main()
89     (
90        Array<int> theArray;        // niz celobrojnih promenljivih
91        Array<Animal> theZoo;       // niz elemenata tipa Animal
92        Animal *pAnimal;
93
94        // popunjavanje nizova
95        for (int 1 = 0; i < theArray.GetSizeO ; i++)
```



```
96:       {                                                    m*F?mv???
97:          theArray[i] = i*2;
98:          pAnimal = new Animal(i*3);
99:          theZoo[i] = *pAnimal;
100:            delete pAnimal;
101:     }
102:     // štampanje sadržaja nizova
103:     for (int j = 0; j < theArray.GetSizeO;
104:     {
105:        cout « "theArray[" « j « "]:\t";
106:        cout « theArray[j] « "\t\t";
107:        cout « "theZoo[" « j « "]:\t";
108:        theZoo[j].DisplayO;
109:        cout « endl;
110:     }
111:
112:     for (int k = 0; k < theArray.GetSizeO; k++)
113:        delete &theZoo[j];
114:  return 0;
115: }
```

theArray[0]	0	theZoo[0]	0
theArray[I]	2	theZoo[I]	3
theArray[2]	4	theZoo[2]	6
theArray[3]	6	theZoo[3]	9
theArray[4]	8	theZoo[4]	12
theArray[5]	10	theZoo[5]	15
theArray[6]	12	theZoo[6]	18
theArray[7]	14	theZoo[7]	21
theArray[8]	16	theZoo[8]	24
theArray[9]	18	theZoo[9]	27

MUllfe* Linije 8-26 prikazuju skraćenu `Animal` klasu, ovde kreiranu, tako da postoje objekti korisnički definsanog tipa koji se dodaju u niz.

Linija 29 deklariše da je ono što sledi templejt, kao i da je parametar za templejt tip koji je dizajniran kao `T`. Klasa `Array` ima dva konstruktora, kao što je i prikazano: prvi, koji uzima veličinu i podrazumevanu vrednost u constant `integer DefaultSize`. Obezbedena je i funkcija pristupa `GetSizeO` koja vraća veličinu niza.

Za bilo koji ozbiljan `Array` program ovo što je ovde prikazano bilo bi neadekvatno. U najmanju ruku, operatori za uklanjanje elemenata, za proširenje niza, za pakovanje niza i tako dalje bili bi neophodni.

Privatni podaci se sastoje od veličine niza i od pointer-a na trenutni (u memoriji) element niza objekata.

Templejt funkcije

Ako želite da prosledite jedan niz objekat funkciji, morate proslediti odredenu instancu niza, a ne templejt. Stoga, ako funkcija SomeFunctionO uzima niz integer-a kao parametar, možete napisati

```
void SomeFunction(Array<int>&);      // uredu
```

ali ne možete napisati

```
void SomeFunction(Array<T>&);  // greška!
```

s obzirom da ne postoji način da se sazna šta je to T&. Takođe ne možete napisati

```
void SomeFunction(Array &);      // greška!
```

pošto ne postoji klasa Array, nego samo templejt i instance.

Deklarisite templejt funkciju.

```
template <class T>
void MyTemplateFunction(Array<T>&);   // uredu
```

Ovde je MyTempl ateFunct i on () funkcija deklarisana kao templejt funkcija u deklaraci-ji u prvom (najvišem) redu. Primetićete da templejt funkcije mogu imati imena, baš kao i sve druge funkcije.

Templejt funkcije mogu za parametre uzimati instance templejta, kao što je dato u sledećem primeru:

```
template <class T>
void MyOtherFunction(Array<T>&, Array<int>&);    // uredu
```

Primetićete da ova funkcija uzima dva niza; parametarizovani niz i niz integera. Prvi može biti niz bilo kakvih objekata, ali je zato drugi uvek niz integer-a.

Templejti i prijatelji

Templejt klase mogu da deklarišu tri tipa prijatelja:

- prijateljske klase, ili funkcije koje nisu templejti
- opšte prijateljske klase, ili funkcije templejti
- prijateljske klase, ili funkcije specifičnog tipa koje su templejti

Prijateljske klase, ili funkcije koje nisu templejti

Moguće je deklarisati da bilo koja klasa, ili funkcija bude prijateljska Vašoj templejt klasi. Svaka instanca klase će tretirati friend osobinu na odgovarajući način, ako je deklaracija o prijateljstvu data u toj odredenoj instanci. Listing 19.3 dodaje trivijalnu prijateljsku funkciju `Intrude()` u templejt definiciju klase `Array`, a drajver program

zatim, poziva Intrude(). Postoje prijateljska, funkcija IntrudeO može da pristupi ««^««^ Private podacima Array-a. Pošto ona nije templejt funkcija, ona može biti pozvana samo sa Array of int.

APOM0tt^ Da biste koristili listing 19.3, kopirajte linije 1-26 iz listinga 19.2 posle linije 1 ovog listinga i, zatim, kopirajte linije 51-86 listinga 19.2 posle linije 37 ovog listinga.

Listing 19.3: Nešablonska prijateljska funkcija.

```
1:    // Listing 19.3 - Klasifikuje odredene prijateljske funkcije
2:
3:     template <class T>  // deklarise Sablon i parametar
4:    class Array              // klasa koja se parametarizuje
5:    (
6:    public:
7:       // konstruktori
8:       Array(int itsSize = DefaultSize);
9:       Array(const Array &rhs);
10:      čArray() { delete [] pType; }
11:
12:      // operatori
13:      Array& operator=(const Array&);
14:      T& operators(int offSet) { return pType[offSet]; }
15:      const T& operators(int offSet) const
16:          { return pType[offSet]; }
17:      // metode pristupa
18:      int GetSizeO const { return itsSize; }
19:
20:     // prijateljska funkcija
21:     friend void Intrude(Array<int>);
22:
23:   private:
24:      T *pType;
25:      int  itsSize;
26:   };
27:
28:     // prijateljska funkcija. Nije šablon, može se koristiti samo
29:     // sa nizovima! Mesa se u privatne podatke.
30:    void Intrude(Array<int> theArray)
31:    {
32:     cout « "\n*** Intrude ***\n";
33:     for (int i = 0; i < theArray.itsSize; i++)
34:        cout « "i: " «     theArray.pType[i] « endl;
35:     cout « "\n";
36:    }
37:
38:   // demonstracioni program
39:   int main()
40:   {
```

nastavlja se

Usting 19.3: Nesoblonska prijotelisko funkcija.

nastavak

```
Array<int> theArray;       // niz celobrojnih promenljivih
Array<Animal> theZoo;      // niz elemenata tipa Animal
Animal *pAnimal;

// puni nizove
for (int i = 0; i < theArray.GetSizeO; i++)
{
   theArray[i] = i*2;
   pAnimal = new Animal(i*3);
   theZoo[i] = *pAnimal;
}

int j, k;
for (j = 0; j < theArray.GetSizeO; j++)
{
   cout « "theZoo[" « j « "]:\t";
   theZoo[j].DisplayO;
   cout « endl;
}
cout « "Now use the friend function to ";
cout « "find the members of Array<int>";
Intrude(theArray);
// vraća alociranu memoriju pre uništavanja nizova.
for (k = 0; k < theArray.GetSizeO; k++)
   delete &theZoo[j];

cout « "\n\nDone.\n";
return 0;
```

```
theZoo[0]:      0
theZoo[I]:      3
theZoo[2]·      6
theZoo[3]       9
theZoo[4]       12
theZoo[5]       15
theZoo[6]       18
theZoo[7]       21
theZoo[8]       24
theZoo[9]       27
Now use the fri end
*** Intrude ***
i: 0
i: 2
i: 4
i: 6
i: 8
i: 10
i: 12
i: 14
I: 16
i: 18

Done.
```

Deklaracija templejta Array je proširena, kako biste uključili prijateljsku funkciju IntrudeO . Ovim ste deklarisali da će svaka instanca niza funkciju IntrudeO smatrati za prijateljsku, zbog čega će IntrudeO imati pristup svim podacima clanovima i funkcijama niza.

U liniji 33 funkcija IntrudeO pristupa promenljivoj itsSize, a u liniji 34 ona takođe direktno pristupa pTypeu. Ovo trivijalno koriscenje nije bilo neophodno, s obzirom da klasa Array obezbeduje javne funkcije pristupa za ove podakte, ali na ovaj način smo demonstrirali kako prijateljske funkcije mogu biti deklarisane sa templejtima.

Opšte prijateljske klase, ili funkcije templejti

Bilo bi vrlo zgodno dodati operator za prikaz klasi Array. Jedan pristup bi bio da se deklariše operator za prikaz za sve moguće tipove Array, ali time bismo izgubli praktičnu stranu korišćenja templejta.

Ono što je potrebno je Insert operator, koji će raditi sa bilo kojim mogućim tipom Array.

```
ostreamS operator« (ostream& Array<T>&);
```

Da bi ovo funkcionisalo, potrebno je da deklarišemo `operator«` kao templejt funkciju.

```
template <class T> ostreamS operator« (ostreamS, Array<T>&)
```

Sada kada je operator« templejt funkcija, potrebno je da samo obezbedite implementaciju. Listing 19.4 prikazuje templejt Array, koji je proširen tako da uključuje ovu deklaraciju i obezbeduje implementaciju za operator«.

^MAPOMEM/t^, Da biste kompajlirali ovaj listing, kopirajte linije 8-26 iz listinga 19.2 i umetnite ih izmedu linija 3 i 4. Takode, kopirajte linije 52-86 iz listinga 19.2 i umetnite ih izmedu linija 37 i 38.

Listing 19.4: Koriscenje operatora ostream.

```
^include <iostream.h>

const int DefaultSize = 10;

template <class T>  // deklariše šablon i parametar
class Array            // klasa koja se parametarizuje
{
public:
   // konstruktori
```

nastavlja se

Smmm Listing 19.4: Koriscenje operotora ostream.

```
10:      Array(int itsSize = DefaultSize);
11:      Array(const Array &rhs);
12:      -ArrayO { delete [] pType; }
13:
14:      // operatori
15:      ArrayS operator=(const ArrayS);
16:      T& operators(int offSet) { return pType[offSet]; }
17:      const T& operator[](int offSet) const
18:       { return pType[offSet]; }
19:      // metode pristupa
20:      int GetSizeO const { return itsSize; }
21:
22:      friend ostream& operator« (ostream&, Array<T>&);
23:
24:   private:
25:      T *pType;
26:      int  itsSize;
27:   };
28:
29:   template <class T>
30:   ostreamS operator« (ostreamS output, Array<T>S theArray)
31:   {
32:      for (int i = 0; i<theArray.GetSizeO; i++)
33:         output « "[" « i « "] " « theArray[i] « endl; return output;
34:   }
35:
36:   enum BOOL { FALSE, TRUE};
37:
38:    int main()
39:   {
40:      B00L Stop = FALSE;       // zastavica za upetljavanje
41:      int offset, value;
42:      Array<int> theArray;
43:
44:      while (!Stop)
45:      {
46:         cout « "Enter an offset (0-9) ";
47:         cout « "and a value. (-1 to stop): " ;
47:         cin » offset » value;
48:
49:         if (offset < 0)
50:            break;
51:
52:         if (offset > 9)
53:         {
54:            cout « M***Please use values between 0 and 9.***\n";
55:            continue;
56:         }
```



```
57:
58:         theArray[offset] = value;
59:      }
60:
61:      cout « "XnHere's the entire array:\n"
62:      cout « theArray « endl;
63:   return 0;
64:   }
```

```
Enter an offset (0-9) and a value. (-1 to stop): 1 10
Enter an offset (0-9) and a value. (-1 to stop): 2 20
Enter an offset (0-9) and a value. (-1 to stop): 3 30
Enter an offset (0-9) and a value. (-1 to stop): 4 40
Enter an offset (0-9) and a value. (-1 to stop): 5 50
Enter an offset (0-9) and a value. (-1 to stop): 6 60
Enter an offset (0-9) and a value. (-1 to stop): 7 70
Enter an offset (0-9) and a value. (-1 to stop): 8 80
Enter an offset (0-9) and a value. (-1 to stop): 9 90
Enter an offset (0-9) and a value. (-1 to stop): IC 10
***Please use values between 0 and g ***
Enter an offset (0-9) and a value. (-1 to stop): -1 -1

Here's the entire array:
[0] 0
[1] 10
[2] 20
[3] 30
[4] 40
[5] 50
[6] 60
[7] 70
[8] 80
[9] 90
```

U liniji 22 funkcija templejt operator«() je deklarisana da bude prijateljska klasi-templejtu Array. Pošto je operator«() implementiran kao templejt funkcija, svaka instanca ovog parametarizovanog niza će automatski imati operator«(). Implementacija za ovaj operator počinje u liniji 29. Svaki clan niza se poziva ponaosob. Ovo će funkcionisati samo ako je definisan operator«() za svaki tip objekta koji je smešten u nizu.

Templejti prijateljskih klasa, ili funkcija specifiaiog tipa

Iako operator za insertovanje, prikazan u listingu 19.4, funkcioniše, on, ipak, nije ono što smo želeli. S obzirom da deklaracija prijateljskog operatora u liniji 29 deklariše templejt, on će raditi sa bilo kojom instancom Array i operator za insertovanje će uzimati niz bilo kojeg tipa.

Templejt operatora za inserovanje, prikazan u listingu 19. 4, će učiniti da sve instance ovog **operatora** « budu prijateljske svim instancama **Array,** bez obzira **da** li je instanca operatora **za** insertovanje **integer, Animal,** ili **Car.** Ovo, medutim, nema smisla. Kako da **insert** operator **za Animal** bude prijatelj sa insert operatorom **za** celobrojni niz?

Ono što je potrebno je, da i **nsert** operator za niz i **nteger-a** bude prijatelj sa **Array** of int klasom i da operator za insertovanje za niz Animals bude prijatelj sa **Array** životinjskom klasom.

Da biste ovo uradili, modifikujte deklaraciju insert operatora u liniji 29 listinga 19.4 i uklonite reči tempi ate<classT>, odnosno, promenite i liniju 30, da biste pročitali

```
friend ostream& operator« (ostream&, Array<T>&);
```

Ovim ćete koristiti tip (T) definisan u templejtu Array. Stoga, operator« za integer-e će raditi samo sa nizom integer-a i tako dalje.

Koriscenje stavki templejta

Stavke tempejta možete posmatrati na isti način kao što biste to učinili i sa bilo kojim drugim tipom. Možete ih proslediti prema referenci, ili prema vrednosti i vratiti ih kao vraćene vrednosti funkcija, takođe prema vrednosti, ili prema referenci. U listingu 19.5 prikazano je kako se prosleduju objekti templejta.

Listing 19.5: Predavanje šablonskih objekata funkcijama i iz funkcija.

```
1:  #include<iostream.h>
2:
3:  const int DefaultSize = 10;
4:
5:  // trivialna klasa za dodavanje nizova
6:  class Animal
7:  {
8:  public:
9:  // konstruktori
10:       Animal(int);
11:      Animal ();
12:      -Animal();
13:
14:      // metode pristupa
15:      int GetWeightO const { return itsWeight; }
16:      void SetWeight(int theWeight) { itsWeight = theWeight; }
17:
18:       // prijateljski operatori
19:      friend ostream& operator<< (ostreamS, const AnimalS);
20:
21:  private:
22:      int itsWeight;
```



```
23:
24:
25:  // operator ekstrakcije za štampanje životinja
26:  ostreamS operator«
27:       (ostream& theStream, const AnimaU theAnimal)
28:  {
29:  theStream « theAnimal.GetWeightO;
30:  return theStream;
31:
32:
33:  Animal::Animal(int weight):
34:   tsWeight(weight)
35:
36:     // cout « "Animal(int)\n";
37:
38:
39:  Animal::Animal():
40:   tsWeight(O)
41:
42:     // cout « "Animal()\n";
43:
44:
45:  Animal::-Animal()
46:
47:   // cout « "Destroyed an animal...\n"
48:
49:
50:  emplate <class T>  // deklariše šablon i parametar
51:  lass Array            // klasa koja se parametarizuje
52:
53:  public:
54:     Array(int itsSize = DefaultSize);
55:     Array(const Array Srhs);
56:     -ArrayO { delete [] pType; }
57:
58:     Array& operator=(const ArrayS);
59:     TS operator[](int offSet) { return pType[offSet]; }
60:     const TS operator[](int offSet) const
61:        { return pType[offSet]; }
62:     int GetSizeO const { return itsSize; }
63
64:   // prijateljska funkcija
65:   friend ostreamS operator« (ostreamS, const Array<T>S)
66:
67:  private:
68:     T *pType;
69:     int  itsSize;
70:
71:
```

nastavlja sc

Listing 19.5: Predavanje šablonskih objekata funkcijama i iz funkcija.

```
70:    template <class T>
72:    ostream& operator« (ostream& output, const Array<T>& theArray)
73:    {
74:       for (int i = 0; i<theArray.GetSize(); i++)
75:          output « "[" « i « "] " « theArray[i] « endl;
76:       return output;
77:    }
78:
79:    void IntFi11Function(Array<int>& theArray);
80:     void AnimalFi11Function(Array<Animal>& theArray);
81:    enum BOOL {FALSE, TRUE};
82:
84:     int main()
85:    {
86:       Array<int>  intArray;
87:       Array<Animal> animalArray;
88:         IntFillFunction(intArray);
87:       AnimalFi11 Function(animal Array);
89:       cout « "intArray..An" « intArray;
90:       cout « "\nanimalArray...\n" « animalArray « endl;
91:       return 0;
92:    }
93:
94:    void IntFi11Function(Array<int>& theArray)
95:    {
96:       BOOL Stop = FALSE;
97:       int offset,  value;
98:       while  (!Stop)
99:       {
100:          cout « "Enter an offset (0-9) ";
101:          cout « "and a value. (-1 to stop): " ;
102:          cin » offset » value;
103:         if (offset < 0)
104:            break;
105:         if (offset > 9)
106:         {
107:            cout « "***Please use values between 0 and 9.***\n";
108:            continue;
109:         }
110:         theArray[offset]  = value;
111:      }
112:   }
113:
114:
115:    void AnimalFi11Function(Array<Animal>& theArray)
116:   {
117:      Animal * pAnimal;
118:      for (int i = 0; i<theArray.GetSizeO; i++)
119:
120:         pAnimal = new Animal;
121:         pAnimal->SetWeight(i*100);
122:         theArray[i] = *pAnimal;
123:         delete pAnimal;   // kopija je stavljena u niz
124:
125:
```

```
Enter an offset (0-9) and a value.  -1 to stop): 1 10
Enter an offset (0-9) and a value.  -1 to stop): 2 20
Enter an offset (0-9) and a value.  •1 to stop):  30
Enter an offset (0-9) and a value.  •1 to stop):  40
Enter an offset (0-9) and a value.  •1 to stop):  50
Enter an offset (0-9) and a value.  -1 to stop):  60
Enter an offset (0-9) and a value.  •1 to stop): 7 70
Enter an offset (0-9) and a value.  •1 to stop): 8 80
Enter an offset (0-9) and a value.  •1 to stop): 9 90
Enter an offset (0-9) and a value,  •1 to stop): 10 10
***Please use values between 0 and 9
Enter an offset (0-9) and a value. ( 1 to stop): -1 -1

intArray:...
[0]
[1]
[2] 20
[3] 30
[4] 40
[5] 50
[6] 60
[7] 70
[8] 80
[9] 90

animalArray:...
[0] 0
[1] 100
[2] 200
[3] 300
[4] 400
[5] 500
[6] 600
[7] 700
[8] 800
[9] 900
```

• MO/,^, Vcći deo implementacije klase `Array` je izostavljen, radi uštede prostora. `Animal` klasa je deklarisana u linijama 6-23. I mada je "skraćena" i pojednostavljena, ona obezbeduje insertovanje sopstvenog `operatora` («), kako bi omogućila prikazivanje klase `Animals`. U ovom izlazu prikazana je "tekuća" (sadašnja) težina `Animal - a`.

« 2 * ^

Primetićete da Animal ima podrazumevani konstruktor. Ovo je neophodno zato što kada neki objekat dodate u niz podrazumevani konstruktor objekta će biti upotrebljen za kreiranje objekta. Ovo stvara neke teškoće, kao što ćete i sami mod da vidite.

U liniji 79 deklarisana je funkcija IntFi ll Function(). Prototip ukazuje na to da ona uzima celobrojini niz. Primetićete da ovo nije templejt funkcija. IntFi HFunction() ocekuje samo jedan tip niza, tj. celobrojni niz. Na sličan način u liniji 80 deklarisana je funkcija AnimalFillFunction(), koja za parameter uzima Array of Animal.

Implementacija ovih funkcija se razlikuje, s obzirom da se popunjavanje niza i nteger-a ne izvršava na isti način kao i popunjavanje niza Animal sa.

Specijalizovane funkcije

Ako skinete komentar sa print naredbi u `Animal` konstruktoru i destruktoru u listingu 19.5, videćete da postoje dodatni konstuktori i destruktori za `Animals`. Kada je jedan objekat dodat nizu, pozvan je podrazumevani konstruktor za taj objekat. Medutim, `Array` konstruktor dodeljuje vrednost 0 svakom članu niza, kao što je prikazano u linijama 59 i 60 listinga 19.2.

Kada napišete `someAnimal = (Animal) 0;`, Vi pozivate podrazumevani `operator =` za `Animal`. Ovo podrazumeva kreiranje privremenog objekta `Animal`, uz koriscenje konstruktora, koji uzima `integer` (nula). Taj privremeni objekat se koristi kao desna strana za `operator =` i zatim se uništava.

Ovo je prekomerno trošenje vremena, s obzirom da je objekat `Animal` već propisno inicijalizovan. Medutim, Vi ne možete ukloniti ovu liniju, s obzirom da integer-i nisu automatski inicijalizovani na vrednost 0. Rešenje je da naučite templejt kako da ne koristi ovaj konstruktor za `Animals`, nego da koristi specijalni `Animal` konstuktor.

Možete napraviti eksplicitnu implementaciju `Animal` klase, kao ito je prikazano u listingu 19.6.

Listing 19.6: Specijalizovane šablonske implementacije.

```
1:    linclude <iostream.h>
2:
3:    const int DefaultSize = 3;
4:
5:    // trivijalna klasa za dodavanje nizovima
6:      class Animal
7:      {
8:      public:
9:         II konstruktori
10:        Animal(int);
11:        Animal();
12:        -Animal();
13:
```



```
      // metode pristupa
      int GetWeightO const { return itsWeight; }
      void SetWeight(int theWeight) { itsWeight = theWeight;

      // prijateljski operatori
      friend ostreamS operator« (ostreamS, const AnimalS);

  private:
     int itsWeight;


   II operator ekstrakcije za štampanje životinja
  ostreamS operator«
       (ostreamS theStream, const AnimalS theAnimal)
  {
    theStream « theAnimal.GetWeightO;
    return theStream;


  Animal::Animal(int weight):
  itsWeight(weight)
  {
     cout « "animal(int) ";


  Animal : .-Animal ():
  i tsWeight(0)
  {
     cout « "animal()


  Animal:;-Animal ()

    cout « "Destroyed an animal.


template <ciass T> /: Ceklariše šablon i parametar
class Array          I; klasa koja se parametarizuje
{
public:
   Array(int itsSize = DefaultSize);
   Array(const Array &rhs);
   -ArrayO { delete [] pType; )

   // operatori
   ArrayS operator=(const ArrayS);
   T& operator[] (int offSet) { return pType[offSet]; }
   const TS operator[](int offSet) const
      { return oType[offSet]; }
```

nastavlja sc

g£^I listing 19.6: Specijaliiovane sablonske implementacije. *nastavak*

```
62:
63:       // metode pristupa
64:       int GetSizeO const { return itsSize; }
65:
66:       // prijateljska funkcija
67:       friend ostream* operator« (ostreami, const Array<T>&);
68:
69:    private:
70:       T *pType;
71:       int   itsSize;
72:    };
73:
74:    template <class T>
75:    Array<T>::Array(int size = DefaultSize):
76:    itsSize(size)
77:    {
78:       pType = new T[size];
79:       for (int i = 0; i<size; i++)
80:          pType[i] = (T)0;
81:    }
82:
83:    template <class T>
84:    Array<T>& Array<T>::operator=(const Array &rhs)
85:    (
86:       if (this == &rhs)
87:          return *this;
88:       delete [] pType;
89:       itsSize = rhs.GetSizeO;
90:       pType = new T[itsSize];
91:       for (int i = 0; i<itsSize; i++)
92:          pType[i] = rhs[i];
93:       return *this;
94:    }
95:    template <class T>
96:    Array<T>::Array(const Array &rhs)
97:    {
98:       itsSize = rhs.GetSizeO;
99:       pType = new T[itsSize];
100:      for (int i = 0; i<itsSize; i++)
101:         pType[i] = rhs[i];
102:   }
103:
104:
105:   template <class T>
106:    ostream& operator« (ostream& output, const Array<T>& theArray)
107:   {
108:       for (int i = 0; i<theArray.GetSize(); i++)
109:          output « "[" « i « "] " « theArray[i] « endl;
```



```
110:      return output;
111:   }
112:
113:
114:   Array<Animal>::Array(int AnimalArraySize):
115:   itsSize(AnimalArraySize)
116:   {
117:      pType = new Animal[AnimalArraySize];
118:   }
119:
120:
121:    void IntFi11Function(Array<int>& theArray);
122:    void AnimalFi11 Function(Array<Animal>& theArray);
123:   enum BOOL {FALSE, TRUE};
124:
125:    int main()
126:   {
127:      Array<int>  intArray;
128:      Array<Animal> animalArray;
129:        IntFillFunction(intArray);
130:       Animal Fill Function(animal Array);
131:       cout « "intArray..An" « intArray;
132:      cout « "\nanimalArray..An" « animalArray « endl;
133:     return 0;
134:   }
135:
136:    void IntFi11Function(Array<int>& theArray)
137:   {
138:      BOOL Stop = FALSE;
139:      int offset, value;
140:      while (!Stop)
141:      {
142:         cout « "Enter an offset (0-9) and a value. ";
143:         cout « "(-1 to Stop): " ;
143:         cin » offset » value;
144:         if (offset < 0)
145:            break;
146:         if (offset > 9)
147:         {
148:            cout « "***Please use values between 0 and 9.***\n"
149:            continue;
150:         }
151:         theArray[offset] = value;
152:      }
153:   }
154:
155:
156:    void AnimalFi11Function(Array<Animal>& theArray)
157:   {
```

nastavljase

Listing 19.6: Specijalizovone Sablonske implementacije.

nastavak

```
158:      Animal * pAnimal;
159:      for (int i = 0; i<theArray.GetSize(); i++)
160:
161:         pAnimal = new Animal(i*10);
162:         theArray[i] = *pAnimal;
163:         delete pAnimal;
164:
165:
```

ШРШШр Redni brojevi su dodati u izlaz, kako bi analiza bila jednostavnija. Ti redni brojevi se neće pojaviti u Vasem izlazu.

```
    animal() animal() animal() Enter an offset (0-9) and a value. (-1 to
    stop): 0 0
    Enter an offset (0-9) and a value. (-1 to stop): 1 1
    Enter an offset (0-9) and a value. (-1 to stop): 2 2
    Enter an offset (0-9) and a value. (-1 to stop): 3 3
    Enter an offset (0-9) and a value. (-1 to stop): -1 -1
    animal(int) Destroyed an animal...animal(int) Destroyed an
    animal. .animal(int) Destroyed an animal...initArray...
    [0] 0
    [1] 1
9:  [2] 2
10:
11: animal array...
12: [0] 0
13: [1] 10
14: [2] 20
15:
16: Destroyed an animal...Destroyed an animal...Destroyed an animal.
17:
«< Second run »>
18: animal(int) Destroyed an animal...
19: animal(int) Destroyed an animal...
20: animal(int) Destroyed an animal...
21: Enter an offset (0-9) and a value. (-1 to stop): 0 0
22: Enter an offset (0-9) and a value. (-1 to stop): 1 1
23: Enter an offset (0-9) and a value. (-1 to stop): 2 2
24: Enter an offset (0-9) and a value. (-1 to stop): 3 3
25: animal(int)
26: Destroyed an animal...
27: ammal (int)
28: Destroyed an animal...
29: animal(int)
30: Destroyed an animal...
31: initArray...
32: [0] 0
33: [1] 1
```



```
35:
36: animal array...
37: [0] 0
38: [1] 10
39: [2] 20
40:
41: Destroyed an animal...
42: Destroyed an animal...
43: Destroyed an animal...
```

дн|M:jp>, Listing 19.6 proizvodi obe klase u celini, tako da možete da vidite kreiranje i destrukciju privremenog `Animal` objekta. Vrednost za `DefaultSize` je redukovana na 3, kako bi se pojednostavio izlaz. Konstruktori i destruktori za `Animal`, koji se nalaze u linijama 33-48, prikazuju red koji indicira trenutak kada su oni pozvani.

U linijama 74-81 deklarisano je ponašanje templejta za `Array` konstruktor. U linijama 114–118 demonstriran je specijalni konstruktor za `Array of Animals`. Primetićete da je u ovom specijalnom konstruktoru podrazumevanom konstruktoru dozvoljeno da postavi inicijalne vrednosti za svaki `Animal`, ali da to nije uredeno.

Prvi put kada startujete ovaj program prikazaće se i prvi izlaz. Linija 1 u izlazu prikazuje poziv tri podrazumevana konstruktora za kreiranje niza. Korisnik je uneo četiri broja u celobrojni niz.

Izvršenje prelazi na `Animal Fi 11 Function()`. Ovde je u liniji 161 kreiran privremeni `Animal` objekat na "steku" i njegova vrednost je iskorišćena da bi se modifikovao `Animal` objekat iz niza u liniji 162. U liniji 163 uništen je privremeni objekat `Animal`. Ovo je ponovljeno za sve clanove niza, kao što se vidi u izlazu,u liniji 6.

Na kraju programa uništeni su i nizovi, i kada su pozvani njihovi destruktori, takođe su unisteni i svi njihovi objekti. Ovo je prikazano u izlazu, u liniji 16.

U drugom izlazu (linije 18-43) skinut je komentar sa spedjalne implementacije konstuktora za niz karaktera iz linija 114–118. Kada je program ponovo startovan, startovan je i `templejt` konstruktor, prikazan u linijama 74-81, po konstruisanju niza `Animal`.

Ovo je prouzrokovalo da privremeni objekti `Animal` budu pozvani za svakog člana niza[1] \ linijama 79 i 80 u programu i to je prikazano u izlazu, u linijama 18-20.

U svim ostalim aspektima izlaz za dva izvršenja programa je identičan, kao što se i moglo očekivati.

Statičkl članovi i templejti

Templejti mogu da deklarišu statičke podatke članove. Svaka instantinacija templejta tada ima sopstvene statičke podatke. i to po jedan za svaki tip klase. Stoga, ako

dodate statički clan u klasu Array (na primer, brojač kreiranih nizova), imaćete po jedan ovakav Clan za svaki tip: jedan za niz Animal s i jedan za sve nizove i nteger-a. Listing 19.7 dodaje statičke članove i statičku funkciju u klasu Array.

Listing 19J: Koriscenje stotickih podataka cianova i funkcija clanica sa iablonima.

```
linclude <iostream.h>

template <class T>  // deklarise Sablon i parametar
class Array            // klasa koja se parametarizuje
{
publ ic:
   // konstruktori
   Array(int itsSize = DefaultSize);
   Array(const Array &rhs);

   -ArrayO { delete [] pType;   itsNumberArrays-;  }

   // operatori
   Array& operator=(const Array&);
   T& operators(int offSet) { return pType[offSet]; }
   const T& operators(int offSet) const
      { return pType[offSet]; }
   // metode pristupa
   int GetSizeO const ( return itsSize; }
   static int GetNumberArraysO { return itsNumberArrays; }

   // prijateljska funkcija
  friend ostream& operator« (ostream&, const Array<T>&);

private:
   T *pType;
   int   itsSize;
   static int itsNumberArrays;
};

template <class T>
   int Array<T>::itsNumberArrays = 0;

template <class T>
Array<T>::Array(int size = DefaultSize):
itsSize(size)
{
   pType = new T[size];
   for (int i = 0; i<size; i++)
     pType[i] = (T)0;
   itsNumberArrays++;
}

template <class T>
Array<T>& Array<T>::operator=(const Array &rhs)

   if (this == &rhs)
      return *this;
   delete П pType;
   itsSize = rhs.GetSizeO;
   pType = new TfitsSize];
   for (int i = 0; i<itsSize; i++)
      pType[i] = rhs[i];
}

template <class T>
Array<T>::Array(const Array &rhs)
{
   itsSize = rhs.GetSizeO;
   pType = new T[itsSize];
   for (int i = 0; i<itsSize; i++)
      pType[i] = rhs[i];
   itsNumberArrays++;
}

template <class T>
ostream& operator« (ostreami output, const Array<T>& theArray)

   for (int i = 0; i<theArray.GetSizeO; i++)
      output « "[" « i « "] - « theArray[i] « endl;
   return output;



Array<Animal>::Array(int  AnimalArraySize):
itsSize(AnimalArraySize)
{
   pType = new T[Anima1ArraySize];
   itsNumberArrays++;


int main()


   cout « Array<int>::GetNumberArrays() « " integer arrays\n"
   cout « Array<Animal>::GetNumberArrays();
   cout « " animal arrays\n\n";
   Array<int> intArray;
   Array<Animal> animalArray;

   cout « intArray.GetNumberArraysO « integer arrays\n"
   cout « animalArray.GetNumberArraysO;
   cout « " animal arrays\n\n";
```

nastavlja se

Listing 19.7: Koriscenje statičkih podotoko članova i funkcijq clanka sa šablonima. *nastavak*

```
93
94        Array<int> *pIntArray = new Array<int>;
95
96        cout « Array<int>:-.GetNumberArraysO « " integer arrays\n";
97        cout « Array<Animal>::GetNumberArraysO;
98        cout « " animal arrays\n\n";
98
99        delete pIntArray;
100
101       cout « Array<int>::GetNumberArrays() « " integer arraysV;
102       cout « Array<Animal>:-.GetNumberArraysO;
103       cout « " animal arrays\n\n";
103      return 0;
104

          0 integer arrays
          0 animal arrays

          1 integer arrays
          1 animal arrays

          2 integer arrays
          1 animal arrays

          1 integer arrays
          1 animal arrays
```

WWpfe> Deklaracija klase Animal je izostavljena, radi uštede prostora. Klasi '* ™ Array je pridodata statička promenljiva itsNumberArrays u liniji 27 i pošto je ovaj podatak privatan, u liniji 19 dodata je statička funkcija pristupa GetNumberArraysO.

Inicijalizacija statičkih podataka je završena punom templejt kvalifikacijom u linijama 30 i 31. Konstruktori za Array i destruktor su modifikovani tako da prate broj nizova koji postoje u bilo kom trenutku.

Pristup statičkim clanovima je isti kao i pristup statičkim clanovima bilo koje klase. To možete učiniti u pomoć postojećeg objekta, kao što je prikazano u linijama 91 i 92, ili korišćenjem potpune specifikacije klase, kao što je prikazano u linijama 85 i 86. Primetićete da morate da koristite specifičan tip niza kada pristupate statičkim podacima. Postoji po jedna promenljiva za svaki tip.

<j| **РАЛП** *И* Koristite staticke promenljive clanove sa templejtima, ako Vam je to potrebno.

Specijalizujte ponašanje templejta, tako što ćete uraditi override templejt funkcija **po** tipu.

Koristite parametre za templejt funkcije, kako biste ograničili njihove instance samo na odgovarajuće tipove.

Standardna templejt biblioteka

Novi razvojni tok u C++-u je usvojen iz Standard Template Library (STL). Svi veći proizvođači kompajlera sada nude STL, kao deo sopstvenog proizvoda. STL je biblioteka kontejner klasa, baziranih na templejtima, uključujući vektore, liste, upite i stekove. U STL je, takode, uključen i veći broj opštih algoritama, kao što su algoritmi za sortiranje i pretraživanje. Cilj STL-a je da Vam ponudi alternativu za "izmišljanje rupe na saksiji" za ove opšte zahteve. STL je testirana i debagovana, nudi visoke performanse, a, što je najlepše, ona je besplatna. Jedna od najvažnijih stvari je da, kada jednom razumete kako se korisiti STL kontejner, možete ga korisiti u svim Vašim programima, bez ponovnog "izmišljanja".

Rezime

Danas ste naučili kako da kreirate i koristite templejte. Templejti su ugradena mogućnost C++ i koriste se za kreiranje parametarizovanih tipova - tipova koji menjaju svoje ponašanje, u zavisnosti od parametara koji su im prosledeni pri kreiranju. Oni su način na koji sigurno i efikasno možete ponovo koristiti kod.

Definiciju templejta odreduje parametarizovani tip Svaka instanca templejta je jedan stvarni objekat, koji se može koristiti kao bilo koji drugi objekat, tj. kao parametar za funkciju, kao vraćena vrednost i tako dalje.

Templejt klase mogu da deklarišu tri tipa prijateljskih funkcija: netemplejtske, opšte templejtske i templejtske sa specifičnim tipom. Templejti mogu da deklarišu statičke podatke članove i u toj situaciji svaka instanca templejta ima sopstveni set statičkih podataka.

Ako imate potrebu da specijalizujete ponašanje nekih templejt funkcija koje su bazirane na aktueinom tipu, možete izvršiti override templejt funkcije sa odredenim tipom. Ovo takođe funkcioniše i sa funkcijama clanovima.

Pitanja i odgovori

P Zašto koristiti templejte, kada postoje makroi?

O Templejti vode računa o tipovima i ugradeni su u jezik.

P Koja je razlika između parametarizovanog tipa `templejt` funkcije i parametara normalne funkcije?

O Obične funkcije (one koje nisu templejt) preuzimaju parametre kojima mogu izvršiti neku akciju. `Teplejt` funkcije Vam omogućavaju da parametarizujete tip odredcnog parametra funkcije. Stoga, funkciji možete proslediti *Array of Type* i, zatim, dobiti `Type` odreden templejt instancom.

P Kada koristiti templejte, a kada nasledivanje?

0 Koristite templejte kada **je** kompletno ponašanje, ili uglavnom kompletno ponašanje nepromenjeno, osim kada **je u** pitanju tip stvari **u** Vašoj klasi. Ako uhvatite samog sebe da kopirate klasu, a menjate samo tip jednog, ili vise njenih cianova, verovatno bi trebalo da razmislite o korišćenju templejta.

P Kada ćete koristiti opšte prijateljske templejt klase?

O Kada svaka instanca, bez obzira na tip, treba da bude prijateljska toj klasi, ili funkciji.

P Kada ćete koristiti templejte specifičnog tipa za prijateljske klase, ili funkcije?

O Kada želite da ostvarite jedan-prema-jedan relaciju između dve klase. Na primer, array<int> mora da odgovaraiterator<int>, ali nei iterator< Animal>.

Radionica

Nudimo Vam test, koji treba da Vam pomogne da utvrdite svoje razumevanje pređenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da razumete odgovore, pre nego što predete na sledeće poglavlje.

Kviz pitanja

1. Koja je razlika između templejta i makroa?
2. Koja je razlika između parametara u templejtu i parametara u funkcijama?
3. Koja je razlika između templejta specifičnog tipa prijateljskih klasa i opšteg templejta prijateljske klase?
4. Da li je moguće obezbediti specijalno ponašanje za jednu instancu templejta, ali ne i za ostale?
5. Koliko statičkih promenljivih će biti kreirano, ako stavite jednog statičkog člana u definiciju templejt klase?

Vezbe

1. Kreirajte templejt na osnovu sledeće List klase:

```
class List
{
pri vate:
public:
     List():head(0),tail(0),theCount(0) {}
     virtual -List();
     void insert( int value );
     void append( int value );
     int is_present( int value ) const;
     int is_empty() const { return head ·* 0; }
     int count() const { return theCount; }
private:
     class ListCell
     {
     public:
          ListCell(int value, ListCell *cell =
) :val(value),next(cell){}
          int val;
          ListCell *next;
     };
     ListCell *head;
     ListCell *tail;
     int theCount;
};
```


2. Napišite implementaciju za List klasu (netemplejt).
3. Napišite templejt verziju implementacije.
4. Deklarišite tri objekta za listanje: list of Strings, list of Cats i list of ints.
5. Lovci na greške: šta nije dobro u sledećem kodu: (imajte u vidu da je List tempiejt definisan, a da je Cat klasa koja je već ranije definisana u knjizi).

```
List<Cat> Cat_List;
Cat Felix;
CatList.append( Felix );
cout « "Felix is " «
    ( Cat__List.is_present( Felix ) ) ? "" : "not " « "present\n";
```

SAVET: (Ovo je teško): šta čini Cat različitim od int?

6. Deklarišite prijateljski operator = = za List.
7. Implementirajte prijateljski operator = = za List.
8. Da li operator = = ima isti problem kao i u Vežbi 5?
9. Implementirajte templejt funkciju za swap, koja vrši zamenu vrednosti dve promenljive.

Dan 20

Izuzeci i obrada grešaka

Kod kojih ste do sada uspeli da upoznate u ovoj knjizi napisan je za ilustraciju teme koja se obraduje. U njega nisu uključeni delovi koji se bave obradom grešaka, kako Vam ne bi skretali pažnju od glavne teme. Medutim, programi u realnom svetu moraju u sebi da sadrže i obradu grešaka.

Danas ćete naučiti:

- šta su izuzeci
- kako se izuzeci koriste
- kako napraviti hijerarhiju izuzetaka
- kako se izuzeci uklapaju u sveukupni pristup obradi grešaka
- šta je debager

Bagovi i greške, promašaji i Code Rot

Svi programi imaju bagove. Što je program ved, to ima i vise bagova. Veći broj tih bagova prolazi u finalnu verziju softvera. To što je ova činjenica tačna, ne znači da je to i ispravno i pravljenje robusnih programa koji nemaju bagove je prioritetni zadatak bilo kojeg ozbiljnog programera.

Najveći problem softverske industrije je "bagovit", nestabilan kod. Najveći troškovi nastaju upravo u naporima za testiranje i ispravljanje programa. Osoba koja bi rešila problem proizvodnje dobrih, solidnih, praktično neprobojnih programa, a za nisku cenu, za kratko vreme uvela bi revoluciju u softversku industriju.

Postoji veći broj različitih tipova bagova koji mogu ugroziti rad programa. Prvi je prosta logika: program radi ono što ste tražili, ali algoritam nije ispravan. Drugi je u sintaksi: upotrebili ste pogrešan izraz, funkciju, ili strukturu. Ova dva problema se najčešće sreću i, stoga, programed na njih i najviše obraćaju pažnju.

Istraživanja i iskustva iz realnog sveta su pokazala da što kasnije u procesu razvoja pronađete problem vise će Vas koštati njegova ispravka. "Najjeftiniji" problemi, ili bagovi su oni koje znate da izbegnete da napravite. Sledeći jeftini bagovi su oni koje presreće kompajler. Standardi C + + primoravaju kompajlere da iskoriste što je moguće vise energije, kako bi otkrili što vise bagova u fazi kompilacije.

Bagovi koji su prošli kompilaciju, ali koje ne uspevate da uhvatite u prvom testu, tj. oni koji dovode do neispravnog prekida programa svaki put, su jeftiniji za pronalaženje i ispravku od onih koji se pojavljuju sporadično.

Veći problem od logičkih i sintaksnih bagova je nepotrebna "lomljivost" programa: Vaš program funkcioniše sasvim dobro kada korisnik unese broj za koji je upitan, ali program pada kada korisnik unese slovo. Drugi programi padaju pri prekoračenju memorije, ili ako vratanca disk-drajva nisu zatvorena, ili ako je modem izgubio vezu.

Da bi se izborili sa ovakvom vrstom pada programa, programed pokušavaju da svoje programe učine "neprobojnim za metke". Program "neprobojan za metke" je onaj koji može da obradi svaku situaciju koja se može desiti u fazi izvršavanja, od bizarnog korisničkog unosa, do prekoračenja memorije.

Važno je napraviti razliku između bagova koji nastaju kada programer napravi grešku u sintaksi, logičkih grešaka koje nastaju kada programer nije razumeo problem, ili nije znao da ga reši i izuzetaka koji nastaju usled neobičnih, ali predvidivih problema, kao što je, na primer, prekoračenje resursa (memorije, ili prostora na disku).

Izuzeci

Programeri koriste snažne kompajlere i filuju svoj kod naredbom asserts, o kojoj je diskutovano u Danu 17, "Pretprocesor", kako bi uhvatitli greške u programiranju. Oni, takođe, stalno ispituju dizajn i vrše iscrpljujuća testiranja, kako bi pronašli logičke greške.

Izuzeci su, međutim, drugadji. Izuzetne situacije ne možete eliminisati; za njih se samo možete pripremiti. Vaši korisnici će ostati bez memorije, s vremena na vreme; a šta ćete Vi uraditi? Vaš izbor se svodi na sledeće:

- program će pasti
- informisaćete korisnika i napustiti program
- informisaćete korisnika i omogućićete mu da pokuša da ispravi grešku i da nastavi sa radom
- izvešćete korektivne akcije i nastaviti, bez obaveštavanja korisnika.

obezbeđuje integrisane metode za obradu predvidivih gresaka, ah koje se retko javljaju u fazi izvršenja programa. Predvidivih

Reč-dve o Code Rot

Code Rot se dobro pokazao u praksi. Perfektan, dobro napisan i potpuno debagovan kod proizvodi novo ponašanje šest meseci nakon instalacije i ne postoji neki sjajan nadn da ga zaustavite. Ono što možete da uradite je, naravno, da napišete Vaš program tako da, kada Vam se vrati na ispravku, uspete da na brz nadn otkrijete u čemu je problem.

^jMAPOMIHA^ Code Rot je vrsta programerskog vica, koji objašnjava kako kod koji nema bagove iznenada postaje nepouzdan. On nas, medutim, uči važnoj lekciji: programi su enormno kompleksni i bagovi i greške mogu biti sakriveni dugo vremena, pre nego što se pojave. Zaštitite se tako što ćete napisati kod koji je jednostavan za održavanje.

Ovo znad da Vaš kod mora biti dobro iskomentarisan, čak i ako očekujete da ga niko drugi nikada vise neće ni pogledati. Šest meseci nakon što isporučite Vaš kod čitaćete ga kao da ga nikad niste videli, uvereni da niko nikada nije mogao da napiše tako komplikovan i nedtljiv kod, a da očekuje od njega bilo šta drugo, osim katastrofe!

Izuzeci

U C + + -u izuzetak je objekat prosleden iz zone koda u kojoj se problem desio u deo koda koji će pokušati da reši problem. Tipovi izuzetaka odreduju koja zona koda će podržati problem, i sadržaj objekta će biti prosleden, a ako postoji, može se iskoristiti za obaveštavanje korisnika.

Osnovne ideje za izuzetke su:

- stvarna lokacija resursa (na primer, alokacija memorije, ili zaključavanje datoteke) je, obično, izvršena na vrlo niskom nivou u programu;
- logika, šta uraditi kada operacija, ili ne uspe, ili ne bude dovoljno memorije za alokaciju i kada datoteka ne može biti zaključana, se, obično, nalazi visoko u programu, unutar koda za interakciju sa korisnikom;
- izuzeci obezbeduju ekspresnu putanju, od koda koji alocira resurse, do koda koji obrađuje greške. Ako tu postoji funkcija za intervenciju, ona će omogudti čišćenje memorijskih lokacija, ali od njih neće biti zahtevano da sadrže kod, dja je jedina uloga da prosledi grešku.

Kako se izuzeci koriste?

**[°l2ll:* SCkreir3JU ^ blZ30krUŽl11Zonek0da *k* ^ ™ g u imati problem.

try blocks are created to surround areas of code that may have a problem. For example:

```
try
{
SomeDangerousFunction();
}
```

catch blocks handle the exceptions thrown in the try block. For example:

```
try
{
SomeDangerousFunctionO;
}
catch(OutOfMemory)
{
// preduzima neku akciju
}
catch(FileNotFound)
{
// preduzima neku akciju
}
```

Osnovni koraci u korišćenju izuzetaka su:

1. Identifikujte zone programa, gde će biti započeta operacija koja može da dovede do izuzetka, i smestite ih u try blokove.
2. Kreirajte catch blokove, kako biste uhvatili izuzetke i ocistili alociranu memoriju, kao i informisali korisnika. U listingu 20.1 ilustrovano je koriscenje try i catch blokova.

:m *ΦЖJJ Izuzeci* su objekti koji se koriste za prenos informacija "vezanih" za program.

&IIIJ try *blok* je blok okružen velikim zagradama, u kome može doći do pojave izuzetka.

catch *blok* je blok koji sledi try blok, a u njemu se vrši obrada izuzetka.

Kada dode do izuzetka, upravljanje se prebacuje catch bloku, koji neposredno sledi posle tekućeg try bloka.

^MAPOMEMA^ Neki stari kompajleri ne podržavaju izuzetke. Izuzeci su, medutim, deo osnovnog C+ + standarda. Svi važniji proizvodači kompajlera su prihvatili da podrže izuzetke u svojim sledećim izdanjima, ako to do sada nisu učinili. Ako imate neki stariji kompajler, nečete mod da kompajlirate i startujete vežbe iz ovog poglav'ja. Medutim, ipak pažljivo pročitajte ceio poglavlje i vratite se na ovaj material, kada nabavite bolji kompajler.

Listing 20.1: Javljanje izuzetka.

```
const int DefaultSize = 10;
```



```
class Array
{
public:
   // konstruktori
   Array(int itsSize = DefaultSize);
   Array(const Array &rhs);
    -ArrayO { delete [] pType;}

    // operatori
    Array& operator=(const Array&);
    int& operator[](int offSet);
    const int& operator[](int offSet) const;

    // metode pristupa

    int GetitsSize() const { return itsSize; }

    // prijateljska funkcija
   friend ostream& operator« (ostream&, const Arrays);
    class xBoundary {};  // definiše klasu izuzetka
private:
   int *pType;
   int  itsSize;
```

```
Array::Array(int size):
itsSize(size)
{
   pType = new int[size];
   for (int i = 0; i<size; i++)
     pTypefi] = 0;
```

```
Array& Array::operator=(const Array &rhs)
{
   if (this « &rhs)
      return *this;
   delete [] pType;
   itsSize = rhs.GetitsSize();
   pType = new int[itsSize];
   for (int i = 0; i<itsSize; i++)
      pType[i] = rhs[i];
   return *this;
```

```
Array::Array(const Array &rhs)
```

nastavlja se

Listing 20.1: Joyjjonje izuietko. *nastavak*

```
53:          itsSize = rhs.GetitsSize();
54:          pType · new int[itsSize];
55:          for (int i · 0; i<itsSize; i++)
56:             pType[i] = rhs[i];
57:       }
58:
59:
60:       inti Array::operator[](int offSet)
61:       {
62:          int size = GetitsSize();
63:          if (offSet >= 0 && offSet < GetitsSizeO)
64:             return pType[offSet];
65:          throw xBoundary();
66:          return pType[0]; // zadovoljava MSC
67:       }
68:
69:
70:       const int& Array::operator[](int offSet) const
71:       {
72:          int mysize = GetitsSize();
73:          if (offSet >= 0 && offSet < GetitsSizeO)
74:             return pType[offSet];
75:          throw xBoundary();
78:          return pType[0]; // zadovoljava MSC
79:       ostream& operator« (ostream& output, const ArrayS. theArray)
80:       {
81:          for (int i = 0; i<theArray.GetitsSizeO; i++)
82:             output « "[" « i « "] " « theArray[i] « endl;
83:          return output;
84:       }
85:
86:       int main()
87:       {
88:          Array intArray(20);
89:          try
90:          {
91:             for (int j = 0; j< 100; j++)
92:             {
93:                intArray[j] = j;
94:                cout « "intArray[" « j « "] okay..." « endl;
95:             }
96:          }
97:          catch (Array::xBoundary)
98:          {}
99:             cout « "Unable to process your input!\n";
```



```
101:          cout « "Done.\n";
102:       return 0;
103:
```

IZLAZ

```
intArray[0] okay...
intArray[1] okay...
intArray[2] okay...
intArray[3] okay...
intArray[4] okay...
intArray[5] okay...
intArray[6] okay...
intArray[7] okay...
intArray[8] okay...
intArray[9] okay...
intArray[10] okay...
intArray[II] okay...
intArray[12] okay...
intArray[13] okay...
intArray[14] okay...
intArray[15] okay...
intArray[16] okay...
intArray[17] okay...
intArray[18] okay...
intArray[19] okay...
Unable to process your input!
Done.
```

p'MIWfiK Listing 20.1 predstavlja jednu vrstu skraćene klase `Array`, bazirane na templejtu, razvijenom u Danu 19, pod nazivom "Templejti". U liniji 23 klasa je sadržana unutar deklarisanih granica.

Nova klasa nije ništa drugo nego jedna klasa-izuzatak. Ona je klasa kao i svaka druga. Izuzetno je jednostavna: ona nema ni podatke, ni metode. Bez obzira na to, ona je ispravna klasa sa bilo koje tačke gledišta.

Ipak, netačno je red da ona nema metode, s obzirom da će joj kompajler automatski dodeliti podrazumevani konstruktor, destruktor, copy-konstuktor i copy-operator; stoga će ona, u stvari, imati četiri funkcije, ali ne i podatke.

Primetićete da njeno deklarisanje iz `Array` služi samo za povezivanje ove dve klase. Kao što je diskutovano u Danu 15, "Napredno nasledivanje", `Array` nema pristup klasi `xBoundary`, niti `xBoundary` ima pristup clanovima `Array`.

U linijama 60-66 i 69-75 ofset operatori su modifikovani, kako bi ispitivali zahtevani ofset, iako je on izvan opsega, da proslede klasu `xBoundary` kao izuzetak. Zagrade su neophodne, da bi se uspostavila razlika izmedu poziva konstruktora za `xBoundary` i korišćenja konstante. Primetićete da Microsoft od Vas *zahteva* da obezbedite `Return` naredbu, koja odgovara deklaraciji (u ovom slučaju, koja vraća integer referencu), čak i ako izuzetak iz linije 65 nikad neće dostid liniju 66. Ovo je bag kompajlera, koji dokazuje da čak i Microsoft nalazi da je ova stvar teška i konfuzna.

U liniji 89 ključna reč try započinje blok koji se završava u liniji 96. Unutar tog try bloka 100 integer-a je dodato u niz deklarisan u liniji 88.

U liniji 97 deklarisan je catch blok za hvatanje izuzetaka xBoundary.

U drajver programu, u linijama 96-103, kreiran je try blok, u kome je inicijalizovan svaki član niza. Kada je j (u liniji 91) uvećano za 20, pristupilo se članu sa ofset-om 20. Ovo je prouzrokovalo da test u liniji 63 ne uspe i operator [] je prouzrokovao xBoundary izuzetak u liniji 65.

Programska kontrola je prebačena na catch blok u liniji 97 i izvršena je obrada izuzetka u istoj liniji koja je prikazala poruku o grešci. Program je nastavio sa radom od kraja catch bloka u liniji 100.

try blokovi

try blok je set naredbi koji počinje rečju try (nju sledi {) i završava se sa }.

Primer:

```
try
{
Function();
};
```

catch blokovi

catch blok je serija naredbi, od kojih svaka počinje rečju catch koja sledi tip izuzetka u zagradama, a iza čega sledi { i završava se sa }.

Primer:

```
try
{
Function();
};
catch (OutOfMemory)
{
// preduzima akciju
)
```

Koriscenje try i catch blokova

Pronalaženje mesta gde treba smestiti try blokove nije trivijalan zadatak. Nije uvek jasno koja akcija može da dovede do izuzetka. Gde uhvatiti izuzetke? To može učiniti tamo gde želite da uhvatite sve memorijske izuzetke kada se alocira memorija, ali ćete želeti i da izuzetke hvatate vise u programu, tamo gde radite sa korisničkim interfejsom.

Kada pokušavate da odredite lokaciju try bloka, potražite mesta gde alocirate memoriju, ili koristite resurse. Takođe bi trebalo da potražite out-of-bounds greške, pogrešan unos i tako dalje.

Hvatanje izuzetaka

Evo kako to funkcioniše: kada dode do izuzetka, ispituje se "stek" poziva. On je *lista poziva* funkcija, koja je kreirala jedan deo programa, pozivajući druge funkcije.

"Stek" poziva prati putanju izvršavanja. Ako main() pozove funkciju Animal::Get Favorite Food() i Get Favorite Food() pozove Animal: :LookupReferences(), koji, zatim, poziva fstream: :operator»(), svi ti pozivi će se naći na "steku" poziva. Rekurzivna funkcija se na "steku" može naći vise puta.

Izuzetak će biti prosleden "steku" poziva za svaki blok. Dok se "stek" razmotava, biće pozivani destruktori za lokalne objekte i objekti će biti uništeni.

Posle svakog try bloka nalaze se jedna, ili vise catch naredbi. Ako izuzetak odgovara nekoj od catch naredbi, on će biti obraden tako što će se ta naredba izvršiti. Ako ne odgovara ni jednoj, razmotavanje "steka" će se nastaviti.

Ako izuzetak dospe čak do početka programa (mainO) i još uvek nije uključen, biće pozvan ugradeni hendler, koji će terminirati program.

Važno je napomenuti da je razmotavanje "steka" zbog izuzetaka "jednosmerna ulica". Dok ono napreduje, "stek" se razmotava i objekti sa steka se uništavaju. Nema načina da se vratite nazad. Kada je izuzetak obrađen, program nastavlja sa radom, posle try bloka catch naredbe, koja je obradila izuzetak.

Stoga, u listingu 20.1 rad će se nastaviti od linije 101, prve linije posle try bloka catch naredbe, koja je obradila xBoundary izuzetak. Zapamtite da, kada dode do izuzetka, program nastavlja sa radom posle catch bloka, a ne posle tačke u kojoj je do izuzetka došlo.

Vise od jedne catch specifikacije

Moguće je postaviti vise od jednog uslova za obradu izuzetka. U torn slučaju, catch naredbe mogu biti poravnate jedna ispod druge, nalik uslovima u switch naredbi. Ekvivalent za default naredbu je "catch everything" naredba, indicirana sa catch (...). Listing 20.2 ilustruje višestruke uslove za obradu izuzetka.

Listing 20.2: **Višestruki izuzeci.**

```
0:    'include <iostream.h>
i.
2:    const int DefaultSize = 10;
3:
```

nastavlja se

mm^r **Listing 20.2: Višestrukl izuzeci.** *nastavak*

```
4:    class Array
5:    {
6:    public:
7:       // konstruktori
8:       Array(int itsSize = DefaultSize);
9:       Array(const Array &rhs);
10:      -ArrayO { delete [] pType;}
11:
12:      // operatori
13:      Array& operator=(const Array&);
14:      int& operator[](int offSet);
15:      const int& operator[](int offSet) const;
16:
17:      // metode pristupa
18:      int GetitsSizeO const { return itsSize; }
19:
20:      // prijateljska funkcija
21:      friend ostream& operator« (ostream&, const Array&);
22:
23:    // definiše klase izuzetaka
24:      class xBoundary {};
25:      class xTooBig {};
26:      class xTooSmal1{};
27:      class xZero {};
28:      class xNegative {};
29:   private:
30:      int *pType;
31:      int itsSize;
32:   };
33:
34:      int& Array::operator[](int offSet)
35:      {
36:           int size = GetitsSizeO;
37:           if (offSet >= 0 && offSet < GetitsSizeO)
38:              return pType[offSet];
39:           throw xBoundary();
40:           return pType[0];  // zadovoljava MFC
41:      }
42:
43:
44:      const int& Array::operator[](int offSet) const
45:      {
46:           int mysize = GetitsSizeO;
47:           if (offSet >= 0 && offSet < GetitsSizeO)
48:              return pType[offSet];
49:           throw xBoundaryO;
50:           return pType[0];
51:           return pType[0];  // zadovoljava MFC
```



```
52:       1
53:
54:
55:   Array: .-Array(int size):
56:   itsSize(size)
57:   {
58:      if (size == 0)
59:         throw xZero();
60:      if (size < 10)
61:         throw xTooSmall();
62:      if (size > 30000)
63:         throw xTooBig();
64:      if (size < 1)
65:         throw xNegativeO;
66:
67:      pType = new int[size];
68:      for (int i = 0; i<size
69:        pType[i] = 0;
70:   }
71:
72:
73:
74:   int main()
75:   {
76:
77:      try
78:      {
79:         Array intArray(O);
           for (int j = 0; j< 100; j++)
           {
              intArray[j] = j;
              cout « "intArray[" « j « "] okay...\n"


        catch (Array::xBoundary)

           cout « "Unable to process your input!\n";

        catch (Array::xTooBig)

           cout « "This array is too big...\n";

        catch (Array::xTooSmal1)

           cout « "This array is too small...Vn";

        catch (Array::xZero)

           cout « "You asked for an array";
```

nastavlja se

Listing 20.2: Višestruki izuzeci. *nastavak*

```
101:          cout « • of zero objects!\n";
102:       )
103:       catch  (...)
104:       {
105:          cout « "Something went wrong!\n";
106:        }
107:         cout « "Done.\n";
108:       return 0;
109:     }
```

IZLAZ

```
You asked for an array of zero objects!
Done.
```

ANALIZA U linijama 24 do 27 kreirane su četiri nove klase: `xTooBig, xTooSmall,` xZero i xNegati ve. U konstruktoru iz linija **55** do 70 ispitana je veličina koja mu je prosledena. Ako je prevelika, premala, negativna, ili nula, doći će do izuzetka. try blok je izmenjen, tako da uključi catch naredbe za svaki uslov, osim za negativni, koji se hvata "catch everything" catch (...) naredbom u liniji 103.

Probajte ovo sa različitim vrednostima za veličinu niza. Zatim 'ppkušajte sa **-5.** Verovatno ste očekivali da će biti pozvan xNegative, ali je redosled testova konstruktora ovo sprečio: size < 10 se izvršilo pre size < 1. Da biste* ovo sprečili, zamenite linije 60 i 61, linijama 64 i 65 i izvršite ponovo kompilaciju.

Hijerarhije izuzetaka

Izuzeci su klase i iz njih se mogu vršiti izvođenja. Može biti prednost kreirati klasu xSize i iz nje izvesti xZero, xTooSmal 1, xTooBig i xNegative. Stoga, neke funkcije mogu hvatati xSize greške, dok ostale funkcije mogu hvatati greške specifičnog tipa. U listingu 20.3 je ilustrovana ova ideja.

Listing 20.3: Klasne hijerarhije i izuzeci.

```
0:    #include <iostream.h>
1:
2:    const int DefaultSize = 10;
3:
4:    class Array
5:    {
6:    public:
7:       //  konstruktori
8:       Array(int itsSize = DefaultSize);
9:       Array(const Array &rhs);
10:       -ArrayO  { delete [] pType;}
11:
12:       // operatori
13:       Array& operator=(const ArrayS);
```



```
   int& operator[](int offSet);
   const.int& operators (int offSet)  const;

   // metode pristupa

   int GetitsSizeO const { return itsSize; }

   // prijateljska funkcija
  friend ostreami operator« (ostream&, const Array&);
 // definiše klase izuzetaka
   class xBoundary {};
   class xSize {};
   class xTooBig : public xSize {};
   class xTooSmall : public xSize {};
   class xZero  : public xTooSmall {};
   class xNegative  : public xSize {};
private:
   int *pType;
   int  itsSize;
};


Array::Array(int  size):
itsSize(size)
{
   if (size == 0)
      throw xZero();
   if (size > 30000)
      throw xTooBig();
   if (size <1)
      throw xNegative();
   if (size < 10)
      throw xTooSmal1();

  pType = new int[size];
  for (int i = 0; i<size; i++)
    pType[i] = 0;


   int& Array::operator[](int offSet)

     int size = GetitsSizeO;
     if (offSet >= 0 && offSet < GetitsSizeO)
        return pType[offSet];
     throw xBoundary();
     return pType[0];  // zadovoljava MFC



const int& Array::operator[](int offSet) const
```

nastavlja se

Usting 20.3: Wasne hijerarhije i izuzeci.

```
64:
65:             int mysize = GetitsSizeO;
66:             if (offSet >= 0 && offSet < GetitsSizeO)
67:                return pType[offSet];
68:             throw  xBoundary 0;
69:             return pType[0];
70:             return pType[0];   // zadovoljava MFC
71:
72:
73:     int  main()
74:     {
75:
76:        try
77:        {
78:           Array  intArray(5);
79:           for (int j = 0; j< 100; j++)
80:           {
81:              intArray[j]  = j;
82:              cout « "intArray[" « j « "]  okay...\n"
83:           }
84:        }
85:        catch (Array::xBoundary)
86:        {
87:           cout « "Unable to process your input!\n";
88:        }
89:        catch (Array::xTooBig)
90:        {
91:           cout « "This array is too big...\n";
92:
93:
94:        catch  (Array::xZero)
95:        {
96:           cout « "You asked for an array ;
97:           cout « "  of zero objects!\n";
98:
99:
100        catch  (Array::xTooSmall)
101        {
102           cout « "This array is too small...\n"
103        }
104
105        catch  (...)
106        {
107           cout « "Something went wrong!\n";
108           1
109           cout « "Done.\n";
110        return 0;
111
```


IZLAZ

```
This array is too small...
Done.
```

ANALIZA Značajna izmena je u linijama 26-29, gde je uspostavljena hijerarhija klasa. Klase xTooBig, xTooSmall i xNegative su izvedene iz xSize, a xZero je izvedena iz xTooSmal 1.

Array je kreirana veličinom nula, ali, šta je to sad? Izgleda kao da je uhvaćen pogrešan izuzetak! Ispitajte pažljivo catch blok i videćete da on prvo ispituje izuzetak tipa xTooSmal 1, pre nego što ispituje izuzetak tipa xZero. Jednom obraden izuzetak nije prosleden drugim hendlerima, pa, stoga, hendler za xZero neće nikada biti ni pozvan.

Rešenje ovog problema je da pažljivo složite hendlere, tako da specifičniji hendleri dodu prvi, a da ih manje specifični slede. U ovom primeru zamenite mesta hendlerima xZero i xTooSmal 1 i rešićete problem.

Podaci o izuzecima i davanje imena objektima izuzetaka

Cesto ćete želeti da znate i nešto vise o kom tipu izuzetka je reč, kako biste na odgovarajući način odgovorili na grešku. Klase izuzetaka su nalik bilo kojoj drugoj klasi. Mogu da imaju podatke, da ih inicijalizuju u konstruktorima i da ih dtaju u bilo kom trenutku. Listing 20.4 Vam ilustruje kako se ovo može uraditi.

Listing 20.4: Vadenje podataka iz objekta izuzetka.

```
linclude <iostream.h>

const int DefaultSize = 10;

class Array
{
publ ic:
   // konstruktori
   Array(int itsSize = DefaultSize);
   Array(const Array &rhs);
    -ArrayO  { delete []  pType;}

    // operatori
    ArrayS operator=(const Array&);
    int&  operator[](int  offSet);
    const int& operator[](int offSet)  const;

    // metode pristupa

    int GetitsSizeO const { return itsSize; }

    // prijateljska funkcija
   friend ostream& operator« (ostream&,  const Array&);
```

nastavlja se

Listing 20.4: Vađenje podatoka iz objekta izuzetka.

```
22
23        // definiše klase izuzetaka
24          class xBoundary {};
25          class xSize
26          {
27          public:
28             xSize(int size):itsSize(size) {}
29             -xSizeOO
30             int GetSizeO { return itsSize; }
31          private:
32             int itsSize;
33          };
34
35          class xTooBig : public xSize
36          {
37          public:
38             xTooBig(int size):xSize(size){}
39
40
41          class xTooSmall : public xSize
42          {
43          public:
44             xTooSmall(int size):xSize(size){}
45
46
47          class xZero  : public xTooSmall
48          {
49          public:
50             xZero(int size) :xTooSmall (size)O
51
52
53          class xNegative : public xSize
54          {
55          public:
56             xNegative(int size):xSize(size){}
57
58
59        private:
60           int *pType;
61           int  itsSize;
62
63
64
65        Array::Array(int size):
66        itsSize(size)
67        {
68           if (size == 0)
69              throw xZero(size);
```


nastavak

```
70:          if (size > 30000)
71:             throw xTooBig(size);
72:          if (size <1)
73:             throw xNegative(size);
74:          if (size < 10)
75:             throw xTooSmal1(size);
76:
77:          pType = new int[size];
78:          for (int i = 0; i<size; i++)
79:            pType[i] = 0;
80:
81:
82:
83:       int& Array::operator[]  (int offSet)
84:       {
85:           int size = GetitsSizeO;
86:           if (offSet >= 0 && offSet < GetitsSizeO)
87:             return pType[offSet];
88:           throw xBoundaryO;
89:           return pType[0];
90:
91:
92:       const int& Array::operator[]  (int offSet) const
93:       {
94:           int size = GetitsSizeO;
95:           if (offSet >= 0 && offSet < GetitsSizeO)
96:             return pType[offSet];
97:           throw xBoundaryO;
98:           return pType[0];
99:
100:
101:        int  main()
102:
103:
104:          try
105:          {
106:             Array  intArray(9);
107:             for (int j = 0; j< 100; j++)
108:             (
109:                intArray[j] = j;
110:                cout « "intArray[" « j « "] okay... « endl;
111:
112:
113:          catch (Array::xBoundary)
114:          {
115:             cout « "Unable to process your input!\n";
116:          >
117:        catch (Array::xZero theException)
118:
```

nastavlja sc

Listing 20.4: Vadenje podataka iz objekta izuzetka. *nastavak*

```
119             cout « "You asked for an Array of zero objects!" « endl;
120             cout « "Received " « theException.GetSizeO « endl;
121
122          catch (Array::xTooBig theException)
123
124             cout « "This Array is too big..." « endl;
125
126             cout « "Received " « theException.GetSizeO « endl;
127
128          catch (Array::xTooSmall theException)
129
130             cout « "This Array is too small..." « endl;
131             cout « "Received " « theException.GetSizeO « endl;
132          catch (...)
133
134             cout « "Something went wrong, but I've no idea what!\n";
135
136          cout « "Done.\n"
137          return 0;
138
```

```
ДМ':УJ^. This array is too small...
    ' ^  Received 9
         Done.
```

Deklaracija za xSize je modifikovana, tako da uključuje promenljivu člana itsSize u liniji 32 i funkciju člana GetSize u liniji 30. Dodat je i konstruktor, koji uzima integer i inicijalizu je promenljivu člana u liniji 28.

Izvedene klase deklarišu konstruktor, koji ne radi ništa drugo, osim inicijalizacije bazne klase. Nisu deklarisane nikakve druge funkcije, kako bismo sačuvali prostor u listingu.

Naredba catch u linijama 113-135 je modifikovana tako da da ime izuzetku koji uhvati, theException, i da koristi taj objekat za pristup podacima smeštenim u itsSize.

Veoma je naporno i podložno greškama, imati ili izgraditi svaku pojedinačnu Catch naredbu za prikaz odgovarajuće poruke. Ovaj posao pripada objektima koji znaju kog su tipa i koju vrednost primaju. U listingu 20.5 prikazan je mnogo vise objektno-orijentisan pristup ovom problemu koji koristi virtuelne funkcije, tako da svaki izuzetak "zna pravu stvar."

Listing 20.5: Predavanje po referenci i koriscenje virtuelnih funkcija u izuzecima.

```
^include <iostream.h>

const int DefaultSize = 10;

class Array
```



```
{
public:
   // konstruktori
   Array(int itsSize = DefaultSize);
   Array(const Array irhs);
    -ArrayO { delete [] pType;}

    // operatori
    Array& operator[3](const Array&);
    int& operators(int offSet);
    const int& operators (int offSet) const;

   // metode pristupa
   int GetitsSizeO const { return itsSize; }

  // prijateljska funkcija
 friend ostream& operator«
     (ostream&, const Array&);

// definite klase izuzetaka
  class xBoundary {};
  class xSize
  {
  public:
     xSize(int size):itsSize(size) {}
     -xSize()0
     virtual int GetSizeO { return itsSize; }
     virtual void PrintError()
     {
         cout « "Size error. Received: ";
         cout « itsSize « endl;
     }
 protected:
    int itsSize;


class xTooBig : public xSize
{
public:
   xTooBig(int size):xSize(size){}
   virtual void PrintError()
   {
       cout « "Too big! Received: ";
       cout « xSize::itsSize « endl;


class xTooSmall : public xSiz
```

nastavlja se

Listing 20.5: Predavanje po referenci i koriscenje virtuelnih funkcija u izuzecima. *nastavak*

```
54          public:
55             xTooSmall (int  size):xSize(size){}
56             virtual  void PrintError()
57             {
58                 cout « "Too small! Received:  ";
59                 cout  «  xSize::itsSize  «  endl;
60
61
62
63          class xZero   : public xTooSmall
64          {
65          public:
66             xZero(int  size) :xTooSmall (size)U
67             virtual  void PrintErrorQ
68             {
69                 cout « "Zero!!. Received:  " ;
70                 cout  «  xSize::itsSize  «  endl;
71
72
73
74          class xNegative  : public xSize
75          {
76          public:
77             xNegative(int  size):xSize(size){}
78             virtual  void PrintError()
79             (
80                 cout « "Negative! Received:  ";
81                 cout  «  xSize::itsSize  «  endl;
82
83
84
85       pri vate:
86          int  *pType;
87          int   itsSize;
88       };
89
90       Array::Array(int  size):
91       itsSize(size)
92       {
93          if  (size  «  0)
94             throw  xZero(size);
95          if  (size  >  30000)
96             throw  xTooBig(size);
97          if  (size  <1)
98             throw  xNegative(size);
99          if  (size  <  10)
100             throw xTooSmal1(size);
101
```



```
102         pType = new int[size];
103         for (int i = 0; i<size; i++)
104           pType[i] = 0;
105
106
107      int& Array::operatorП  (int  offSet)
108      {
109          int size = GetitsSizeO;
110          if (offSet >= 0 && offSet < GetitsSizeO)
111            return  pType[offSet];
112          throw xBoundaryO;
113          return  pType[0];
114      }
115
116      const  int& Array::operator[]  (int offSet)  const
117      {
118          int size = GetitsSizeO;
119          if (offSet >= 0 && offSet < GetitsSizeO)
120            return  pType[offSet];
121          throw xBoundaryO;
122          return  pType[0];
123      }
124
125       int  main()
126
127
128          try
129          {
130             Array  intArray(9);
131             for (int j = 0; j< 100; j++)
132             {
133                intArray[j]  = j;
134                cout  «  "intArray["  «  j  «  "]  okay...\n";
135             }
136          }
137
138          catch  (Array::xBoundary)
139
140            cout  «  "Unable  to  process  your  input!\n";
141
142        catch  (Array::xSize&  theException)
143
144            theException.PrintError();
145
146        catch  (...)
147            cout  «  "Something  went  wrong!\n";
148
149        cout  «  "Done.\n"
150       return  0;
151
```


```
Too small! Received: 9
Done.
```

Listing 20.5 deklariše virtuelnu metodu klase `xSize`, pod imenom `PrintError()`. Zatim prikazuje poruku o grešci i stvarnu veličinu klase. U izvedenim klasama izveden je `override` ove metode.

U liniji 141 objekat izuzetak je deklarisan kao referenca. Kada se `Pri nt Error` pozove sa referencom na objekat, polimorfizam prouzrokuje da bude pozvana ispravna **ve-** rzija metode `PrintError()`. Kod je čistiji, jednostavniji za razumevanje i lakši za održavanje.

Izuzeci i templejti

Kada kreirate izuzetke da rade sa templejtima, Vi imate izbor: možete kreirati jedan izuzetak za svaku instancu templejta, ili koristiti klase izuzetaka, deklarisane izvan templejt deklaracije. U listingu 20.6 ilustrovana su oba pristupa.

Listing 20.6: Korišćenje izuzetaka sa šablonima.

```
0:      linclude <iostream.h>
1:
2:      const int DefaultSize = 10;
3:      class xBoundary {};
4:
5:      template <class T>
6:      class Array
7:      {
8:      public:
9:         // konstruktori
10:         Array(int itsSize = DefaultSize);
11:         Array(const Array &rhs);
12:         -Array() { delete [] pType;}
13:
14:         // operatori
15:         Array& operator=(const Array<T>&);
16:         T& operators(int offSet);
17:         const T& operator[](int offSet) const;
18:
19:         // metode pristupa
20:         int GetitsSize() const { return itsSize; }
21:
22:         // prijateljska funkcija
23:         friend ostream& operator« (ostream&, const Array<T>&);
24:
25:      // definiše klase izuzetaka
26:
27:         class xSize {};
28:
```



```
private:
   int *pType;
   int itsSize;
};

template <class T>
Array<T>::Array(int size):
itsSize(size)
{
   if (size <10 I! size > 30000)
      throw xSize();
   pType = new T[size];
   for (int i = 0; i<size; i++)
     pType[i] = 0;


template <class T>
Array<T>& Array<T>::operator=(const Array<T> &rhs)
{
   if (this == &rhs)
      return *this;
   delete [] pType;
   itsSize = rhs.GetitsSizeO;
   pType = new T[itsSize];
   for (int i = 0; i<itsSize; i++)
      pType[i] = rhs[i];
}
template <class T>
Array<T>::Array(const Array<T> &rhs)
{
   itsSize = rhs.GetitsSizeO;
   pType = new T[itsSize];
   for (int i = 0; i<itsSize; i++)
      pType[i] = rhs[i];


template <class T>
T& Array<T>::operator[](int offSet)

   int size = GetitsSizeO;
   if (offSet >= 0 && offSet < GetitsSizeO)
      return pType[offSet];
   throw xBoundary();
   return pType[0];
}

template <class T>
const T& Array<T>::operator[](int offSet) const
```

nastavljase

Listing 20.6: Koriscenje iiuietako sa sablonima. *nastavak*

```
78          int mysize - GetitsSize0;
79          if (offSet >= 0 && offSet < GetitsSize0)
80             return pType[offSet];
81          throw xBoundary0;
82       }
83
84       212                (ostream output, const Arrays theArray)
85
86       {
87          for (int i - 0; i<theArray.GetitsSize0;
88             output « "[" « 1 <K "l " K< ^eArray[i] « endl;
89          return output;
90
91
92
93       int main()
94
95
96          try
97
98             Array<int>  intArray(9);
99             for (int j · 0; j< 100;
100              (
101                 int Array [j] = 3*.
102                 cout « "intArray[" « j « "] okay.. H « endl;
103
104
105         catch (xBoundary)
106
107            cout « "Unable to process your input!\n";
108
109         catch (Array<int>::xSize)
110
               cout « "Bad Size!\n";


            cout « "DoneAn";
            return 0;
```

IZLAZ

```
Bad Size!
Done.
```

ANALIZA

Prvi izuzetak, `xBoundary`, deklarisan je izvan definicije templejta, u liniji 3. Drugi izuzetak `xSize` je deklansan unutar definicije templejta, u limn ***it.***

Izuzetak `xBoundary` nije povezan sa `templejt` klasom, all se moze koja druga klasa. Izuzetak `xSize` je povezan sa templejtom i mora *s e ^ U i* instance `Array`. Videćete razliku izmedu sintaksi za ove dve naredbe za hvatanje.

liniji 105 piše catch **(xBoundary),** ali u liniji 109 pise catch **(Array<i** nt>: **rxSize).** Ovo drugo je povezano sa **Array.**

Izuzeci bez grešaka

Kada se C + + programeri nadu zbog virtuelnog piva u Sajber baru posle posla, razgovor se često preokrene na izuzetke koje bi trebalo koristiti u rutinama. Izuzeci bi trebalo da budu rezervisani za predvidive, ali izuzetne situacije, njih programer mora da predvidf, ali one nisu deo koda.

Drugi ukazuju, da izuzeci nude snažan i čist način za povratak iz većine poziva funkcija, bez opasnosti od gubljenja memorije. Primer koji se često javlja je sledeći: korisnik zahteva akciju u Gui okruženju. Deo koda koji prihvata zahtev mora da pozove funkciju člana menadžera za dijalog. Taj clan zatim poziva kod za obradu zahteva, a on poziva kod koji odlučuje koji dijalog da upotrebi. Dijalog zatim poziva kod za dijalog, a on konačno poziva kod za obradu unosa korisnika. Ako korisnik pritisne dugme **Cancel,** kod se **mora** vratiti do prve pozvane metode, gde je obraden originalni zahtev.

Jedan pristup ovom problemu je staviti try blok "oko" originalnog poziva i uhvatati **Cancel** dijalog, kao izuzetak koji bi mogao nastati hendlerom za **Cancel** dugme. Ovo je sigurno i efikasno, ali pritisak na **Cancel** je jedna od redovnih rutina, a ne izuzetak.

Ovo Često postaje nešto kao "religiozni dokument", ali postoje načini da se odgovori na pitanja: da li **koriscenje** izuzetaka na ovaj način čini kod jednostavnijim, ili komplikovanijim za razumevanje? Da li postoji manje rizika za pojavu grešaka i gubljenje memorije, ili vise? Da li će biti teže, ili jednostavnije održavati ovakav kod? Odgovori na ova, kao i na mnoga druga pitanja, zahtevaju analizu isplativosti i ne postoji jednostavno ispravno rešenje.

Bagovi i debagovanje

U Danu 17, "Pretprocesor", videli ste kako se koristi **assert ()** za presretanje bagova u fazi izvršavanja **tokom** faza testiranja, a danas ste videli kako se koriste izuzeci za presretanje problema u **fazi** izvršavanja. Postoji još jedno, izuzetno **jako** oružje koje ćete želeti da dodate u Vaš arsenal, kada napadate bagove. Ono se naziva debager.

Uglavnom sva moderna razvojna okruženja uključuju jedan, ili vise debagera izuzetne snage. Osnovna ideja korišćenja je sledeća: Vi startujete debager, koji učitava Vaš izvorni kod, i zatim startujete Vaš program unutar debagera. Ovo Vam omogućava da vidite svaku instrukciju u Vašem programu kako se izvršava i da ispitate Vaše promenljive i njihove izmene tokom života Vašeg programa. Svi kompajleri će Vam dozvoliti da izvršite kompilaciju sa, ili bez simbola. Kompajliranje sa simbolima govori kompajleru da kreira neophodna mapiranja između Vašeg izvornog koda i generisanog programa; debager koristi ovo da **bi** ukazao na liniju izvornog koda koja odgovara sledećoj akciji u programu.

Ekranski simbolički debageri cine ove napore jednostavnijim. Kada učitate Vaš debager, on će pregledati ceo Vaš izvorni kod i prikazati ga u prozoru. Vi ćete moći da, korak, po korak, pozivate funkcije, ili da naredite debageru da ude unutar funkcije i izvršava je liniju, po liniju.

Kod većine debagera možete se "šetati" između izvornog koda i izlaza, kako biste videli rezultate svake izvršene naredbe. Štaviše, možete ispitati trenutno stanje bilo koje promenljive, pregledati kompleksne strukture podataka, ispitati vrednosti podataka cianova bilo koje klase i videti stvarne vrednosti u memoriji za različite pointere i druge memorijske lokacije. Možete postaviti veći tip kontrola unutar debagera, kao što su prekidne tačke, tačke za posmatranje, ispitivanje memorije i pregled asemblerskog koda.

Prekidne tačke

Prekidne tacke su instrukcije debageru da kada naide na liniju koja je označena prekidnom tačkom zaustavi rad programa. Ovim Vam je omogućeno da program nesmetano radi, dok ne naide na problematičnu liniju. Prekidne tačke Vam pomažu da analizirate tekuće uslove u kojima se program nalazi, kao i stanje promenljivih pre i posle kritične linije koda.

Tacke posmatranja

Moguće je red debageru da Vam prikaže vrednost odredene promenljive, ili da zaustavi program kada dode do čitanja, ili pisanja određene promenljive. *Tačke posmatranja* Vam omogućavaju da postavite ove uslove i vremenom čak i da izmenite vrednosti promenljivih u toku rada programa.

Ispitivanje memorije

Ponekad je vrlo važno videti stvarnu vrednost koja se nalazi u memoriji. Moderni debageri Vam mogu prikazati vrednosti u formi određene promenljive, što znači da će stringovi biti prikazani kao karakteri, longs kao brojevi umesto četiri bajta i tako dalje. Sofisticirani C + + debageri Vam, čak, mogu prikazati kompletne klase, zajedno sa trenutnim vrednostima svih promenljivih članova, uključujući i this pointer.

Asembler

Iako debagovanje kroz izvorni kod može biti sve što Vam je potrebno da pronadete bag, postoje situacije u kojima je, kada ništa drugo ne uspeva, moguće naložiti debageru da Vam prikaže trenutni asemblerski kod generisan za svaku liniju Vašeg izvornog koda. Možete ispitati registre i flag-ove i generalno duboko ući u rad Vašeg programa, ako je to potrebno.

Naučite da koristite debager. On Vam može biti najsnažnije oružje u Vašem "svetom ratu" protiv bagova. Bagovi u fazi izvršavanja su najteži za pronalaženje i izvršavanje, pa Vam snažan debager može pomoći, iako ne na jednostavan način, da pronadete praktično bilo koji od njih.

§ Rezime

Danas ste naučili kako da kreirate i da koristite izuzetke. Izuzeci su objekti koji mogu biti kreirani i prosledeni tačkama u programu, gde izvršni kod ne može da obradi grešku, ili tamo gde je nastala greška izuzetka. Drugi delovi programa koji se nalaze vise u steku poziva, implementiraju catch blokove koji hvataju izuzetke i izvršavaju odgovarajuće akcije.

Izuzeci su normalni, korisnički kreirani objekti i mogu biti prosledeni po vrednosti, ili po referenci. Oni mogu sadržati podatke i metode i catch blokovi mogu koristiti te podatke, kako bi odlučili na koji način da se izbore sa izuzetkom.

Moguće je kreirati višestruke catch blokove, ali kada izuzetak jednom odgovori signaturi catch bloka, on će biti obraden i neće biti prosleden sledećim catch blokovima. Zato je važno na odgovarajući način u programu složiti catch blokove tako da specifični catch blokovi prvi dobiju šansu, a opštiji Catch blokovi slede za njima.

U ovom poglavlju su takođe objašnjene osnove simboličkog debagera, uključujući koriscenje prekidnih tačaka, tačaka za posmatranje i tako dalje. Ovi alati Vam mogu pomoći da pregledate delove programa koji prouzrokuju grešku i omogućavaju da vidite vrednosti promenljivih i kako se one menjaju tokom izvršenja programa.

Pitanja i odgovori

P Zašto koristiti izuzetke? Zašto greške ne obrađivati tamo gde one nastaju?

O Vrlo često, ista greška može nastati u velikom broju različitih delova koda. Izuzeci Vam omogućavaju da centralizujete obradu grešaka. Uz to, deo koda koji je generisao grešku ne mora biti najbolje mesto za odredivanje procedure, na osnovu koje će se greška obraditi.

P Zašto generisati objekat? Zar nije dovoljno samo prosledti kod greške?

O Objekti su fleksibilniji i snažniji od kodova grešaka. Oni mogu sadržati vise informacija, kao i konstruktor/destruktor mehanizme, za kreiranje i uklanjanje resursa, koji mogu biti neophodni za adekvatnu obradu greške i uslove izuzetaka.

P Zašto se izuzeci ne koriste u uslovima kada greške ne postoje? Zar ne bi bilo zgodno ekspresno se vratiti u prethodnu zonu koda, čak i kada je došlo do uslova koji nije izuzetak?

^

^^^^^^^^ **Da. Neki C + + programed koriste izuzetke baš na taj način. Opasnost je da izuzeci mogu dovesti do gubitka memorije tokom odmotavanja steka, kada neki objekti nenamerno ostanu u slobodnom prostoru. Sa tehnikama pažljivog programiranja i sa dobrim kompajlerom, ovo će verovatno biti izbegnuto. Inače, to je lična stvar svakog programera. Neki programed osećaju da ne bi trebalo da koriste ove izuzetke u redovnim rutinama.**

P **Da li izuzetak mora da bude uhvaćen na istom mestu gde je try blok kreirao izuzetak?**

0 **Ne. Izuzetak je moguće uhvatititi bilo gde u steku poziva. Dok se stek odmotava, izuzetak će se prosledivati, sve dok ne bude obraden.**

P **Zašto koristiti debager, kada možemo koristiti cout sa uslovnim (**#ifdef debug**) kompajliranjem?**

0 **Debager obezbeduje mnogo jače mehanizme za izvršenje koda, korak, po korak, i posmatranje izmena vrednosti, bez potrebe da Vaš kod filujete hiljadama naredbi za debagovanje.**

Radionica

Nudimo Vam test, koji treba da Vampomogne da utvrdite svoje razumevanje predenog materijala i vežbe, koje Vam obezbeđuju iskustvo u korišćenju onog šta ste naučili. Pokušajte da odgovorite na test i da uradite vežbanja, pre proveravanja odgovora u Dodatku D, i budite sigurni da **razumete odgovore, pre nego što predete** na **sledeće poglavlje.**

Kviz pitanja

1. Sta **je izuzetak?**
2. **Štaje try blok?**
3. **Šta je** catch **naredba?**
4. **Koje informacije izuzetak može da sadrži?**
5. **Kada se kreira izuzetak objekat?**
6. **Mogu li se izuzeci proslediti po vrednosti, ili po referenci?**
7. **Da li će catch naredba uhvatiti izvedeni izuzetak, ako ona traži baznu klasu?**
8. **Ako postoje dve** Catch **naredbe, jedna za baznu, a druga za izvedenu klasu, koja će se prva izvršiti?**
9. **Šta znači** catch (...)?
10. **Šta je prekidna tačka?**

Vežbe

^3P^88*

1. **Kreirajte try blok, catch naredbu i jednostavan izuzetak.**
2. **Modifikujte odgovor iz Vežbe 1, tako što ćete smestiti podatke u izuzetak, zajedno sa pristupnom funkcijom, i što ćete to iskoristiti u catch bloku.**
3. **Modifikujte klasu iz Vežbe 2, tako da bude hijerarhija izuzetaka; modifikujte catch blok, tako da koristi izvedene objekte i bazne objekte.**
4. **Modifikujte program iz Vežbe 3, tako da ima tri nivoa poziva funkcija.**
5. **ISTERIVAČI BAGOVA: šta je neispravno u sledećem kodu:**

```
class xOutOfMemory
{
public:
     xOutOfMemory( const String& message ) : itsMsg( message ){}
     -xOutOfMemory(){}
     virtual const String* Message(){ return itsMsg};
private:
     String itsMsg;
} .

main()
{
     try {
          char *var = new char;
          if ( var == 0 )
               throw xOutOfMemory();
     }
     catch( x0ut0fMemory& theException )
     {
          cout «  theException.Message() « "\n";
```

Dan 21

v

Sta dalje?

Čestitamo! Gotovo ste pri kraju tronedeljnog intenzivinog upoznavanja C++-a. Sada bi trebalo da ste stekli već jedno solidno znaje o C+ +, ali u modernom programiranju uvek postoji još toga što treba naučiti. Ovo poglavlje će upotpuniti Vaše znanje sa još nekim propuštenim detaljima, a zatim će trasirati put za dalji rad.

Danas ćete naučiti

- šta su standardne biblioteke
- kako manipulisati pojedinačnim bitovima i koristiti ih kao flag-ove
- koji su sledeći koraci potrebni da biste naučili da efikasno koristite C + + .

Standardne biblioteke

Svaka implementacija C + + uključuje standarne biblioteke, a većina ih uključuje i dodatne biblioteke. Biblioteke su setovi funkcija, koji mogu biti povezani sa Vašim kodom. Vi ste već koristili izvestan broj standardnih bibliotečkih funkcija i klasa, što se najbolje vidi kod iostream biblioteke.

Da biste upotrebili biblioteku, Vi, obično, u Vaš izvorni kod uključujete i zaglavlja datoteke, baš kao što ste radili kada ste pisali #i nclude <iostream.h> u velikom broju primera u ovoj knjizi. Uglaste zagrade ispred i iza naziva datoteke su signal kompajleru da "zaviri" u direktorijum, gde ste smestili zaglavlja datoteka za Vaše standardne biblioteke kompajlera.

Postoji na desetitne biblioteka, koje pokrivaju sve, od manipulisanja datotekom do postavljanja datuma i vremena. Danas ćemo pobrojati samo nekoliko najpopularnijih funkcija i klasa iz standardne biblioteke, o kojima, do sada, još nije bilo reči u ovoj knjizi.

String

Najpopularnija biblioteka je, gotovo sigurno, string biblioteka, sa funkcijom stri en () koja se, možda, najčešće i poziva.; stri en () vraća dužinu Null-terminiranog stringa. U listingu 21.1 prikazano je kako se ona koristi.

Listing 21.1: s t r l e n Q .

```
1:    #include <iostream.h>
2:    linclude <string.h>
3:
4:    int  main()
5:    {
6:       char  buffer80];
7:       do
8:       {
9:          cout « "Enter a string up to 80 characters:
10:          cin.getline(buffer,80);
11:          cout « "Your string is " « strlen(buffer);
12:          cout « " characters long." « endl;
13:       }    while (strien(buffer));
14:       cout « "\nDone." « endl;
15:    return 0;
16: }
```

IZLAZ

```
Enter a st/ing up to 80 characters: This sentence has 31 characters
Your string is 31 characters long.
Enter a string up to 80 characters: This sentence no verb
Your string is 21 characters long.
Enter a string up to 80 characters:
Your string is 0 characters long.

Done.
```

feuM:^Pfc U liniji 6 kreiran je bafer karaktera, a u liniji 9 korisnik je upitan da unese string. Sve vreme dok korisnik unosi string, dužina stringa se "prijavljuje" u liniji 11.

Primetićete uslov u do...while() naredbi: while (stri en (buffer)). Tekpošto stri en () bude vratila 0, kada se bafer isprazni i stoga 0 proizvede FALSE, ova whi le petlja će nastaviti (sa radom), sve dok u baferu bude karaktera.

strcpyO i strncpy()

Druga ne manje popularna funkcija u string.h/e, verovatno, strcpyO, koja kopira jedan string u drugi. Možda je ova njena popularnost sada na neki način umanjena, budući da su Ntdl-tenninirani stringovi u stilu C jezika izgubili svoj značaj u C++-u tipično je da se manipulisanje stringom vrši u okviru stri ng klase, koju je obezbedio proizvodač, ili koju je napisao korisnik. Pa ipak, Vaša string klasa mora da podrži operator dodeljivanja i kopi konstruktor, za čiju se implementaciju često koristi strcpyO, kao što je prikazano u listingu 21.2.

Listing 21.2: Koriscenje s t r c p y .

```
      linclude <iostream.h>
      linclude <string.h>

      int  main()
      {
         char string0ne80];
         char stringTwo80];

         string0ne0]='\0';
10       stringTwoO]='\0';

12       cout « "String One:   « stringOne « endl;
13       cout « "String Two:   « StringTwo « endl;
14
15       cout « "Enter a string:  ";
16       cin.getline(stringOne,80);
17
18       cout « "\nString One:  " « stringOne « endl;
19       cout « "String Two:  " « StringTwo « endl;
20
21       cout « "copying..." « endl;
22       strcpy(stringTwo,stringOne);
23
24       cout « "\nString One:  " « stringOne « endl;
25       cout « "String Two:  " « StringTwo « endl;
26       cout « "\nDone  " « endl;
27    return 0;
28
```

```
String One:
                String Two:
Enter a string:  Test of strcpyO

String One:  Test of strcpyO
String Two:
copying...
```

```
String One:   Test of strcpyO
String Two:   Test of strcpyO

Done
```

JEZBSdhfe Dva Null-terminirana stringa C-stila su deklarisana u linijama 6 i 7. Oni su inicijalizovani kao prazni u linijama 9 i 10 i njihove vrednosti su prikazane u linijama 12 i 13. Korisnik je upitan da unese string i rezultat je unet u stringOne; dva stringa su ponovo prikazana, sada je samo stringOne imao ulaz. Zatim je pozvana strcpyO i stringOne je kopiran u stri ngTwo.

Primetićete da se sintaksa strcpyO može pročitati (protumačiti) kao "kopiraj u prvi parametar string iz drugog parametra. Šta će se dogoditi ako je odredišni string (stri ngTwo) suviše mali da bi mogao da prihvati kopirani string? Ovaj problem i njegovo rešenje su ilustrovani u listingu 21.3.

Listing 21.3: Koriscenje strncpy().

```
linclude <iostream.h>
linclude <string.h>

int  main()
{
   char stringOne[80];
   char stringTwo[10];
   char stringThree[80];

   stringOne[0] = '\0';
   stringTwo[0]='\0';
   stringThree[0]='\0';

   cout « "String One:  " « stringOne « endl;
   cout « "String Two:  " « StringTwo « endl;
   cout « "String Three:  " « stringThree « endl;

   cout « "Enter a long string: ";
   cin.getline(stringOne,80);
   strcpy(stringThree,stringOne);
//    strcpy(stringTwo,stringOne);

   cout « "\nString One:  " « stringOne « endl;
   cout « "String Two:  " « StringTwo « endl;
   cout « "String Three:  " « stringThree « endl;

   strncpy(stringTwo,stringOne,9);

   cout « "\nString One:  " « stringOne « endl;
   cout « "String Two:  " « StringTwo « endl;
   cout « "String Three:  " « stringThree « endl;
```



```
   stringTwo[9]='\0';

   cout « "\nString One:  " « stringOne « endl;
   cout « "String Two:  " « StringTwo « endl;
   cout « "String Three:  " « stringThree « endl;
   cout « "\nDone." « endl;
return 0;


   String One:
                  String Two:
   String Three:
   Enter a long string: Now is the time for all.

   String One: Now is the time for all...
   String Two:
   String Three:  Now is the time for all...

   String One: Now is the time for all...
   String Two:  Now is th_+đđ
   String Three:  Now is the time for all...

   String One: Now is the time for all...
   String Two:  Now is th
   String Three:  Now is the time for all...

   Done.
```

*IIIE.IIIII$£** U linijama 6,7 i 8 deklarisana su tri bafera za string. Primetićete da je `stri ngTwo` deklarisan samo za 10 karaktera, dok su ostali deklarisani za 80. Sva tri su inicijalizovana na dužinu 0 u linijama 10 do 12 i prikazana u linijama 14 do 16.

Korisnik je upitan da unese string i ovaj string je kopiran u string tri u liniji 20. Linija 21 je iskomentarisana; kopiranje ovog dugačkog stringa u `StringTwo` dovelo bi do pada programa, s obzirom da bi došlo do pisanja u memoriju, koja je kritična za program.

Standardna funkcija strcpyO počinje kopiranjem adrese, koja je označena kao prvi parametar (naziv niza), i kopiraće kompletan string, bez prethodne provere da li ste za njega alocirali dovoljan prostor!

Standardna biblioteka nudi drugu, sigurniju funkciju, `strncpy()`, koja kopira samo specificiran broj karaktera u odredisni string. Slovo n u sredini funkcije `strncpy`() je mesto za broj. Ovo je konvencija koja se primenjuje u svim standardnim bibliotekama.

U liniji 27 prvih devet karaktera `stringOne` je (is)kopirano u `stri ngTwo` i prikazan je rezultat. Pošto `strncpy` () ne stavlja Null na kraj (is)kopiranog stringa, dobijeni rezultat neće biti onakav kako je planirano. Primetićete da strcpyO stavlja Null na kraj iskopiranog stringa, a da `strncpy`0 to ne čini, tek da bi "život učinila interesantnijim!"

Null je dodato u liniji 33 i tada su stringovi prikazani poslednji put.

strcat() i strncat()

U vezi sa strcpyO i strncpyO, postoje standardne funkcije strcatQ i strncat() Prva vrši konkatenaciju jednog stringa sa drugim, odnosno, ona dodaje string koji uzima kao svoj drugi parametar na kraj stringa koji uzima kao svoj prvi parametar a, strncatQ, kao što ste mogli da očekujete, dodaje prvih n karaktera iz jednog stringa u drugi.

Listing 21.4: Korišćenje strcatQ i strncatQ.

```
linclude <iostream.h>
#include <string.h>


int  main()
{
   char string0ne[255];
   char stringTwo[255];

   stringOne[0]='\0';
   stringTwo[0]='\0';

   cout « "Enter a  string: ";
   cin.getline(string0ne,80);

   cout « "Enter a second string:
   cin.getline(stringTwo,80);

   cout « "String One:  " « stringOne « endl;
   cout « "String Two:  " « stringTwo « endl;

   strcat(string0ne," ");
   strncat(stringOne,stringTwo,10);

   cout « "String One:  " « stringOne « endl;
   cout « "String Two:  " « stringTwo « endl;

  return 0;
9:    }
```

IZLAZ

```
Enter a string: Oh beautiful
Enter a second string: for spacious skies for amber waves of grain
String One: Oh beautiful
String Two: for spacious skies for amber waves of grain
String One: Oh beautiful for spacio
String Two: for spacious skies for amber waves of grain
```

U linijama 7 i 8 kreirana su dva niza karaktera i od korisnika je traženo da unese dva stringa koji su smešteni u ove nizove. Prazan karakter je dodat stringOne-u, u liniji 22. U liniji 23 i prvih deset karaktera stringTwo-a je dodato na stri ngOne. Rezultat je prikazan u liniji 25 i 26.

Ostale string funkcije

Sting biblioteka obezbeđuje veći broj drugih string funkcija, uključujući i one za pronalaženje različitih karaktera, ili token-a unutar stringa. Ako Vam je potrebno da pronadete zarez, ili određenu reč, a koja se pojavljuje u stringu, pogledajte biblioteku, da biste videli da li funkcija koja Vam je potrebna već postoji.

Vreme i datum

"Vremenska biblioteka" Vam nudi veći broj funkcija za dobijanje približne aproksimacije tekućeg vremena i datuma i za međusobno poređenje vremena i datuma. Centralno mesto u ovoj biblioteci pripada strukturi tm, koja se sastoji od devet celobrojnih vrednosti za sekunde, minute, sate, dane u mesecu, brojeve meseci (gde je januar = 0), brojeve godina (od 1900), dane u nedelji (gde je nedelja = 0), dane u godini (0-365) i logičke vrednosti koje ozn*čavaju da li se koristi, ili ne letnje računanje vremena (ovo poslednje ne mora biti podržano u svim sistemima).

Većina ovih funkcija zahteva promenljivu tipa time_t, ili pointer na promenljivu ovog ti-pa. Postoje rutine koje vrše konverziju promenljivih ovog tipa u tm strukturu i obrnuto. Standardna biblioteka sadrži funkciju time(), koja zahteva pointer na promenljivu time_t i popunjava je tekućim vremenom. Funkcija ctime() zahteva promenljivu time_t, popunjenu vremenom, i vraća ASCII string koji se može koristiti za prikazivanje. Ako želite veću kontrolu nad izlazom, možete proslediti promenljivu time_t funkciju local_time() - ona će vratiti pointer na tm strukturu. U listingu 21.5 ilustrovane su različite funkcije koje rade sa vremenom.

Listing 21.5: Korišćenje ctimeQ.

```
#include <time.h>
#include <iostream.h>

int  main()
{
   time_t currentTime;

   // uzima i štampa trenutno vreme
   time (&currentTime); // sada popuni sa trenutnim vremenom
   cout « "It is now " « ctime(&currentTime) « endl;

   struct tm * ptm= localtime(&currentTime);

   cout « "Today is " « ((ptm->tm_mon)+l) « "/";
   cout « ptm->tm_mday « "/";
   cout « ptm->tm_year « endl;

   cout « "\nDone.";
 return 0;
```

```
igggj^  It is now Mon Mar 31 13:50:10 1997

        Today is 3/31/97

        Done.
```

j^IIIIIIj^ U liniji 6 promenljiva CurrentTi me je deklarisana kao promenljiva tipa time_t. Adresa ove promenljive je prosleđena standardnoj bibliotečkoj funkciji time(), člme je promenljiva CurrentTime postavljena na tekući datum i vreme. Adresa ove promenljive je, zatim, prosledena funkciji ctime(), koja je vratila ASCII string, a on je uz pomoć cout naredbe, prikazan u liniji 12.

Adresa promenljive CurrentTime je, zatim, prosledena standardnoj bibliotečkoj funkciji Local Ti me(), koja je vratila pointer na tm strukturu, a koja je iskorišćena za inicijalizaciju lokalne promenljive ptm. Podatak clan ove strukture je, nakon toga, iskorišćen za prikaz tekućeg meseca, dana u mesecu i godine.

stdlib

stdl i b je kolekcija različitih opštih funkcija, koje nisu mogle da se smeste ni u jednu drugu biblioteku. Ona uključuje jednostavne celobrojne matematičke funkcije, funkcije za sortiranje (recimo, qsort(), jedan od najbržih sort alogoritama) i tekst konverziju za pretvaranje ASCII teksta u integer-e, long-ove, float-e, itd.

Funkcije iz stdl ib-a, koje ćete najčešće koristiti, su: atoi (), itoa(), kao i familija sličnih funkcija. atoi () obezbeduje konverziju ASCII u integer, atoi () uzima jedan argument: pointer na konstantni karakter string. Ova funkcija vraća integer (kao što se i moglo očekivati). Listing 21.6 ilustruje njeno koriscenje.

Listing 21.6: Koriscenje atoi () i srodnih funkcija.

```
1:    linclude <stdlib.h>
2:    linclude <iostream.h>
3:
4:    int main()
5:    {
6:       char buffer[80];
7:       cout « "Enter a number: ";
8:       cin » buffer;
9:
10:      int number;
11:      // number = buffer; greška prilikom prevodenja
12:      number = atoi(buffer);
13:      cout « "Here's the number: " « number « endl;
14:
15:      // int sum = buffer + 5;
16:      int sum = atoi(buffer) + 5;
17:      cout « "Here's sum: " « sum « endl;
```



```
18:      return 0;
19:   }

        Enter a number: 9
        Here's the number: 9
        Here's sum; 14
```

U liniji 6 ovog jednostavnog programa alociran je bafer od 80 karaktera i u liniji 7 od korisnika je traženo da unese broj. Ulaz je preuzet kao tekst i upisan u bafer.

U liniji 10 deklarisana je celobrojna promenljiva number i u liniji 11 program je pokušao da dodeli sadržaj bafera celobrojnoj promenljivoj. Ovo prouzrokuje grešku u kompilaciji i zbog toga je iskomentarisano.

U liniji 12 problem je rešen pozivom standardne biblioteke funkcije atoi (), kojoj je bafer prosleden kao parametar. Vraćena vrednost - celobrojna vrednost tekst stringa je dodeljena celobrojnoj promenljivoj number, prikazanoj u liniji 13.

U liniji 15 je deklarisana nova celobrojna promenljiva sum i pokušano je da joj se dodeli rezultat sabiranja celobrojne konstante 5 i bafera. Ovo je, naravno, prouzrokovalo grešku u kompilaciji i rešeno je pozivanjem standardne funkcije atoi ().

yNAPOMHA^ Neki kompajleri implementiraju procedure za standardnu konverziju, kao što je atoi (), uz pomoć makroa. Obično možete koristiti ove funkcije bez mnogo brige o tome kako su implementirane. Za vise detalja "posavetujte* se sa dokumentacijom Vaseg kompajlera.

qsort()

Vremenom ćete poželeti da sortirate tabelu, ili niz; qsort() Vam obezbeduje brz i jednostavan način da biste to uradili. Teži deo korišćenja qsort() je postavljanje struktura koje mu treba proslediti.

Funkcija qsort() zahteva Četiri argumenta. Prvi je pointer na početak tabele koju treba sortirati (biće prihvaćeno i ime niza), drugi je broj elemenata u tabeli, treći je veličina svakog elementa i četvrti je pointer na funkciju za uporedivanje.

Funkcija za uporedivanje mora da vraća int i mora da kao parametre uzima dva konstantna void pointera. koji pointeri se ne koriste Često u C + 4**--U,** s obzirom da prave probleme u proveri tipa, ali imaju prednost, jer se mogu koristiti za ukazivanje na stavke bilo kojeg tipa. Ako budete pisali sopstvenu qsort() funkciju, mogli biste da razmislite o korišćenju templejta. U listingu 21.7 ilustrovano je kako se koristi standardna qsort() funkcija.

Listing 21.7: Koriscenje qsort().

```
1:    /* qsort primer */
2:
3:    linclude <iostream.h>
```

nastavlja se

nastavak

Listing 21 J: Koriscenje qsort().

```
linclude <stdlib.h>

// oblik sort_function koji je potreban za qsort
int sortFunction( const void *intOne, const void *intTwo);

const int TableSize = 10;  // veličina niza

int main(void)
{
   int i,table[TableSize];

   // puni tabelu sa vrednostima
   for (i = 0; i<TableSize; i++)
   {
      cout « "Enter a number:
      cin » table[i];
   }
   cout « "\n";

   // sortira vrednosti
   qsort((void *)table, TableSize, sizeof(table[0]), sortFunction);

   // štampa rezultate
   for (i = 0; i < TableSize; i++)
      cout « "Table [" « i « "]: " « table[i] « endl;

   cout « "Done." « endl;
  return 0;

int sortFunction( const void *a, const void *b)
{
   int intOne = *((int*)a);
   int intTwo = *((int*)b);
   if (intOne < intTwo)
      return - 1;          «
   if (intOne == intTwo)
      return 0;
   return 1;
```

```
Enter a number: 2
Enter a number: 9
Enter a number: 12
Enter a number: 873
Enter a number: 0
Enter a number: 45
Enter a number: 93
Enter a number: 2
Enter a number; 66
Enter a number: 1

Table[0]: 0
Tablefl]: 1
Table[2]: 2
Table[3]: 2
Table[4]: 9
Table[5]: 12
Table[6]: 45
Table[7]: 66
Table[8]: 93
Table[9]: 873
Done.
```

plMlkjjh^ U liniji 4 uključeno je zaglavlje standardne biblioteke - njega zahteva funkcija qsort(). U liniji 7 deklarisana je funkcija sortFunction(), koja preuzima četiri parametra.

Niz je deklarisan u liniji 13 i popunio ga je korisnik u linijama 16-20; qsort je pozvan u liniji 24, dodelivši adresi imena niza Table da bude void*.

Primetićete da parametri za sortFunction () nisu prosledeni u pozivu qsort (). Ime sortFunction, koje je, samo po sebi, pointer na tu funkciju, je parametar za qsort().

U svom radu funkcija qsort će popuniti konstantne void pointere a i b vrednostima niza. Ako je prva vrednost manja od druge, funkcija za poredenje će vratiti - 1 . Ako su vrednosti jednake, funkcija za poređenje će vratiti nula. I na kraju, ako je prva vrednost veća od druge, funkcija za poređenje će vratiti 1. Ovo je prikazano u sortFunction(), u linijama 34-43.

Druge biblioteke

Uz Vaš C + + kompajler dobićete i ved broj drugih biblioteka, kao što su biblioteke za standardni ulaz i izlaz i steam biblioteke, koje ste koristili u ovoj knjizi. Dobro bi bilo da uložite izvestan napor i odvojite malo Vašeg vremena, kako biste istražili dokumentaciju koja dolazi sa Vašim kompajlerom i saznali šta Vam ove biblioteke nude.

Igra sa bitovima

Cesto ćete imati potrebu da postavite flag-ove u Vašim objektima, kako biste mogli da pratite njihovo stanje (Da li smo u stanju uzbune? Da li je ovaj objekat inicijalizovan? Da li dolazimo ili odlazimo?).

Ovo možete uraditi uz pomoć korisnički definisanih Booleana, ali, kada imate veći broj flag-ova, zgodnije je znati "baratati" individualnim bitovima kao flag-ovima.

Svaki bajt ima osam bitova. Stoga, u **4**-bajtnom `long-u` možete držati 32 odvojena flag-a. Za bit se kaže da je postavljen, ako je njegova vrednost 1, ili da je poništen, ako je njegova vrednost 0. Kada postavljate bitove, Vi im dajete vrednost 1, a kada ih poništavate dajete im vrednost 0. Bijtove možete postavljati i poništavati tako što ćete menjati vrednost promenljive tipa `long`, ali ovo je prilično teško i zbunjujuće.

yMAPOШИMA^ Dodatak C binarno i heksadecimalno obezbeduje vreme dodatne informacije "vezane" za manipulaciju binarnim i heksadecimalnim brojevima.

C++ obezbeduje operacije za rad sa bitovima koje se mogu izvršavati nad pojedinačnim bitovima. Ovo liči na rad sa logičkim operatorima, ali se razlikuje od njega tako da su početnici u programiranju često zbunjeni. Operatori za rad sa bitovima su prikazani u tabeli 21.1.

Tabela 21.1: Operatori la rad $a bitovima

symbol	operator
&	AND
I	OR
	exclusive OR
	complement

Operator AND

Operator AND (&) je jednostruki &, za razliku od logičkog AND, gde je dvostruki &. Kada uradite AND nad dva bita, rezultat je 1, ako su oba bita 1, ali 0, ako je bilo koji od bitova 0. Način na koji ovo možete razumeti je: rezultat je 1, ako je postavljen bit 1 i ako je postavljen bit 2.

Operator OR

Drugi operator za rad sa bitovima je OR (I). Ovo je jednostruka vertikalna crta, za razliku od logičkog OR, koje se prikazuje sa dve vertikalne crte. Kada radite OR nad dva bita, rezultat je 1, ako je bilo koji od njih setovan, ili ako su setovana oba.

Operator ekskluzivno OR

Treći operator za rad sa bitovima je ekskluzivno OR (^{A}). Kada radite ekskluzivnim OR nad dva bita rezultat je 1, ako su bitovi različiti.

Operator komplement

Operator komplement (-) će poništiti svaki bit u broju koji je postavljen i postaviti svaki bit koji je poništen. Ako je tekuća vrednost broja 1010 0011, komplement tog broja je 0101 1100.

Postavljanje bitova

Kada želite da postavite, ili da poništite određeni bit, koristite operacije maskiranja. Ako imate četvorobajtni flag i želite da postavite bit 8 na TRUE, potrebno je da uradite OR flag-a sa vrednošću 128. Zašto? 128 je 1000 0000 u binarnom sistemu. Stoga, vrednost ovih osam bitova je 128. Bez obzira koja je tekuća vrednost tog bita (postavljen, ili poništen), ako uradite OR sa vrednošću 128, Vi ćete postaviti taj bit i nećete promeniti preostale bitove. Pretpostavimo da je tekuća vrednost 1010 0110 0010 0110. Pomoću 0R-a sa 128 ovo će izgledati ovako:

```
9 8765 4321
1010 0110 0010 0110   // bit 8 je ubrisan
j    0000 0000 1000 0000   // 128

1010 0110 1010 0110   // bit 8 je postavljen
```

Postoji nekoliko detalja na koje treba obratiti pažnju. Kao prvo, kao što je i uobičajeno, bitovi se broje sdesna nalevo. Kao drugo, vrednost 128 se sastoji od svih nula, osim u bitu 8, tj. u bitu koji želite da postavite. Kao treće, broj 1010 0110 0010 0110 ostaje nepromenjen primenom OR operatora, osim što mu se postavlja osmi bit.

Poništavanje bitova

Ako želite da poništite osmi bit, to možete učiniti AND funkcijom i komplementom od 128. Komplement od 128 je broj koji ćete dobiti kada uzmete bit šemu za broj 128 (1000 0000), postavite svaki bit koji je poništen i poništite svaki bit koji je postavljen (0111 1111). Kada izvršite AND sa ovim brojem, originalni broj će biti nepromenjen, osim što će osmi bit biti poništen.

```
1010 0110 1010 0110  // bit 8 je postavljen
& 1111 1111 0111 1111  // Č128

1010 0110 0010 0110  // bit 8 je ubrisan
```

Da biste potpuno razumeli kako ovo funkcionise, potrebno je da malo eksperimentisete. Svaki put kada su oba bita 1, upi Si te 1 u odgovor. Kada je bilo koji od bitova 0, upišite 0 u odgovor. Uporedite Vaš odgovor sa originalnim brojem. On bi morao da bude isti, osim što je osmi bit poništen.

"Prevrtanje" bitova

Na kraju, ako želite da "prevrnete" bit 8 (promenite mu vrednost u suprotnu), bez obzira u kom stanju se nalazi, izvršićete ekskluzivni OR tog broja sa 128. Stoga'

```
1010 0110 1010 0110  // broj
^ 0000 0000 1000 0000  // 128

1010 0110 0010 0110  // invertovan bit
```

```
n 0000 0000 1000 0000  // 128

1010 0110 1010 0110  // invertovan u prvobitno stanje
```

4g PAOTI *LI* Postavljajte bitove, tako sto ćete koristiti maske i OR operator.

Poništavajte bitove, tako što cete koristiti maske i AND operator.

Menjajte vrednost bitova, tako što ćete koristiti maske i ekskluzivni OR operator.

Binarna polja

Postoje situacije u kojima se svaki bajt računa i ušteda 5-6 bajtova u klasi može dovesti do velikih poboljšanja. Ako Vaša klasa, ili struktura ima seriju logičkih promenljivih, ili promenljive mogu da imaju sasvim mali broj mogućih vrednosti, možete uštedeti na prostoru, korišćenjem binarnih polja.

Korišćenjem standardnih tipova C + + podataka, najmanji tip koji možete koristiti u Vašoj klasi je tip char, čija je dužina 1 bajt. Često ćete koristiti polja tipa i nt, koja su dugačka dva, ili, često, četiri bajta. Korišćenjem binarnih polja, možete smestiti osam binarnih vrednosti u char, ili 32 takve vrednosti u long.

Evo kako binarna polja funkcionišu: binarna polja se imenuju i pristupa im se kao bilo кфјет drugom članu klase. Njihov tip se uvek deklariše kao unsigned int. Posle imena binarnog polja napišite :, koje slede broj i instrukcija kompajleru koliko bitova da dodeli toj promenljivoj. Ako napišete 1, bit će predstavljati vrednost 0, ili 1. Ako napišete 2, bit će predstavljati 0, 1, 2, ili 3, tj. Četiri moguće vrednosti. Trobitna polja mogu da predstave osam vrednosti i tako dalje. U dodatku C prikazani su binarni brojevi. U listingu 21.8 ilustrovano je koriscenje binarnih polja.

Listing 21.8: Koriscenje bitnih polja.

```
linclude <iostream.h>
linclude <string.h>

enum STATUS { Full Time, PartTime } ;
enum GRADLEVEL { UnderGrad, Grad } ;
enum HOUSING { Dorm, OffCampus };
enum F00DPLAN { OneMeal, AllMeals, WeekEnds, NoMeals

class student
{
 public:
 student():
      myStatus(FullTime),
      myGradLevel(UnderGrad),
      myHousing(Dorm),
      myFoodPlan(NoMeals)
```



```
      -studentOi)
      STATUS GetStatusO;
      void SetStatus(STATUS);
      unsigned GetPlanQ { return myFoodPlan; }

 private:
    unsigned myStatus : 1;
    unsigned myGradLevel: 1;
    unsigned myHousing : 1;
    unsigned myFoodPlan : 2;


STATUS student::GetStatus()
{
   if (myStatus)
      return FullTime;
   else
      return PartTime;
}
void student::SetStatus(STATUS theStatus)

   myStatus = theStatus;
}


int main()
{
   student Jim;

   if (Jim.GetStatus()== PartTime)
      cout « "Jim is part time" « endl;
   else

      cout « "Jim is full time" « endl;

   Jim.SetStatus(PartTime);

   if (Jim.GetStatusO)
      cout « "Jim is part time" « endl;
   el se

      cout « "Jim is full time" « endl;

   cout « "Jim is on the " ;

   char Plan[80];
   switch (Jim.GetPlanO)
   {
      case OneMeal: strcpy(Plan,"One meal"); break;
      case AllMeals: strcpy(Plan,"Al1 meals"); break;
      case WeekEnds: strcpy(Plan,"Weekend meals"); break;
```

nastavlja se

listing 21.8: Koriscenje bitnih polja. ^vak

```
66:              case NoMeals: strcpy(Plan,"No Meals") -.break;
67:              default : cout « "Something bad went wrong!\n"; break;
68:           }
69:           cout « Plan « " food plan." « endl;
70:        return 0;
```

```
Jim is Part time
jim is full time
Jim is on the No Meals food plan.
```

IJIUMfc* U linijama 3-7 definisano je nekoliko nabrojanih tipova. Oni služe za definiciju mogućih vrednosti binarnih polja unutar klase student.

Klasa student je deklarisana u linijama 8-27. Iako je trivijalna, ona je interesantna zbog toga što su svi podaci spakovani unutar pet bitova. Prvi bit predstavlja status studenta (redovan, ili vanredan), drugi bit predstavlja podatak da li je student diplomirao, ili ne, treći bit predstavlja podatak da li student živi u domu, ili ne. Poslednja dva bita predstavljaju četiri moguća plana ishrane.
Metode klase su napisane kao i za bilo koju drugu klasu i sama činjenica da se koriste binarna polja, a ne integer-i, uopšte im ne smeta.

Funkcija clan GetStatusO čita odgovarajući logički bit i vraća tip nabrajanja. Ali, ovo nije neophodno. Jednostavno je moglo biti napisano da vraća direktnu vrednost bita. Kompajler bi izvršio translaciju.

Da biste dokazali ovo sami sebi, zamenite implementaciju za GetStatusO sledećim kodom:

```
STATUS student::GetStatus()
{
return myStatus;
}
```

Ne bi trebalo da dode do bilo kakve izmene u funkcionisanju programa. Ovo je samo detalj koji se tiče jasnoće programa, dok je kompajleru potpuno svejedno.

Primetićete da kod u liniji 46 mora da proveri status i da, zatim, prikaže razumljivu poruku. Ovo je pokušaj da se napiše sledeće:

```
cout « "Jim is " « Jim.GetStatus() « endl;
```

Ovo će jednostavno prikazati sledeće:

```
Jim is 0
```

Kompajler nema načina da izvrši translaciju nabrojane konstante `PartTime` u logički tekst.

U liniji 61 program se prebacuje na plan ishrane i za svaku moguću vrednost on smešta razumljivu poruku u bafer, koja se, zatim, prikazuje u liniji 69. Switch naredba mogla je biti napisana na sledeći način:

```
case  0: strcpy(Plan,"One meal"); break;
case  1: strcpy(Plan,"All meals"); break;
case  2: strcpy(Plan,"Weekend meals"); break;
case  3: strcpy(Plan,"No Meals");break;
```

Mnogo važnija okolnost pri korišćenju binarnih polja je, da klijent te klase ne mora da brine o implementaciji podataka. Pošto su binarna polja privatna, možete ih menjati bez potrebe za izmenom interfejsa.

Sfil

Kao što je prikazano u celoj knjizi, veoma je važno prilagoditi se konzistentnom načinu kodiranja, iako, u principu, nije mnogo važno koji stil ćete prihvatiti. Konzistentan stil Vam olakšava da odredite šta ste želeli da uradite u odredenom delu koda.

Sledeće smernice nemojte shvatiti kao zakon; njih sam koristio u projektima koje sam radio i pokazale su se kao uspešne. Vi možete napraviti sopstvene, ali bih Vam preporučio da krenete od ovih koje ću navesti.

Iako je Emerson rekao "glupa konzistencija je bauk za plitke mozgove", ipak je postojanje konzistencije u Vašem kodu ispravno. Na Varna je da odlučite, ali ovakav pristup može učiniti samo dobro Vašem programiranju.

Identacija

Veličina tabova bi trebalo da bude četiri prazna mesta. Proverite da li Vaš editor konvertuje svaki Vaš tab u četiri prazna mesta.

Vitičaste zagrade

S obzirom da je poravnanje vitičastih zagrada najkontraverznija tema za diskusiju izmedu C i C+ + programera, Slobodan sam da Vam ponudim nekoliko predloga:

- Odgovarajuće vitičaste zagrade bi trebalo da budu vertikalno poravnate.
- Set vitičastih zagrada, najbliži početku za definicije, ili deklaracije, trebalo bi da se nalazi na levoj margini. Unutrašnje naredbe bi trebalo da budu indentirane. Svi ostali setovi vitičastih zagrada bi trebalo da budu u liniji sa njihovim vodećim naredbama.
- Ni jedan kod ne bi smeo da se pojavi u istoj liniji sa vitičastom zagradom. Na primer:

```
if  (condition==true)
{
j = k;
SomeFunctionO;
}
m++;
```

Dugačke linije

Održavajte dužinu linije, tako da ne bude veća od širine ekrana. Kod koji se nalazi previše desno teže se uocava, a horizontalno skrolovanje je samo gubljenje vremena. Kada prekinete liniju identujte sledeće linije. Pokušajte da prekinete liniju na razumnom mestu, tako da bude jasno da linija nije samostalna i da iza nje sledi još nešto.

U C++-U funkcije teže da budu kraće nego što su bile u C-u. Ali i prethodni savet i dalje važi. Pokušajte da Vaše funkcije budu dovoljno kratke, kako bi cela funkcija stala na jednu stranu.

Naredbe switch

Indentujte switch-eve, kao što sledi, kako biste sačuvali horizontals prostor.

```
switch(variable)
{
case ValueOne:
ActionOneO;
break;
case ValueTwo:
ActionTwo();
break;
default:
assert("bad Action");
break;
}
```

Programski tekst

Postoji nekoliko saveta za kreiranje koda koji je jednostavan za čitanje. Kod jednostavan za dtanje je jednostavan i za održavanje:

- Koristite prazne karaktere, kako biste poboljšali čitljivost.
- Objekti i nizovi, u stvari, ukazuju na jedno: nemojte koristiti prazna mesta unutar referenci objekta (.,->, []).
- Unarni operatori su dodeljeni njihovim operandima. Stoga ih nemojte razdvajati praznim karakterom. Stavite prazno mesto iza operanda. Unarni operatori su !, ++, - -,* (za pointere), &, sizeof.
- Binarni operatori bi trebalo da imaju prazna mesta sa obe strane: +,=,*, /, %, », «,<,>,==, !=, &, !,&&, !!,?:,+=, itd.
- Nemojte koristiti prazna mesta, da biste označili prednosti izvršenja operacija (4+3*2).
- Stavite prazan karakter iza zapeta i tačka-zapeta, a ne pre njih.
- Zagrade ne bi trebalo da imaju prazna mesta sa bilo koje strane.
- Ključne reči, kao što je if, trebalo bi da budu odvojene praznim mestima: if (a = = b)
- Telo komentara bi trebalo da bude odvojeno od // praznim mestom.
- Smestite pointer, ili indikator reference odmah do imena tipa, a ne do imena promenljive:

```
char* foo;
int& theInt;
```

umesto

```
char *foo;
int &theInt
```

- Nemojte deklarisati vise od jedne promenljive u jednoj liniji.

Imena identifikatora

Evo nekoliko smernica za rad sa identifikatorima:

- Imena identifikatora bi trebalo da budu dovoljno dugačka, da bi bila deskriptivna.
- Izbegavajte skriptične skraćenice.
- Uložite vreme i energiju, kako biste opisali identifikatore.
- Nemojte koristiti Madarsku notaciju. C++ ima veoma definisane tipove i nema potrebe da smeštate tip unutar promenljive. Sa korisnički definisanim tipovima (klasama), Madarska notacija vrlo brzo postaje neupotrebljiva. Izuzetak može biti koriscenje prefikasa za pointere (p) i reference (r), kao i za promenljive clanove klase (its).
- Kratka imena (i, p, x) mogu se samo koristiti u delovima koda, u kojima će dovesti do čitljivosti i u kojima je njihovo koriscenje toliko očigledno da deskriptivno ime i nije potrebno.

/

- Dužina imena promenljive bi trebalo da bude proporcionalna opsegu u kojem ona važi.
- Neka identifikatori izgledaju i zvuče različito jedni od drugih, kako biste izbegli konfuziju.
- Imena funkcija, ili metoda su, obično, glagoli: `search()`, `reset()`, `findParagraph()`, `ShowCursor()`. Imena promenljivih su, obično, apstraktne imenice: `count`, `state`, itd. Logičke promenljive bi, takođe, trebalo da imaju odgovarajuća imena: `FilelsOpen`, `Windowlconized`.

Spelovanje i kapitalizacija imena

Probleme spelovanja i kapitalizacije imena ne biste smeli da izostavite kada kreirate Vaš sopstveni stil. Evo nekoliko saveta:

- Koristite velika slova i donju crticu za razdvajanje logičkih red u imenima, kao na primer, `S0URCE_FILE_TEMPLATE`. Medutim, primetićete da je ovo retkost u C++ -u. Razmislite o korišćenju konstanti i templejta u većini slučajeva.
- Svi drugi identifikatori bi trebalo da budu kombinovani od velikih i malih slova, i to bez donjih crtica. Imena funkcija, metoda, klasa, typedef i imena struktura bi trebalo da počinju velikim, a elementi, kao što su podaci članovi, malim slovom.
- Konstante nabrajanja bi trebalo da počnu sa nekoliko malih slova, koja su skraćenica za enum. Na primer

```
enum TextStyle
{
    tsPlain,
    tsBold,
    tsItalic,
    tsIInderscore,
```

Komentari

Komentari olakšavaju dtanje programa. Ponekad, Vi nećete raditi na programu nekoliko dana, ili, čak, meseci. Za to vreme možete zaboraviti šta odredeni kod radi, ili zašto je on tu bio uključen. Problemi u razumevanju koda se, takođe, mogu javiti kada neko drugi čita Vaš program. Komentari koji su konzistentno primenjeni mogu Vam pomoći u radu. Evo nekoliko saveta koje treba zapamtiti:

- Kad god je moguće, koristite C++ // komentare, umesto, /* */ komentara.
- Komentari višeg nivoa su daleko važniji od detalja procesa. Imajte meru. Nemojte nepotrebno opterećivati kod.


```
n++; // n se inkrementira za jedan
```

Ovaj komentar ne vredi vremena potrebnog da se ukuca. Koncentrišite se na semantiku funkcija i blokove koda. Recite šta funkcija radi. Ukažite na propratne efekte, tipove parametara i vraćene vrednosti. Opišite sve što ste podrazumevali (ili niste), na primer, "podrazumeva se da n nije negativno", ili "vratiče **-1,** ako je x neispravno". Unutar kompleksne logike koristite komentare koji će indicirati uslove u toj tački koda.

- Koristite kompletne rečenice sa odgovarajućom interpunkcijom i odgovarajućom kapitalizacijom slova. Nemojte biti kriptični i nemojte praviti nepotrebna skraćivanja. Ono što Vam se u jednom trenutku može učiniti sasvim jasno, verovatno će biti potpuno nejasno i začuđujuće za nekoliko meseci.
- Slobodno koristite prazne linije, kako biste pomogli čitaocu da shvati šta se u programu dogada. Odvojte naredbe u logičke grupe.

Pristup

Nadn na koji se pristupa delovima Vašeg programa takođe bi morao da bude konzistentan. Evo nekoliko saveta:

- Uvek koristite public:, private: i protected: labele. Nemojte ostavljati podrazumevane vrednosti.
- Prvo izlistajte javne clanove, zatim zaštićene, a tek onda privatne. Izlistajte podatke danove u grupama posle metoda.
- Smestite konstruktore u odgovarajući odeljak posle destruktora. Izlistajte metode nad kojima ste izvršili overload, sa istim imenom, priključene jedne uz drugu. Definišite funkcije pristupa kad god je to moguće.
- Pokušajte da metode i promenljive članove prikažete u alfabetskom redosledu. Takođe, to učinite i za imena datoteka i include naredbe.
- Iako je koriscenje virtuelnih ključnih red opciono, kada ste uradili override, ipak ih koristite; ovo će Vam pomoći da se prisetite da je ona virtuelna i takođe će Vam pomod da deklaracije budu konzistentne.

Definicije klase

Pokušajte da održite definicije metoda u istom redosledu kao i deklaracije. Ovim se funkcije lakše pronalaze.

Kada definišete funkciju, smestite tip podatka koji će biti vraćen i sve ostale modifikatore u prethodnu liniju, tako da ime klase i ime funkcije podnju na levoj margini. Ovim ćete mnogo lakše pronaći funkciju.

include datoteke

Trudite se, koliko god je to moguće, da što manje uključujete datoteke u Vaš program. Idealni minimum je datoteka zaglavlja za klasu iz koje se ona izvodi. Ostali obavezni include-ovi su oni za objekte koji su članovi deklarisane klase.

Nemojte ostavljati I nel udef i 1 e u zaglavlju samo zato što ste podrazumevali da svaka CPP datoteka, koja uključuje Vašu, zahteva da ukljudte Include.

^ savtt *IJ* Sve datoteke zaglavlja bi trebalo da koriste zašititu uključivanja.

assert()

Slobodno koristite assert(). Ona Vam pomaže pri pronalaženju grešaka, ali je, takode, od velike pomoći čitaocu, tako što olakšava pronalaženje funkcija u samom programu. Ona Vam takođe pomaže da se usredsredite na to šta je pisac smatrao za ispravno, a šta za netačno.

const

Koristite const kad kod to ima smisla: za parametre, promenljive i metode. Cesto postoji potreba i za const i za non-const verzijom metode. Nemojte ovo koristiti kao razlog za izostavljanje jedne od njih. Budite veoma pažljivi kada eksplicitno menjate ulogu iz const u non-const u bilo kojem pravcu i obrnuto (u stvari, nekada je ovo jedini nadn da biste nešto udnili), ali budite uvereni da to ima smisla i ukljudte komentar.

Sledeći koraci

Upravo ste potrošili tri duge i teške nedelje u radu sa C++ i sada ste kompetentni C++ programer, ali ovo nije kraj. Postoji još mnogo toga što bi trebalo da naudte, još mnogo mesta na kojima možete pronaći vredne informacije, kako se budete kretali od početnika u C++ do eksperta.

Sledeći odeljci će Vam preporudti nekoliko specifičnih izvora informacija i te preporuke samo odražavaju moje lično iskustvo i mišljenje. Postoje na desetine knjiga za svaku od ovih, ali, raspitajte se pre nego što porudte bilo koju od njih.

Gde naci pomoć i savet

Prvo što ćete želeti da uradite kao C++ programer verovatno će biti da se prikljudte nekoj od C++ konferencija na nekom od svetskih servisa. Ove grupe obezbeduju trenutni kontakt sa stotinama, ili hiljadama C++ programera, koji Vam mogu odgovoriti na Vaša pitanja i ponuditi Vam korisne savete.

Ja sam učlanjen u C++ InternetNewsGroup (comp. 1 ang.c++ i comp.lang.C++.moderated) i predlažem ih kao izvanredan izvor informacija i podrške.

Takođe, možete potražiti i lokalne grupe korisnika. Mnogi gradovi imaju C++ grupe, u kojima se možete upoznati sa drugim programerima i razmeniti ideje.

Ostanimo u kontaktu

11 Ako imate komentare, sugestije, ili ideje o ovoj, ili nekoj drugoj, knjizi, pišite mi na jliberty@l ibertyassociates.com, ili pogledajte moj Web sajt:

www.1ibertyassociates.com.

4|J paztti;|£> Pogledajte druge knjige. Postoji mnogo štošta što se može naučiti i ne postoji jedinstvena knjiga koja Vas može naučiti svemu onome sto je potrebno da znate.

Nemojte samo čitati programe! Najbolji način da naucite C++ je da pišete C++ programe.

Pretplatite sa na dobar C++ časopis i pridružite se dobroj C++ grupi korisnika.

Rezime

Danas ste saznali kako se mogu koristiti neke od standardnih biblioteka, koje se isporučuju sa Vašim C++ kompajlerom, za obavljanje nekih rutinskih zadataka. StrcpyO, stri en () i slične funkcije se mogu koristiti za manipulaciju Null-terminiranim stringovima. Iako ovo neće funkcionisati sa string klasama koje ćete kreirati, uvidećete da one obezbeduju esencijalnu funkcionalnost u implementaciji Vaših vlastitih klasa.

Funkcije "vezane" za datum i vreme Vam omogućavaju da manipulišete vremenskim strukturama. One se mogu koristiti za pristup sistemskom vremenu u Vašim programima, ili za manipulaciju vremenskim i datumskim objektima koje kreirate.

Takođe ste naudli kako da postavljate i testirate pojedinačne bitove i kako da alocirate ograničeni broj bitova za danove klase.

Na kraju, objašnjeni su stilovi u C++ -u i resursi koji postoje i koji su Vam na raspolaganju u daljem učenju.

Pitanja i odgovori

P Zašto se standardne biblioteke isporučuju sa C++ kompajlerima i kada ih treba koristiti?

O Biblioteke su uključene zbog povratne kompatibilnosti sa C-om. One nisu sigurne u pogledu tipova i ne funkcionišu ispravno sa korisnički kreiranim klasama, tako da se koriste vrlo ograničeno. Vremenom, možete očekivati da će sva njihova funkcionalnost migrirati u specifične C++ biblioteke, kada će standardne C biblioteke polako nestati.

P **Kada koristiti binarne strukture umesto integer-a?**

O Kada je veličina objekta od krucijalne važnosti. Ako radite sa ograničenom memorijom, ili sa komunikacionim softverom, verovatno će Vam biti potrebne uštede, koje nude ove strukture, a esencijalne su za uspeh Vaših proizvoda.

P **Zašto rat stilova izaziva toliko emocija?**

O Programeri su vrlo osetljivi na svoje navike. Ako ste koristili sledeću identaciju

```
if (SomeConditionM
    // iskazi
}    // zatvarajuća zagrada
```

teško ćete je se odreći. Novi stilovi deluju pogrešno i dovode do konfuzije. Ako Vam je dosadno, uključite se na neki od popularnih servisa i upitajte, koji stil identacije bolje funkcioniše; koji editor je najbolji za C++; ili koji proizvod je najbolji Word procesor. Zatim se lepo uvalite u svoju stolicu i posmatrajte na stotine poruka, medusobno kontradiktornih, kao i svadu koja će iz toga nastati.

Kviz

1. Koja je razlika izmedu strcpyO i strncpy()?
2. Šta radi ctimeO?
3. Koju funkciju ćete pozvati da biste izvršili konverziju ASCII stringa u 1 ong?
4. Šta radi komplement operator?
5. Koja je razlika između OR i exclusive OR?
6. Koja je razlika između & i &&?
7. Koja je razlika izmedu J i |' ?

Vežbe

1. Napišite program koji će na siguran način kopirati sadržaj 20-bajtnog stringa u 10-bajtni string, odsecajući višak.
2. Napišite program koji će tekući datum prikazati u formi 7/28/94.
3. Napišite program koji će kreirati 26 flag-ova (pod nazivom A-Z). Zatražite od korisnika da unese rečenicu i zatim ga obavestite koja slova su korišćena, tako što ćete setovati, a, zatim, iščitati flag-ove.
4. Napišite program koji će sabrati dva broja, bez korišćenja operatora + za sabiranje. Savet: Koristite bit operatore!

Pregled sadržaja

Sledeći program povezuje većinu naprednih tehnika o kojima ste učili tokom prethodne tri nedelje teškog rada. Kratak pregled sadržaja Nedelje 3 sadrži na templejtima bazirane povezane liste, sa obradom izuzetaka. Detaljno ga ispitajte; ako ga stvarno razumete, smatrajte da ste C++ programer!

vUPOZORIHJII Ako Vaš kompajler ne podržava try i catch blokove, nećete biti u mogućnosti da kompajlirate ili startujete ovaj listing.

Listing PS3.1: Pregled sadriaja Sedmice 3

```
0:  //  **********************************************
1:  //
2:  //  Title:      Week 3 in Review
3:  //
4:  //  File:       Week3
5:  //
6:  //  Description:   Provide a template-based linked list
7:  //                 demonstration program with exception handling
8:  //
9:  //  Classes:    PART - holds part numbers and potentially other
10: //                     information about parts. This will be the
11: //                     example class for the list to hold
12: //                     Note use of operator« to print the
13: //                     information about a part based on its
14: //                     runtime type.
15: //
```

nastavlja se

Listing PS3.1: Pregled sadiiajo Sedmice 3

nastavak

```
//                Node - acts as a node in a List
//
//                List - template-based list which provides the
//                       mechanisms for a linked list
//
//
//
// Author:      Jesse Liberty (jl)
//
// Developed:   Pentium 200 Pro. 128MB RAM MVC 5.0
//
// Target:      Platform independent
//
// Rev History:   9/94 - First release (jl)
//                4/97 - Updated (jl)
Ij      ************************************************

#include <iostream.h>

// exception classes
class Exception {};
class OutOfMemory :    public Exception!};
class NullNode :     public Exception!};
class EmptyList :    public Exception {};
class BoundsError :    public Exception {};


Ij  ***************  part  ************
// Abstract base class of parts
class Part

public:
 Part():itsObjectNumber(l)  {}
 Part(int  ObjectNumber):itsObjectNumber(ObjectNumber){}
 virtual  -Part(){};
 int GetObjectNumberO  const  { return itsObjectNumber; }
 virtual  void  DisplayO  const =0;   // must be overridden

private:
 int  itsObjectNumber;


// implementation  of  pure  virtual  function  so  that
// derived classes can chain up
void Part::Display()  const

   cout  «  "ĐnPart  Number:  "  «  itsObjectNumber  «  endl;
```

```
64:   // this one operator« will be called for all part objects.
65:   // It need not be a friend as it does not access private data
66:   // It calls DisplayO which uses the required polymorphism
67:   // We'd like to be able to override this based on the real type
68:   // of thePart, but C++ does not support contravariance
69:   ostreami operator«( ostreami theStream,Part& thePart)
70:   {
71:    thePart.DisplayO;   // virtual contravariance!
72:    return theStream;
73:   }
74:
75:   II  ***************        Part  ************
76:   class CarPart : public Part
77:   {
78:   publi c:
79:    CarPart():i tsModelYear(94){}
80:    CarPart(int year,  int partNumber);
81:    int GetModelYear()  const  { return itsModeiYear; }
82:    virtual  void  DisplayO  const;
83:   private:
84:    int  itsModelYear;
85:
86:
87:   CarPart::CarPart(int year,  int partNumber):
88:    itsModelYear(year),
89:    Part(partNumber)
90:   {}
91:
92:   void CarPart::Display()  const
93:   {
94:       Part::Display();
95:       cout  «  "Model  Year:  "  «  itsModelYear  «  endl;
96:
97:
98:   II  ***************  д-jrPiane  Part  ************
99:   class AirPlanePart : public Part
100:   {
101:   public:
102:    AirPlanePartO:itsEngineNumber(1) {};
103:    AirPlanePart(int EngineNumber,  int PartNumber);
104:    virtual  void  Display!)  const;
105:    int GetEngineNumber()const  { return itsEngineNumber;
106:   private:
107:    int  itsEngineNumber;
108:
109:
110:   AirPlanePart::AirPlanePart(int EngineNumber,  int  PartNumber):
111:    itsEngineNumber(EngineNumber),
112:    Part(PartNumber)
```

nastavlja se

Listing PS3.1: Pregled sadiiaio Sedmice 3

nastavak

```
void AirPlanePart::DisplayO const
(
 Part::Display();
 cout « "Engine No.      « itsEngineNumber « endl;
}

// forward declaration of class List
template <class T>
class List;

// ***************  Node ************
// Generic node, can be added to a list
/j       ************************************


template <class T>
class Node
{
public:
  friend class List<T>;
  Node (T*);
  -Node();
  void SetNext(Node * node) { itsNext = node; }
  Node * GetNext() const;
  T * GetObjectO const;
private:
  T* itsObject;
  Node * itsNext;
};

// Node Implementations...

template <class T>
Node<T>::Node(T* pOjbect):
itsObject(pOjbect),
itsNext(0)


template <class T>
Node<T>::-Node()
{
  delete itsObject;
  itsObject = 0;
  delete itsNext;
  itsNext = 0;
```



```
// Returns NULL if no next Node
template <class T>
Node<T> * Node<T>::GetNext 0 const
{
     return itsNext;
}

template <class T>
T * Node<T>::GetObjectO const
{
 if  (itsObject)
     return itsObject;
 else
     throw NullNodeO;
}

jl ***************  List ************
// Generic list template
// Works with any numbered object
lj       ************************************
template <class T>
class List
{
public:
  List();
  -List();

  T*         Find(int & position, int ObjectNumber)  const;
  T*       GetFirstO const;
  void       Insert(T *);
  T*        operator[] (int) const;
  int       GetCountO const { return itsCount; }
private:
  Node<T> * pHead;
  int        itsCount;
};

// Implementations for Lists...
template <class T>
List<T>::Li st():
 pHead(O),
 itsCount(O)


template <class T>
List<T>::~List()
{
 delete pHead;
```

nastavljase

Listing PS3.1: Pregled sadriaja Sedmice3

```
210:
211:   template <class T>
212:   T*    List<T>::GetFirst() const
213:   {
214:     if (pHead)
215:        return  pHead->itsObject;
216:     else
217:        throw  EmptyListQ;
218:   }
219:
220:   template <class T>
221:    T *   List<T>::operator[](int offSet)  const
222:   {
223:    Node<T>* pNode = pHead;
224:
225:     if (IpHead)
226:        throw  EmptyListQ;
227:
228:     if (offSet > itsCount)
229:        throw BoundsError();
230:
231:     for (int i=0;i<offSet; i++)
232:        pNode = pNode->itsNext;
233:
234:    return    pNode->itsObject;
235:   }
236:
237:    // find a given object in list based on its unique number (id)
238:   template <class T>
239:    T*    List<T>::Find(int & position, rnt ObjectNumber)   const
240:   {
241:     Node<T> * pNode = 0;
242:     for (pNode = pHead, position = 0;
243:           pNode!=NULL;
244:           pNode = pNode->itsNext, position++)
245:     {
246:        if  (pNode->its0bject->Get0bjectNumber()  ==  ObjectNumber)
247:           break;
248:     }
249:     if (pNode  ==  NULL)
250:        return NULL;
251:     else
252:        return  pNode->itsObject;
253:   }
254:                                                                    \
255:    // insert if the number of the object is unique
256:    template <class T>
257:    void List<T>::Insert(T* pObject)
```

```
    Node<T> * pNode = new Node<T>(p0bject);
    Node<T> * pCurrent = pHead;
    Node<T> * pNext = 0;

    int New =   p0bject->Get0bjectNumber();
    int Next = 0;
    itsCount++;

    if  (IpHead)
    {
       pHead = pNode;
       return;


    // if this one is smaller than head
    // this one is the new head
    if  (pHead->itsObject->GetObjectNumber()  >  New)
    {
       pNode->itsNext = pHead;
       pHead = pNode;
       return;


    for   (;;)
    {
       // if there is no next, append this new one
       if   (!pCurrent->itsNext)
       {
          pCurrent->itsNext = pNode;
          return;


       // if this goes after this one and before the next
       // then insert it here, otherwise get the next
       pNext = pCurrent->itsNext;
       Next = pNext->itsObject->GetObjectNumber();
       if  (Next > New)
       {
          pCurrent->itsNext = pNode;
          pNode->itsNext = pNext;
          return;
       }
       pCurrent = pNext;



int  main()
```

nastavlja se

Listing PS3.1: Pregled sadriaja Sedmice 3

```
{
 List<Part> theList;
 int choice;
 int ObjectNumber;
 int value;
 Part * pPart;
 while (1)
 {
    cout « "(O)Quit (l)Car (2)Plane: ";
    cin » choice;

    if (!choice)
       break;

    cout « "New PartNumber?: ";
    cin » ObjectNumber;

    if (choice == 1)
    {
       cout « "Model Year?:
       cin » value;
       try

          pPart = new CarPart(value,ObjectNumber)
       }
       catch (OutOfMemory)
       {
          cout « "Not enough memory; Exiting..." « endl;
          return 1;
       }

    else
    {
       cout « "Engine Number?: ";
       cin » value;
       try
       {
          pPart = new AirPlanePart(value,ObjectNumber);
       }
       catch (OutOfMemory)
       {
          cout « "Not enough memory; Exiting..." « endl;
          return 1;


    try

       theList.Insert(pPart);
```

nastavak

```
    catch (NullNode)

       cout « "The list is broken, and the node is null!" « endl;
       return 1;

    catch (EmptyList)

       cout « "The list is empty!" « endl;
       return 1;


 try
 {
    for (int i = 0; i < theList.GetCount(); i++ )
       cout « *(theList[i]);

    catch (NullNode)

       cout « "The list is broken, and the node is null!" « endl;
       return 1;

    catch (EmptyList)

       cout « "The list is empty!" « endl;
       return 1;

    catch (BoundsError)

       cout « "Tried to read beyond the end of the list!" « endl;
       return 1;

return 0;
}
```

```
(O)Quit (l)Car (2)Plane: 1
New PartNumber?: 2837
Model Year? 90

(O)Quit (l)Car (2)Plane: 2
New PartNumber?: 378
Engine Number?: 4938

(O)Quit (l)Car (2)Plane: 1
New PartNumber?: 4499
Model Year? 94

(O)Quit (l)Car (2)Plane: 1
New PartNumber?: 3000
Model Year? 93
```

```
(O)Quit (I)Car (2)Plane: 0

Part Number: 378
Engine No. 4938

Part Number: 2837
Model Year: 90

Part Number: 3000
Model Year: 93

Part Number 4499
Model Year: 94
```

fJSElte Ovaj listing je modifikovani program iz Nedelje 2, kome su dodati templejti, ostream obrada i obrada izuzetaka. Inače, izlaz je identičan.

U linijama 35-39 deklarisan je veći broj klasa izuzetaka.Obezbedena je primitivna obrada izuzetaka, tako da ne postoje ni podaci, ni metode za ove izuzetke. Oni se ponašaju kao flag-ovi za catch naredbe koje prikazuju jednostavna upozorenja, a zatim prestaju sa radom. Robusniji program bi trebalo da prosledi ove izuzetke po referenci, kao i dodati sadržaj i druge podatke iz objekata izuzetaka pri pokusaju da se izvuče iz problema.

U liniji 44 deklarisana je apstraktna bazna klasa Part, baš kao i u Nedelji 2. Jedina interesantna izmena je, da ovde postoji operator «( ) , koji je deklarisan u linijama 69-73. Primetite da on nije clan klase Part, niti prijatelj. On jednostavno uzima part referencu kao jedan od svoji argumenata.

Možda ćete želeti da imate operator« koji uzima CarPart i Ai rplanePart, u nadi da će ispravni operator« biti pozvan, u zavisnosti od toga da li je prosleden deo za kola ili za avion.

S ozbirom da program prosleduje pointer na deo, a ne pointer na deo kola i deo aviona, C+ + će morati da pozove odgovarajuću funkciju, u zavisnosti od stvarnog tipa argumentra funkcije. Ovo se naziva contravariance i nije podržano u C + + .

Postoje samo dva načina za primenu polimorfizma u C+ + : polimorfizan funkcija i virtuelne funkcije. Polimorfizam funkcija neće ovde funkcionisati, pošto u svakom slučaju imate isti potpis: onaj koji ukazuje je Part.

Virtuelne funkcije takođe neće raditi ovde, pošto operator « nije funkcija clan klase Part. Operator« ne može biti funkcija clan klase Part, pošto Vi želite da pozovete

```
cout « thePart
```

a što znači da bi stvaran poziv bio (count.operator«Part&) a count nema verziju za operator « koji uzima Part referencu.

Da biste prevazišli ova ograničenja program Nedelje 3 koristi samo jedan operator« koji uzima referencu na Part. On zatim poziva DisplayO koja je virtuelna funkcija član i stoga je pozvana ispravna verzija.

U linijama 129-142, Node je definisano kao templejt. Ono izvršava istu funkciju kao i Node u Nedelji 2, ali ova verzija Node nije povezana sa Part objektom. Ona može, u stvari, biti "čvor" za bilo koji tip objekta.

Primetite da, ako pokušate da dobijete objekat iz Node, a ne postoje objekti, to će dovesti do pojave izuzetke u liniji 174.

U linijama 181-197 definisana je, uz pomoć templejta, generička Li st klasa. Ova Li st klasa može da sadrži čvorove za bilo koji objekat koji ima jedinstven identifikacioni broj i ona ih čuva u sortiranom rastućem redosledu. Svaka od ovih list funkcija proverava situacije u kojima može doći do izuzetaka i obraduje ih na odgovarajući način.

U linijama 306-388 drajver program kreira listu dva tipa Part objekata i zatim prikazuje vrednosti objekata u liste, korišćenjem standardnog stream mehanizma.

Prioriteti operatora

Važno je razumeti da operatori imaju svoje prioritete.

Ш 2 Ш *Prioritet* je redosled u kojem program izvršava operacije u formuli. Ako jedan operator ima ved prioritet od drugog, on će se izvršiti prvi.

Operatori višeg prioriteta "jače povezuju" od operatora nižeg prioriteta. Stoga se operatori višeg prioriteta prvi izvršavaju. Što je manji rang u sledećoj tabeli, viši je prioritet:

Tabela A.l: Prioritet operatora

Rang	Ime	Operator
1	rezolucija opsega	
2	izbor člana, subscript, pozivi funkcije, increment i decrement	-> 0 ++
3	si ze of, prefiks increment i decrement, komplement, and, not, unarni minus i plus, adresa dereference, new, new[], delete, deleteQ, casting, sizeoff)	!- + & *()
4	izbor člana za pointer	
5	množenje, deljenje, moduo	* /
6	sabiranje, oduzimanje	
7	shift	
8	nejednakosti	< < = > > =
9	jednakost, nejednakost	
10	AND za rad sa bitovima	
11	ekskluzivno OR za rad sa bitovima	
12	OR za rad sa bitovima	

nastavlja se

Tabela A.I: Prioritet operatora *astavak*

Rang	Ime	Operator
13	logičko AND	&&
14	logicko OR	!!
15	uslov	?:
16	operatori dodeljivanja	= *= /= %= + = « = &= != ^=
17	throw operator	throw
18	zapeta	,

Kljucne reči C+ +

Ključne reči su rezervisane za kompajler i koristi ih jezik. Ne možete definisati klase, promenljive, ili funkcije koje imaju ove ključne red za svoja imena. Lista je relativno diskutabilna, s obzirom da su neke od tih red specifične za odredene kompajlere. Stoga, lista u Vašem slučaju može neznatno da varira:

auto
break
case
catch
char
cl ass
const
conti nue
default
delete
do
double
el se
enum
extern
float
for
friend
goto
if
int
1 ong
mutable
new
operator
private
protected

```
public
register
return
short
signed
sizeof
static
struct
switch
template
this
throw
typedef
union
unsigned
virtual
void
volatile
whi1e
```


Binarni i heksadecimalni sistem

Osnove aritmetike ste naučili davno i teško bi bilo zamisliti na šta bi sve ovo ličilo bez tog znanja. Kada pogledate broj 145, Vi momentalno prepoznajete "stočetrdesetpet", i to bez ikakvog razmišljanja.

Razumevanje binarnih i heksadecimalnih brojeva zahteva ponovno ispitivanje broja 145 i videnja da to i nije broj, nego kod za broj.

Krenite sa malim: ispitajte odnos između broja tri i "3". Cifra 3 je "vijuga" na parčetu hartije; dok je broj tri ideja. Cifre se koriste za predstavljanje brojeva.

Razlika može biti jasnija kada shvatite da tri, 3, ###, III i *** može biti upotrebljeno za predstavljanje iste ideje o broju tri.

U osnovi 10 (decimalno) koriste se cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 za predstavljanje svih brojeva. Kako se predstavlja broj deset?

Možete zamisliti strategiju koja će koristiti slovo A za predstavljanje broja deset, ili možete koristiti IIIIIIIIII za predstavljanje ove ideje. Rimljani su koristili X. Arapski sistem, koji i mi koristimo, upotrebljava pozicije u kombinaciji sa ciframa, da bi predstavio vrednosti. Prva (krajnja desna kolona) se koristi za jedinice, dok se sledeća kolona koristi za desetice. Stoga se broj petnaest predstavlja kao 15, što je jedna desetica i pet jedinica.

Iz ovoga je moguće izvesti neke generalne zaključke:

1. Baza deset koristi cifre 0-9.
2. Kolone su stepeni broja deset: jedinice, desetice, stotine, itd.
3. Ako je treća kolona 100, najveći broj koji možete napraviti sa dve kolone je 99, ili, uopšteno, sa n kolona možete predstaviti 0 do (10"-1). Stoga, sa tri kolone možete predstaviti 0 do (103-1), ili 0-999.

Druge baze

To što koristimo bazu 10 nije slučajnost; razlog je što imamo 10 prstiju. Neko, međutim, može zamisliti drugačiju bazu. Koristeći pravila, koja smo pronašli u bazi 10, možemo opisati bazu 8:

1. Cifre koje se koriste u bazi 8 su 0 do 7.
2. Kolone su stepeni broja 8: jedinice, osmice, šezdesetčetvorke, itd.
3. Sa n kolona možete predstaviti 0 do osamn-1.

Da bismo razlikovali brojeve koje smo pisali u različitim bazama, dodajmo bazu, kao subscript, odmah iza broja. Broj petnaest u bazi 10 bi tada trebalo napisati kao 15_{10} i čitati kao "jedan, pet u bazi deset."

Prema tome, da biste predstavili broj 15_{10} u bazi 8, potrebno je da napišete 17_8. Ovo se čita "jedan, sedam u bazi 8". Primetite da se ovo može pročitati i kao "petnaest", jer je ovo i dalje broj koji on nastavlja da predstavlja.

Zašto 17? Jedan znači jednu osmicu, a sedam znači sedam jedinica. Jedna osmica u sedam jedinica daju petnaest. Razmotrite petnaest zvezdica:

```
*****     *****
*****
```

Prirodna namera je da se naprave dve grupe: jedna od 10 i druga od pet zvezdica. To bi u decimalnom predstavljanju izgledalo kao 15 (jedna desetica i pet jedinica). Zvezdice, takođe, možete grupisati na sledeći nadn:

```
****          *******
****
```

To je pet zvezdica i sedam. Ovo bi moglo biti predstavljeno u bazi 8 kao 17_8, tj. jedna osmica i sedam jedinica.

Iz baze u bazu

Broj petnaest možete predstaviti u bazi deset kao 15, u bazi devet kao 16_9, u bazi osam kao 17_8, u bazi sedam kao 21_7. Zašto 21_7? U bazi sedam ne postoji cifra 8. Da biste prikazali petnaest, potrebne su Vam dve sedmice i jedna jedinica.

Kako da generalizujemo proces? Da biste konvertovali broj iz baze 10 u bazu 7, razmislite o kolonama: u bazi sedam one su jedinice, sedmice, četrdesetdevetke, tristotinečetrdesettrojke i tako dalje. Čemu ove kolone? One predstavljaju 7^0, 7^1, 7^2, 7^4 1 tako dalje. Kreirajte tabelu:

4	3	2	1
343	49	7	1

Prvi red predstavlja broj kolone, drugi stepene broja sedam, a treći decimalnu vrednost svakog broja u torn redu.

Da biste izvršili konverziju decimalne vrednosti u bazu sedam, sledite sledeću proceduru: ispitajte broj i odludte koju kolonu ćete upotrebiti prvu. Ako je broj, na primer 200, uočićete da je kolona 4 (343) jednaka nuli i nećete morati da brinete o njoj.

Da biste saznali koliko četrdesetdevetki postoji, podelite dvesta sa 49. Odgovor je četiri. Stoga, stavite četiri u kolonu tri i ispitajte ostatak: 4. U četvorci nema sedmica - stavite nulu u kolonu sedmica. Postoji četiri jedinice u broju četiri pa, zbog toga, smestite 4 u prvu kolonu. Odgovor je 404_7.

Da biste konvertovali broj 968 u bazu 6:

5	4	3	2	1
6^4	6^3	6^2	6^1	6^0
1296	216	36	6	1

Pošto nema 1296-ica u 968, kolona 5 će biti nula. Deljenjem 968 sa 216 dobićete 4, sa ostatkom 104. Kolona 4 je 4. Podelite 104 sa 36 i dobićete 2 sa ostatkom 32. Kolona 3 je dva. Podelite 32 sa 6 i dobićete 5 sa ostatkom 2. Iz ovoga sledi da je odgovor 4252_6.

5	4	3	2	1
64	63	62	61	60
1296	216	36	6	1
0	4	2	5	2

Postoji predca kada vršite konverziju iz jedne baze u drugu, na primer, iz baze 6 u bazu 10. Možete pomnožiti

```
4 * 216     =     864
2 * 36      =     72
5*6         =     30
2*1         =     2
968
```

Binarni sistem

Baza dva je proširenje ove ideje. U njoj postoje samo dve cifre: 0 i 1. Kolone su:

kolona:	8	7	6	5	4	3	2	1
stepen:	2^7	2^6	2^5	2^4	23	2^2	2^1	20
vednost:	128	64	32	16	8	4	2	1

Da biste konvertovali broj 88 u bazu 2, sledite sledeću proceduru: ne postoje 128-ice, pa je kolona 8 jednaka 0.

Postoji jedna 64-ka u broju 88, pa je kolona 7 jednaka 1, a 24 je ostatak. Ne postoje 32-ke u 24, pa je kolona 6 jednaka 0.

Postoji jedna 16-ica u 24, pa je kolona 5 jednaka 1, a ostatak je 8.

Postoji jedna 8-ica u broju 8, tako da je kolona 4 jednaka 1. Pošto nema ostatka, sve ostale kolone su 0.

0 1 0 1 1 0 0 0

Da biste proverili ovaj odgovor, izvršite konverziju unazad.

```
1*64=  64
0*32=   0
1*16=  16
1* 8=   8
0* 4=   0
0* 2=   0
0* 1=   0
        88
```

Zašto baza 2?

Snaga baze 2 je u tome što perfektno odgovara onome što kompjuteri žele da predstave. Kompjuteri ne znaju ništa o slovima, brojevima, instrukcijama, ili programima. U njihovom "srcu" se nalaze samo električna kola.

Da bi logika ostala čista, inženjeri nisu pokušali da se služe relativnom skalom (malo struje, malo vise struje, još vise struje, mnogo struje, veoma mnogo struje), nego su radije iskoristili binarnu skalu (dovoljno struje, ili nema dovoljno struje). Umesto da kažu dovoljno, ili nije dovoljno, uprostili su to sa da, ili ne. Da, ili ne se može predstaviti kao 1, ili 0. Po konvenciji, 1 znad da, ali to je samo konvencija koju, jednostavno, možemo promeniti, tako da 1 može biti i ne.

Kada ovo intuitivno shvatite, biće Vam jasnija i snaga binarnih brojeva. Sa jedinicama i nulama možete predstaviti stanje bilo kojeg kola (da li postoji struja, ili ne). Sve što kompjuteri znaju je da li si ti ti ili nisi. Ako jesi - jednako jedan, ako nisi - jednako nula.

Bitovi, bajtovi i niblovi

Kada je doneta odluka da se istina i laž prikazuju sa jedinicama i nulama, binarne cifre i bitovi su postali vrlo važni. S obzirom da su rani kompjuteri mogli da pošalju osam bitova u jednom trenutku, bilo je logično pisati programe koršćenjem osmobitnih brojeva, koji se zovu *bajtovi*.

^ **МАРОМШ** ^ Polovina bajta (četiri bita) se naziva nibl.

Sa osam binarnih cifara možete predstaviti najviše 256 različitih vrednosti. Zašto? Ispitajte kolone. Ako su svih osam bitova postavljeni (1), vrednost je 255. Ako su svi poništeni (svi bitovi su postavljeni na 0), vrednost je nula. Izmedu 0 i 255 postoji 256 mogućih stanja.

v

Sta je kilobajt?

Nadalje, 2^{10} (1024) je približno jednako 10^3 (1.000). Ova koincidencija je bila previse dobra da se ne bi iskoristila, tako da su naučnici vrednost 2^{10} bajtova nazvali 1KB, ili jedan kilobajt, a na osnovu prefiksa kilo za vrednost 1.000.

Na sličan način 1.024 x 1.024 (1.048.576) je dovoljno blisko milionu, da bi zaslužilo naziv 1MB (ili jedan megabajt), a 1.024 MB-ta je nazvano 1GB (giga potiče od hiljadu miliona, odnosno milijarde).

Binarni brojevi

Kompjuteri koriste seme jedinica i nula da bi kodirali sve što rade. Mašinske instrukcije su kodirane kao serija jedinica i nula i interpretiraju se osnovnim kolima. Odgovarajući setovi jedinica i nula se mogu pretvoriti u brojeve, ali bi bila greška misliti da ti brojevi imaju neko interesantno značenje. Na primer, Intelov 80x6 čip set interpretira šemu bitova 1001 0101 kao instrukciju. Naravno da ovo možete pretvoriti u decimalan broj 149, ali taj broj, sam po sebi, nema nikakvo značenje.

Nekada su ti brojevi instrukcije, nekada vrednosti, a nekada kodovi. Jedan vrlo važan standardizovan kod set je ASCII. U ASCII-ju je svakom slovu, interpunkciji i broju data 7-cifrena binarna reprezentacija. Na primer, malo slovo "a" je predstavljeno kao 0110 0001. Ovo nije broj, iako ga možete pretvoriti u broj 97 (64+32 + 1). Imajud prethodno rečeno u vidu, kaže se da je slovo "a" predstavljeno brojem 97 u ASCII standardu, ali prava istina je, da je binarna reprezentacija broja 97, 0110 0001 kod za slovo "a", a da je decimalna vrednost 97 tu da bi ljudima olakšala rad.

Heksadedmalni sistem

Pošto su binarni brojevi teški za dtanje, tražen je nadn da se ovo pojednostavi. Translacija iz binarnog sistema u bazu 10 uključuje komplikovanu bit manipulaciju brojevima. Medutim, pretvaranje brojeva iz baze 2 u bazu 16 je veoma jednostavno, s obzirom da postoji savršena predca.

Da biste ovo razumeli, prvo moramo da Vam objasnimo bazu 16, koja se naziva heksadecimalna baza. U bazi 16 postoji 16 cifara: 0,1,2,3,4,5,6,7,8,9, A, B, C, D, E i E Slova A-F su izabrana s obzirom da se Iako mogu dobiti sa tastature. Kolone u heksadecimalnom sistemu su:

4 3 2 1
16^3 16^2 16^1 16°
4096 256 16 1

da biste izvršili translaciju iz heksadecimalnog u decimalni sistem, upotrebićete množenje. Stoga, broj F8C predstavlja:

F * 256 = 15 * 256 = 3840
8 * 16 = 128
C * 1 = 12 * 1 = 12
3980

Translacija broja FC u binarni sistem će se obaviti tako što ćete prvo izvršiti translaciju u bazu 10, a, zatim, u binarni.

F * 16 = 15 * 16 = 240
C * 1 = 12 * 1 = 12
252

Konverzija 252_{10} u binarni zahteva tabelu

Col: 9 8 7 6 5 4 3 2 1
Power: 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2°
Value: 256 128 64 32 16 8 4 2 1

There are no 256s.
1 128 leaves 124
1 64 leaves 60
1 32 leaves 28
1 16 leaves 12
1 8 leaves 4
1 4 leaves 0
0
0
1 1 1 1 1 1 0 0

Stoga, odgovor u binarnom sistemu je 1111 1100.

Sada, ako ovaj binarni broj tretirate kao dva seta od četiri cifre, možete da izvršite magičnu transformaciju.

Desni set je 1100 Decimalno, to je 12, ili heksadecimalno C.

Levi set je 1111, koji je decimalno broj 15, a heksadecimalno F.

Stoga, imate

1111 1100
F C

Sastavljanjem ova dva heksadecimalna broja, dobijate FC, §to je realna vrednost binarnog broja 11111100. Ova prečica uvek funkcioniše. Možete uzeti bilo koji binarni broj, bilo koje dužine, i podeliti ga na setove od po četiri cifre, zatim izvršiti translaciju svakog seta u heksadecimalni broj i na kraju dobijene heksadecimalne brojeve složiti jedne pored drugih, kako bi se dobio rezultat u heksu. Evo jednog mnogo većeg broja:

1011 0001 1101 0111

Kolone su 1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384 i 32768.

1 x 1 = 1
1 x 2 = 2
1 x 4 = 4
0 x 8 = 0

1 x 16 = 16
0 x 32 = 0
1 x 64 = 64
1 x 128 = 128

1 x 256 = 256
0 x 512 = 0
0 x 1024 = 0
0 x 2048 = 0

1 x 4096 = 4,096
1 x 8192 = 8,192
0 x 16384 = 0
1 x 32768 = 32,768
Total: 45,527

Konvertovanje ovoga u heksadecimalni sistem zahteva tabelu sa heksadecimalnim vrednostima

65535 4096 256 16 1

Pošto ne postoji 65.536-ica u 45.527-ica, prva kolona je 4096. Postoji 11 4096-ica u 45.056 sa ostatkom 471. Postoji jedna 256-ica u 471 sa ostatkom 251. Postoji 13 16-ica u 208 sa ostatkom 7. Stoga je heksadecimalni broj B1D7.

Proverimo matematiku

B (11) * 4096 = 45,056
1 * 256 = 256
D (13) * 16 = 208
7 * 1 = 7
Total 45,527

Skraćena verzija uzima originalni binarni broj 1011000111010111 i deli ga u grupe po **4** cifre 1011 0001 1101 0111. Svaka od četvorki se, zatim, pretvara u heksadecimalni broj.

```
1011 =
1 x 1 =     1
1 x 2 =     2
0 x 4 =     0
1 x 8 =     8
Total       11
Hex:        B

0001 =
1 x 1 =     1
0 x 2 =     0
0 x 4 =     0
0 * 8 =     0
Total       1
Hex:        1

1101 =
1 x 1 =     1
0 x 2 =     0
1 x 4 =     4
1 x 8 =     8
Total       13
Hex =       D

0111 =
1 x 1 =     1
1 x 2 =     2
1 x 4 =     4
0 x 8 =     0
Total       7
Hex:        7
```

I na kraju, ukupni heksadecimalni broj je B1D7.

Dodotak

Odgovori

Dan 1

Kviz

1. Koja je razlika izmedu interpretatora i prevodioca?

   Interpretatori čitaju izvorni program i vrše translaciju, prevodeći programerski kod, ili programerske instrukcije direktno u "akcije." Kompajleri, tj. prevodioci, prevode izvorni kod u izvršni program, koji se kasnije može startovati.

2. Kako prevodite izvorni kod uz pomoć prevodioca?

   Svaki kompajler je različit. Prostudirajte dokumentaciju koja je stigla sa prevodiocem.

3. Šta radi povezivač (linker)?

   Posao povezivača je da poveže prevedeni kod sa bibliotekama koje su došle uz prevodioca, kao i sa drugim izvorima. Povezivač Vam omogućava da program pravite u delovima i da ga tek na kraju povežete u jedan veliki program.

4. Koji su koraci u normalnom razvojnom ciklusu?

   Izmena izvornog koda, prevodenje, povezivanje, testiranje, pa opet iz početka.

Vežbe

1. Inicijalizuje dve celobrojne promenljive, a, zatim, prikazuje njihov zbir i njihov proizvod.

2. Pogledajte uputstvo za prevodioca.

3. Neophodno je da stavite simbol "#" pre ključne reči include u prvoj liniji.
4. Ovaj program prikazuje red "Hel l o worl d" na ekranu, za kojima sledi karakter za novu liniju (CR).

Dan 2

Kviz

1. Koja je razlika između prevodioca i pretprocesora?

   Svaki put kada se startuje prevodilac, pre njega se startuje pretprocesor. On dta Vaš izvorni kod i priključuje datoteke koje ste tražili i izvršava druge poslove. 0 pretprocesoru će detaljnije biti diskutovano u Danu 18, "Objektno-orijentisana analiza i dizajn.

2. Zašto je funcija main() posebna?

   Main() se poziva automatski svaki put kada se izvršava Vaš program.

3. Koja su dva tipa komentara i po čemu se razlikuju?

   Komentari C + + stila su dve kose crte (//) i njima se komentariše bilo koji tekst do kraja linije. Komentari C stila dolaze u paru (/**/) [i s v e] izmedu odgovarajućih parova se smatra za komentar. Morate biti pažljivi u radu sa komentarima C stila.

4. Da li komentari mogu biti ugnježdeni?

   Da, komentari C + + stila mogu biti ugnježdeni unutar komentara C stila. Takođe je moguće ugnjezditi komentare C stila unutar komentara C+ + stila, dokle god imate u vidu da komentari C+ + stila važe samo do kraja linije.

5. Da li komentari mogu biti duži od jedne linije?

   Komentari C stila mogu. Ako želite da produžite komentare C+ + stila u narednu liniju, morate staviti još jedan set kosih crta (//) na početak linije.

Vežbe

1. Napišite program koji štampa Ja volim C + + na ekran.

```
linclude <iostream.h>

int main()
{
   cout « "I love 0+\n"
   return 0;
```


2. Napišite najmanji program koji se može prevesti, povezati i izvršiti.

```
int main(){ return 0; }
```

3. ISTERIVČI BAGOVA: Unesite sleded program i prevedite ga. Zašto prevodenje ne uspeva? Kako to možete popraviti?

```
1:  #include <iostream.h>
2:  main()
3:  {
4:       cout « Is there a bug here?";
5:  }
```

   U liniji 4 nedostaje navodnik koji otvara string.

4. Popravite grešku iz vežbe 3 i ponovo prevedite program, pa ga povežite i izvršite.

```
1:  #include <iostream.h>
2:  main()
3:  {
4:       cout « "Is there a bug here?";
5:  }
```

Dan3

Kviz

1. Koja je razlika između celobrojne i realne promenljive?

   Celobrojna promenljiva se sastoji samo od brojeva; realna promenljiva ima "realni deo" i "plutajud" decimalni deo. Realni brojevi se mogu predstaviti korišćenjem mantise i eksponenta.

2. Koja je razlika između unsigned short int i long int?

   Ključna reč unsigned znad da će ova promenljiva zadržati samo pozitivne brojeve. Na vedni kompjutera kratki integer-i su dugački dva, dok su 1 ong dužine 4 bajta.

3. Koje su prednosti korišćenja simboličkih konstanti nad literalima?

   Simboličke konstante objašnjavaju same sebe. Ime konstante govori čemu ona služi. Takođe, simboličke konstante mogu biti predefinisane na jednoj lokaciji unutar izvornog koda, umesto da programer ispravlja kod na mestima gde je korišćen literal.

4. Koje su prednosti korišćenja službene red const nad Idefine?

   Const promenljive imaju svoj tip. Stoga, kompajler može da pronađe greške u njihovom korišćenju. One, takođe, "preživljavaju" pretprocesor, tako da će njihovo ime biti na raspolaganju unutar debagera.

5. Šta čini naziv promenljive dobrim, ili lošim?

   Dobar naziv promenljve nam govori čemu ona služi, a loš nema nikakvu informaciju; myAge i PeopleOnTheBus su dobri nazivi promenljivih, dok xjk i prndl, verovatno ne znače ništa.

6. Posmatrajući ovu nabrojivu konstantu (enum), koju vrednost ima BLUE?

```
enum COLOR {WHITE, BLACK=100, RED, BLUE, GREEN=300};
BLUE = 102
```

7. Koje od ovih promenljivih imaju dobre nazive, koje lose, a koje su nedozvoljene?

   a. Age

   Dobar

   b. !ex

   Nelegalan

   c. R79J

   Legalan, ali je loš izbor

   d. Total Income

   Dobar

   e.___Invalid

   Legalan, ali je loš izbor

Vežbe

1. Koji bi tip promenljive bio pogodan za čuvanje slededh informacija?

   a. Starosti.

      `Unsigned short integer`

   b. Površine dvorišta.

      `Unsigned long integer`, ili `unsigned float`.

   c. Broja zvezda u galaksiji.

      `Unsigned double`.

   d. Prosečne količine padavina u januaru.

      `Unsigned short integer`.

2. Nadite dobre nazive promenljivim za sledeće informacije.

   a. MyAge

   b. BackyardArea

   c. StarsInGalaxy

   d. AverageRainFal1

   Deklarišite konstantu pi kao 3.14159.

```
const float PI = 3.14159;
```

   Deklarišite promenljivu tipa float i inicijalizujte je konstantom pi.

```
float mypi = PI;
```


Dan 4

Kviz

1. Šta je izraz?

   Bilo koja naredba koja vraća vrednost.

   Da li je `x=5+7` izraz? Koja mu je vrednost?

   Da. `12`.

   Koja je vrednost od `201/4`?

   `50`.

   Koja je vrednost od `201%4`?

   1

   Ako su `myAge`, `a` i `b` celobrojne int promenljive, koje su im vrednosti posle:

```
myAge=39;
a = myAge++;
b=++myAge;
```

   myAge: 41, a: 39, b: 41.

   Koliko je `8+2*3`?

   14.

   Koja je razlika izmedu `if(x=3)` i `if(x= =3)`?

   Prvi izraz dodeljuje `x` vrednost `3` i vraća `TRUE`. Drugi izraz proverava da li je `x=3` - vraća `TRUE`, ako jeste, a `FALSE`, ako nije.

   Da li će sledeće vrednosti biti tačne `(TRUE)`, ili netačne `(FALSE)`?

   a. 0

FALSE

b. 1

TRUE

C. -1

TRUE

d. x=0

FALSE

e. x==0 // Pretpostavite da x ima vrednost 0

TRUE

Vežbe

1. Napišite jednu if naredbu koja ispituje dve celobrojne promenljive i menja vrednost veće vrednošću manje promenljive, koristeći samo jedno else.

```
if (x > y)
    x = y;
else          // y > x i ' y == x
```

2. Proučite sledeći program. Zamislite da unosite tri broja i napišite koji izlaz očekujete.

```
linclude <iostream.h>
int main()
{
     int a, b, c;
     cout « "Please enter three numbers\n";
     cout « "a: ";
     cin » a;
     cout « "\nb: ";
     cin » b;
     cout « "\nc: ";
     cin » c;

     if (c = (a-b))
     {
          cout « "a: " « a « " minus b: "
          cout « b « " _equals c: " « c;


          cout « "a-b does not equal c:
   return 0;
```


3. Unesite u računar program iz vežbe 2; prevedite ga, povežite i izvršite. Unesite vrednosti 20, 10 i 50. Da li ste dobili izlaz koji ste očekivali? Zašto niste?

   Unesite 20, 10 i 50.

   Dobićete a: 20, b: 30, c: 10.

   Linija 13 dodeljuje vrednost, a ne vrši testiranje jednakosti.

4. Proučite ovaj program i predvidite njegov izlaz:

```
1:    #include <iostream.h>
2:    int main()
3:    {
4:         int a = 2, b = 2, c;
5:         if (c = (a-b))
6:              cout « "The value of c is: " « c;
7:    return 0;
8:    }
```

5. Unesite, prevedite, povežite i izvršite program iz vežbe 4. Sta je njegov izlaz? Zašto?

   S obzirom da linija 5 dodeljuje promenljivoj c vrednost a-b, vrednost dodeljivanja je a(1) minus b(1), što će red 0. S obzirom da nula proizvodi FALSE u if naredbi, ništa neće biti prikazano.

Dan 5

Kviz

1. Koja je razlika između prototipa funkcije i definicije funkcije?

   Prototip funkcije deklariše funkciju, a definicija je defmiše. Prototip se završava sa ;, a definicija ne. Deklaracija može da ukljud ključnu reč `inline` i podrazumevane vrednosti za parametre. Definicija to ne može. Deklaracija nema potrebu da ukljud imena za parametre, dok definicija to mora.

2. Da li imena parametara treba da se slažu u prototipu, definiciji i pozivu funkcije?

   Ne, svi parametri su identifikovani pozicijom, a ne imenom.

3. Ako funkcija ne vraća vrednost, kako ćete deklarisati funkciju?

   Deklarišite funkciju da vraća void.

4. Ako ne deklarišete šta će funkcija vratiti, koji tip će biti podrazumevan?

   Bilo koja funkcija koja eksplicitno ne deklariše vraćeni tip, vratiće `int`.

5. Sta je to lokalna promenljiva?

Lokalna promenljiva je promenljiva prosledena ili deklarisana unutar bloka, uobičajeno funkcije. Ona je vidljiva samo u okviru bloka.

6. Šta je to opseg?

Opseg odreduje vidljivost i život lokalne i globalne promenljive. Opseg je obično ograničen setom vitičastih zagrada.

7. Šta je to rekurzija?

Rekurzija obično objašnjava mogućnost funkcije da pozove samu sebe.

8. Gde bi trebalo da se koriste globalne promenljive?

Globalne promenljive se obično koriste kada vise funkcija ima potrebu da pristupi istim podacima. Globalne promenljive su vrlo retke u C++. Jednom kada naučite da kreirate statičke promenljive klase, praktično nikada vise nećete imati potrebu za globalnim promenljivima.

9. Šta je overload funkcije?

Overload funkcije je mogućnost da napišete vise od jedne funkcije sa istim imenom, ali sa različitim brojem ili tipovima parametara.

10. Šta je to polimorfizam?

Polimorfizam je mogućnost da tretirate vise objekata različitih, ali srodnih tipova, na isti način, bez obzira na njihove razlike. U C++ polimorfizam se izvršava korišćenjem izvodenja klasa i virtuelnih funkcija.

Vežbe

1. Napišite prototip za funkciju pod imenom Perimeter koja vraća unsigned long int i koja uzima dva parametra i to oba unsigned short int.

```
unsigned long int Perimeter(unsigned short int, unsigned short int);
```

2. Napišite definiciju za funkciju Perimeter iz vežbe 1. Dva parametra predstavljaju dužinu i širinu pravougaonika, imaju funkciju da vrate obim (dva puta dužina + dva puta širina).

```
unsigned long int Perimeter(unsigned short int length, unsigned short int width)
{
  return 2*length + 2*width;
}
```

3. ISTERIVAČI BAGOVA: Šta je pogrešno u sledećoj funkciji:

```
linclude <iostream.h>
void myFunc(unsigned short int x);
int  main()
{
    unsigned short int x, y;
    y = myFunc(int);
    cout « "x:  " « x « " y:  • « y « "\n";
return 0;
)

void myFunc(unsigned short int x)
{
    return  (4*x);
}
```

Funkcija je deklarisana da vraća void i ne može da vrati vrednost.

Šta je neispravno u sledećoj funkciji:

```
linclude <iostream.h>
int myFunc(unsigned short int x);
int  mainQ
{
    unsigned short int x, y;
    y = myFunc(int);
    cout « "x:  " « x « " y:  " « y « "\n"*
return 0;
}

int myFunc(unsigned short int x)
{
    return  (4*x);
)
```

Funkcija je u redu, ali postoji; na kraju zaglavlja definicije funkcije.

Napišite funkciju koja uzima dva unsigned `short i` nt argumenta i vraća rezultat njihovog deljenja. Nemojte izvršiti deljenje ako je drugi broj nula, nego vratite **-1.**

```
short int Divider(unsigned short int valOne, unsigned short int valTwo)

    if (valTwo == 0)
          return -1;
    el se
          return valOne / valTwo;
}
```

Napišite program koji traži od korisnika da unese dva broja i poziva funkciju koju ste napisali u Vežbi 5. Prikažite odgovor ili poruku o grešci ako ste dobili -1.

```
linclude <iostream.h>
typedef unsigned short int USHORT;
typedef unsigned long int ULONG;
short  int  Divider(
unsigned short int valone,
unsigned short  int  valtwo);
```

```
int mainQ
{
    USHORT one. two;
    short int answer;
    cout « "Enter two numbers.\n Number one:
    cin » one;
    cout « "Number two: ";
    cin » two;
    answer = Divider(one, two);
    if (answer > -1)
       cout « "Answer: " « answer;
    else
       cout « "Error, can't divide by zero!";
return 0;
}
```

ttt!"

Napišite program koji pita za broj i za stepen. Napišite rekurzivnu funkciju koja izračunava stepen broja. Stoga, ako je broj 2, a stepen 4, funkcija bi trebalo da vrati 16.

```
linclude <iostream.h>
typedef unsigned short USHORT;
typedef unsigned long ULONG;
ULONG GetPower(USHORT n, USHORT power);
int main()
{
   USHORT number, power;
   ULONG answer;
   cout « "Enter a number: ";
   cin » number;
   cout « "To what power? ";
   cin » power;
   answer = GetPower(number,power);
   cout « number « " to the " « power « "th power is " «
answer « endl;
return 0;
}

ULONG GetPower(USHORT n, USHORT power)

    if(power == 1)
     return n;
    el se
       return (n * GetPower(n.power-l));
```


Dan 6

Kviz

1. Šta je to tačka operator i za šta se on koristi?

   Tačka operator je tačka (.) i koristi se za pristup clanovima klase.

2. Šta dovodi do odvajanja memorije: deklaracija ili definicija?

   Definicija promenljivih zahteva memoriju. Deklaracija klasa ne odvaja memoriju.

3. Da li je deklaracija klase njen interfejs ili njena implementacija?

   Deklaracija klase je njen interfejs. On govori klijentu klase kako da komunicira sa klasom. Implementacija klase je set funkcija cianova koje su smeštene u odgovarajuće CPP datoteke.

4. Koja je razlika između public i private podataka cianova?

   Public podacima clanovima mogu pristupiti klijenti klase. Private podacima clanovima se može pristupiti samo sa funkcijama clanovima te klase.

5. Mogu li funkcije članovi biti private?

   Da. I funkcije članovi i podaci članovi mogu biti private.

6. Mogu li podaci članovi biti `public`?

   Iako podaci članovi mogu biti public, dobra programerska praksa je da oni budu private, ali da se obezbede `public` pristupne funkcije tim podacima.

7. Ako deklarišete dva `Cat` objekta, mogu li oni imati različite vrednosti za `itsAge` podatak član?

   Da, svaki objekat klase ima svoje sopstvene podatke clanove.

8. Da li se deklaracije klase završavaju sa ; ? A da li definicije metoda klase?

   Deklaracije se završavaju sa ; iza zatvorene vitičaste zagrade, dok definicije funkcija ne.

9. Sta bi bilo zaglavlje za `Cat` funkciju `Meow` koja ne uzima parametre i vraća `void`?

   Zaglavlje za `Cat` funkciju `Meow()` koja ne uzima parametre i vraća `void` bi bilo

   ```
   void Cat::Meow()
   ```

10. Koja funkcija se poziva za inicijalizaciju klase?

    Konstruktor se poziva za inicijalizaciju klase.

Vežbe

1. Napišite kod koji deklariše klasu pod nazivom **Employee** sa sledećim podacima clanovima: **Age, Years** of **Service** i **Salary.**

```
yearsOfService, and Salary.
class Employee
{
    int Age;
    int YearsOfService;
    int Salary;
};
```

2. Ispravite Empl oyee klasu tako da podaci članovi budu private i obezbedite public pristupne metode za preuzimanje i postavljanje svakog od podataka cianova.

```
class Employee
{
public:
    int GetAgeO const;
    void SetAge(int age);
    int GetYearsOfServiceOconst;
    void SetYearsOfService(int years);
    int GetSalary()const;
    void SetSalary(int salary);

private:
    int Age;
    int YearsOfService;
    int Salary;
};
```

3. Napišite program sa Employee klasom koji pravi dva Employees. Postavite njihove Age, Years of Service i Salary i na kraju otštampajte te vrednosti.

```
main()
{
    Employee John;
    Employee Sally;
    John.SetAge(30);
    John.SetYears0fService(5);
    John.SetSalary(50000);

    Sally.SetAge(32);
    Sally.SetYears0fService(8);
    Sally.SetSalary(40000);

    cout « "At AcmeSexist company, John and Sally have the same
job.\n";
    cout « "John is " « John.GetAgeO « " years old and he has
been with";
    cout « "the firm for " « John.GetYearsOfService « "
years.\n";
    cout « "John earns $" « John.GetSalary « " dollars per
year.\n\n";
    cout « "Sally, on the other hand is " « Sally.GetAgeO « "
years old and has";
    cout « "been with the company " « Sally.GetYearsOfService;
    cout « " years. Yet Sally only makes $" « Sally.GetSalaryQ;
    cout « " dollars per year!  Something here is unfair.";
```


4. Nastavite sa vežbom 3, tako što ćete obezbediti metod za **Employee** koji izveštava, koliko hiljada dolara zaposleni zarađuju, zaokruženo na najbliže 1000.

```
f 1 oat Empl oyee -.GetRoundedThousands () const
{
    return Salary / 1000;
}
```

5. Izmenite Empl oyee klasu, tako da možete da inicijalizujete Age, Years of Servi ce i Salary kada kreirate zaposlenog.

```
class Employee
{
public:

    Employee(int age, int yearsOfService, int salary);
    int GetAgeQconst;
    void SetAge(int age);
    int GetYearsOfService()const;
    void SetYearsOfService(int years);
    int GetSalaryQconst;
    void SetSalary(int salary);

private:
    int Age;
    int YearsOfService;
    int Salary;

};
```

6. ISTERIVAČI BAGOVA: Šta je neispravno u sledećoj deklaraciji:

```
class Square
{
public:
    int Side;
}
```

Deklaracija klase mora da se završava sa ;

7. ISTERTVAČI BAGOVA: Zašto sledeća deklaracija klase nije baš mnogo korisna?

```
class Cat
f
```

```
    int GetAge Oconst;
private:
    int itsAge;
};
```

Pristupna funkcija GetAgeO je private. Zapamtite: Svi članovi klase su private, ako drugačije nije rečeno.

8. ISTERIVAČI BAGOVA: Koja tri baga u ovom kodu će pronaći kompajler?

```
class TV
{
public:
    void SetStation(int Station);
    int GetStation() const;
private:
    int itsStation;
};

main()
{
    TV myTV;
    myTV.itsStation = 9;
    TV.SetStation(IO);
    TV my0therTv(2);
}
```

Ne možete direktno pristupiti itsStation, s obzirom da je private.

Ne možete pozvati SetStationQ u klasi. SetStationQ možete pozvati samo u objektu.

Ne možete izvršiti inicijalizaciju itsStation, s obzirom da ne postoji odgovarajući konstruktor.

Dan 7

Kviz

1. Kako da inicijalizujem vise od 1 promenljive u for petlji? Odvojite inicijalizacije zapetama, kao u sledećem primeru.

```
for (x = 0, y = 10; x < 100; x++, y++)
```

2. Zašto treba izbegavati goto?

   Goto skače u bilo kom pravcu, na bilo koju liniju koda. Zbog ovoga izvorni kod postaje težak za razumevanje, a samim tim i za održavanje.

3. Da li je moguće napisati for petlju sa telom koje se nikada neće izvršiti?

   Da, ako je uslov FALSE po inicijalizaciji, telo for petlje se nikada neće izvršiti.

I Evo primera:

f
```
for (int X = 100; X < 100; X++)
```

4. Da li je moguće ugnjezditi whi 1 e petlju unutar for petlje?

   Da, bilo koja petlja može biti ugnježdena unutar bilo koje druge petlje.

5. Da li je moguće kreirati petlju koja se nikada neće završiti? Dajte jedan primer.

* Da. Evo primera i za for i za whi 1 e petlju.

```
{
    // Ova for petlja se nikada ne završava!
}
while(I)
{
    // Ova while petlja se nikada ne završava!
}
```

6. Sta će se dogoditi ako kreirate petlju koja se nikada ne završava?

   Vaš program će se "obesiti" i verovatno ćete morati da restartujete kompjuter.

Vezbe

1. Koju vrednost će imati x kada se završi for petlja?

```
for (int x = 0; x < 100; x++)
```

   100.

2. Napišite ugnježdenu for petlju koja prikazuje šemu od 10x10 nula.

```
for (int i = 0 ; i< 10; i++)
{
    for ( int j = 0; j< 10; j++)
       cout « "0";
       cout « "\n";
}
```

3. Napišite for naredbu koja broji od 100 do 200 sa inkrementom 2.

```
for (int x = 100; x<=200; x+=2)
```

4. Napišite whi 1 e petlju koja broji od 100 do 200 sa inkrementom 2.

```
int x = 100;
while (x <= 200)
    x+= 2;
```

5. Napišite do-whi 1 e petlju koja broja od 100 do 200 sa inkrementom 2.

```
int x = 100;
do
{
    x+=2;
} while (x <= 200);
```

6. ISTERIVAČI BAGOVA: Šta nije u redu sa sledećim kodom:

```
int counter = 0
while (counter < 10)

    cout « "counter:   «   counter;
    counter++;
```

Brojač nije nikad inkrementiran i while petlja se nikad neće završiti.

7. ISTERIVAČI BAGOVA: Šta je pogrešno u sledećem kodu:

```
for (int counter = 0; counter < 10; counter++);
    cout « counter « "\n";
```

Postoji ; posle petlje i petlja ne radi ništa. Programer je verovatno nameravao da prikaže svaku vrednost, ali to nije uradio.

8. ISTERIVAČI BAGOVA: Šta je pogrešno u sledećem kodu:

```
int counter = 100;
while (counter < 10)
{
    cout « "counter now:  " « counter;
    counter-;;
}
```

Counter je inicijalizovan na 100, ali test proverava da li je on manji od 10, tako da se telo petlje nikada neće izvršiti. Ako bi prva linija bila zamenjena u int Counter = 5; , petlja se ne bi završila sve dok Counter ne bi dosegao najmanju moguću int vrednost. S obzirom da je int označen, ovo ne bi bilo ono što ste želeli.

9. ISTERIVAČI BAGOVA: Šta je pogrešno u sledećem kodu:

```
cout « "Enter a number between 0 and 5:  ";
cin  »  theNumber;
switch  (theNumber)
{
   case 0:
         doZero();
   case 1:              // propada
   case 2:              // propada
   case 3:              // propada
   case 4:              // propada
   case 5:
         do0neToFive();
         break;
   default:
         doDefault();
         break;
}
```

Case 0 verovatno zahteva Break naredbu. Ako ne zahteva, trebalo bi biti dokumentovano komentarom.

Dan 8

Kviz

1. Koji operator se koristi za odredivanje adrese promenljive?

   Adresa operatora (&) se koristi za odredivanje adrese bilo koje promenljive.

2. Koji operator se koristi za pronalaženje vrednosti, smeštene na adresi, koju sadrži pointer?

   Operator za dereferenciranje (*) se koristi za pristup vrednosti na adresi iz pointera.

3. Šta je pointer?

   Pointer je promenljiva koja sadrži adresu druge promenljive.

4. Koja je razlika između adrese smeštene u pointer i vrednosti te adrese?

   Adresa smeštena u pointer je adresa druge promenljive. Vrednost smeštena na toj adresi je bilo koja vrednost smeštena u bilo kojoj promenljivoj. Operator (*) vraća vrednost smeštenu na adresi koja je sama smeštena u pointeru.

5. Koja je razlika između *operatora i adrese operatora?

   ^operator vraća vrednost sa adrese koja je smeštena u pointeru. Adresa operatora (&) vraća memorijsku adresu promenljive.

6. Koja je razlika između const int * ptrOne i int * const ptrTwo?

   Prvim je deklarisano da je ptrOne pointer na constant integer. Sam integer se ne može promeniti korišćenjem ovog pointera.

   Drugim je deklarisano da je ptrTwo constant pointer na integer i jednom kada je inicijalizovan ovaj pointer ne može biti ponovo dodeljen.

Vežbe

1. Šta sledeće deklaracije rade:

a. `int * pOne;`

b. `int vTwo;`

c. `int * pThree = &vTwo;`

a. deklarišu pointer na integer.

b. deklarišu celobrojnu promenljivu.

c. deklarišu pointer na integer i inicijalizuje ga sa adresom druge promenljive.

2. Ako imate promenljivu unsigned short pod imenom YourAge, kako biste deklarisali pointer za manipulaciju sa YourAge?

```
unsigned short *pAge = &yourAge;
```

3. Dodelite vrednost 50 promenljivoj YourAge uz pomoć pointera koji ste deklarisali u vežbi 2.

```
*pAge = 50;
```

4. Napišite mali program koji deklariše integer i pointer na integer. Dodelite adresu integer-a pointeru. Iskoristite pointer za postavljanje vrednosti integer promenljive.

```
int theInteger;
int *pInteger = &theInteger;
*pInteger = 5;
```

5. ISTERIVAČI BAGOVA: Šta je pogrešno u sledećem kodu?

```
linclude <iostream.h>
int mainQ
{
     int *pInt;
     *pInt = 9;
     cout « "The value at pint: " « *pInt;
return 0;
}
```

pint bi morao da bude inicijalizovan. Sto je još važnije, s obzirom da on nije inicijalizovan i da mu nije dodeljena adresa u memoriji, on ukazuje na slučajno mesto u memoriji. Dodeljivanje broja 9 slučajnom mestu u memoriji je vrlo opasan bag.

6. ISTERIVAČI BAGOVA: Šta je neispravno u sledećem kodu:

```
int mainQ
{
    int SomeVariable = 5;
    cout « "SomeVariable:  " « SomeVariable « "\n";
    int *pVar = & SomeVariable;
    pVar = 9;
    cout « "SomeVariable:  " « *pVar « "\n";
return 0;
}
```

Pretpostavljamo da je programer želeo da dodeli 9 vrednosti na koju ukazuje pVar. Na nesreću, 9 je dodeljena vrednosti pVar, postoje zaboravljen operator*. Ovo će dovesti do katastrofe kada se pVar iskoristi za dodeljivanje vrednosti.

Dan 9

Kviz

1. Koja je razlika između reference i pointera?

   Referenca je alias, a pointer je promenljiva koja sadrži adresu.

2. Kada morate da koristite pointere umesto referenci?

   Kada imate potrebu da ponovo dodelite vrednost na ono na šta se ukazuje ili kada pointer treba da bude null.

3. Sta će new vratiti, ako nema dovoljno memorije za kreiranje Vašeg novog objekta?

   Null pointer (0).

4. Šta je konstantna referenca?

   To je kraći nadn da kažete: referenca na konstantan objekat.

5. Koja je razlika između prosledivanja po referenci i prosledivanja reference?

   Prosledivanje po referenci znad da se ne pravi lokalne kopija. Ono može biti završeno prosleđivanjem reference ili prosledivanjem pointera.

Vežbe

1. Napišite program koji deklariše `int`, referencu na int i pointer na `int`. Iskoristite pointer i referencu za manipulaciju vrednosti `int`-a.

```
int mainQ
{
int varOne;
int& rVar = varOne;
int* pVar = &varOne;
rVar = 5;
*pVar = 7;
```

```
return 0;
}
```

2. Napišite program koji deklariše constant pointer na constant integer. Inicijalizujte pointer na integer promenljivu VarOne. Dodelite 6 promenljivoj VarOne. Koristite pointer da biste dodeli 7 promenljivoj VarOne. Kreirajte drugu celobrojnu promenljivu VarTwo. Ponovo dodelite pointer na VarTwo.

```
int main()
{
   int varOne;
   const int * const pVar = ivarOne;
   *pVar = 7;
   int varTwo;
   pVar = &varTwo;
return 0;
}
```

3. Kompajlirajte program iz Vežbe 2. Šta proizvodi greške, a šta proizvodi upozorenja?

   Ne možete dodeliti vrednost constant objektu i ne možete ponovo dodeliti constant pointer.

4. Napišite program koji proizvodi neispravan pointer.

```
int main()
{
int * pVar;
*pVar = 9;
return 0;
}
```

5. Ispravite program iz Vežbe 4.

```
int main()
{
int VarOne;
int * pVar = &var0ne;
*pVar = 9;
return 0;
}
```

6. Napišite program koji dovodi do curenja memorije.

```
int Func0ne();
int main()
{
   int localVar = FunOneQ;
   cout « "the value of localVar is: " « localVar;
return 0;
)
```



```
int FuncOneO
{
   int * pVar = new int (5);
   return *pVar;
}
```

7. Ispravite program iz Vežbe 6.

```
void FuncOneO;
int main()
{
   FuncOneO;
return 0;
}

void FuncOneO
{
   int * pVar = new int (5);
   cout « "the value of *pVar is: " « *pVar ;
}
```

8. ISTERIVAČI BAGOVA: Šta nije u redu sa sledećim programom?

```
1:    linclude <iostream.h>
2:
3:    class CAT
4:    {
5:       public:
6:          CAT(int age) { itsAge = age; }
7:          -CAT(){}
8:          int GetAgeO const { return itsAge;}
9:       private:
10:         int itsAge;
11:   };
12:
13:   CAT & MakeCat(int age);
14:    int main()
15:   {
16:      int age = 7;
17:      CAT Boots = MakeCat(age);
18:      cout « "Boots is " « Boots.GetAgeO « " years old\n";
19:     return 0;
20:   }
21:
22:   CAT & MakeCat(int age)
23:   {
24:      CAT * pCat = new CAT(age);
25:      return *pCat;
26:   }
```

MakeCat vraća referencu na Cat koji je kreiran u slobodnoj memoriji. Ne postoji nacin da se ta memorija oslobodi, a to dovodi do curenja memorije.

9. Ispravite program iz Vežbe 8.

```
1:  linclude <iostream.h>
2:
3:  class CAT
4:  {
5:     public:
6:        CAT(int age) { itsAge = age; }
7:        ~CAT(){}
8:        int GetAgeO const { return itsAge;}
9:     private:
10:       int itsAge;
11
12
13  CAT * MakeCat(int age);
14  int main()
15  {
16     int age = 7;
17     CAT * Boots = MakeCat(age);
18     cout « "Boots is " « Boots->GetAge() « " years old\n";
19     delete Boots;
20   return 0;
21
22
23  CAT   MakeCat(int age)
24  {
25     return new CAT(age);
26
```

Dan 10

Kviz

1. Kada vršite overload funkcija cianova, na koji način one moraju da se razlikuju?

   Funkcije članovi nad kojima je izvršen overload su funkcije klase koje dele isto ime, ali se razlikuju u broju ili tipu njihovih parametara.

   Koja je razlika između deklaracije i definicije?

   Definicija zahteva memoriju, dok deklaracija ne. Uglavnom su sve deklaracije i definicije; glavni izuzeci su deklaracije klasa, prototipovi funkcija i typedef naredba.

3. Kada se poziva copy konstruktor?

   Svaki put kada se kreira privremena kopija objekta. Ovo se dešava svaki put kada se objekat prosleduje po vrednosti.

4. Kada se poziva destruktor?

   Destruktor se poziva svaki put kada se objekat uništava, bilo zato što je izvan opsega ili zato što ste pozvali brisanje pointera koji ukazuje na objekat.

5. Kako se copy konstruktor razlikuje od operatora za dodeljivanje?

   Operator za dodeljivanje deluje na postojeće objekte; copy konstruktor kreira nove.

6. Šta je to this pointer?

   Thi s pointer je skriveni parametar u svakoj funkciji članu, koji ukazuje na sam objekat.

7. Kako ćete razlikovati overload prefiks i postfiks inkremenata?

   Prefiks operator ne uzima parametre. Postfiks operator uzima i nt parametar koji se koristi kao signal kompajleru da je on postfiks varijanta.

8. Možete li izvršiti overload operatora plus za short integer?

   Ne, ne možete izvršiti overload nijednog operatora za ugradene tipove.

9. Da li je u C++ legalno izvršiti overload operatora++ tako da vrši dekrementiranje vrednosti u Vašoj klasi?

   To je legalno, ali je loša ideja. Overload operatora treba raditi na taj nadn da budu razumljivi svakome ko čita Vaš kod.

10. Koja vraćena vrednost mora da izvrši konverziju operatora koji ima u svojoj deklaraciji?

    Nijedna, kao konstruktori i destruktori oni nemaju vraćenu vrednost.

Vežbe

1. Napišite deklaraciju SimpleCircle klase sa jednom promenljivom članom ItsRadius. Ukljudte podrazumevani konstruktor, destruktor i pristupne metode za ItsRadius.

```
class  SimpleCircle
{
publi c:
     SimpleCircleQ;
     -SimpleCircleO;
     void SetRadius(int);
     int  GetRadius();
private:
     int  itsRadius;
```

2. Korišćenjem klase koju ste kreirali u Vežbi 1, napišite implementaciju za podrazumevani konstruktor, inicijalizujud ItsRadius sa vrednošću 5.

```
SimpleCircle::SimpleCircle():
itsRadius(5)
{}
```

3. Korišćenjem iste klase, dodajte isti konstruktor koji uzima vrednost za svoj parametar i dodeljuju tu vrednost ItsRadius-u.

```
SimpleCircle::SimpleCircle(int  radius):
itsRadius(radius)
{}
```

4. Kreirajte prefiks i postfiks inkrement operator za Vašu SimpleCircle klasu koji inkrementira ItsRadius.

```
const SimpleCircle* SimpleCircle::operator++()
{
     ++(itsRadius);
     return  *this;
}

// Operator ++(int)  postfiks.
// Prvo vra}a a onda inkrementira
const SimpleCircle SimpleCircle::operator++ (int)
{
// deklariše lokalni SimpleCircle i i inicijalizuje na vrednost *this
    SimpleCircle  temp(*this);
    ++(itsRadius);
    return  temp;
}
```

5. Izmenite SimpleCircle tako da smešta ItsRadius u Slobodan prostor i ispravite postojeće metode.

```
{
public:
     SimpleCircleO;
     SimpleCircle(int);
     -SimpleCircleO;
     void  SetRadius(int);
     int  GetRadiusO;
     const SimpleCircle& operator++();
     const SimpleCircle operator++(int);
private:
     int  *itsRadius;
};

SimpleCircle::SimpleCircleO
{itsRadius = new int(5);}

SimpleCircle::SimpleCircle(int  radius)
{itsRadius = new int(radius);}
```



```
const SimpleCirclei SimpleCircle::operator++()
{
     ++(itsRadius);
     return  *this;
}

// Operator ++(int)  postfiks.
// Prvo vra}a a onda inkrementira
const SimpleCircle SimpleCircle::operator++ (int)

// deklariše lokalni SimpleCircle i i inicijalizuje na vrednost *this
    SimpleCircle  temp(*this);
    ++(itsRadius);
    return  temp;
}
```

6. Obezbedite copy konstruktor za SimpleCircle.

```
(
     int val = rhs.GetRadiusO;
     itsRadius = new int(val);
}
```

7. Obezbedite operator = za SimpleCircle.

```
SimpleCircle& SimpleCircle::operator=(const SimpleCircle & rhs)
{
     if (this == &rhs)
          return  *this;
     delete  itsRadius;
    itsRadius = new int;
    *itsRadius = rhs.GetRadiusO;
    return  *this;
}}
```

8. Napišite program koji kreira dva SimpleCircle objekta. Koristite podrazumevani konstruktor za 1 i inicijalizujte drugi sa vrednošću 9. Pozovite inkrement za svaki i zatim prikažite njihove vrednosti. Na kraju, dodelite drugi prvome i prikažite njihove vrednosti.

```
#include <iostream.h>

class SimpleCircle
{
public:
      //  konstruktori
     SimpleCircleO;
     SimpleCircle(int);
     SimpleCircle(const SimpleCircle &);
     -SimpleCircleO  {}

// funkcije pristupa
```

```
    void SetRadius(int);
    int GetRadiusQconst;

// operatori
    const SimpleCircle& operator++();
    const SimpleCircle operator++(int);
    SimpleCircle* operator=(const SimpleCircle &);

private:
    int *itsRadius;



SimpleCircle: :SimpleCircleQ
{itsRadius = new int(5);}

SimpleCircle: :SimpleCircle(int radius)
{itsRadius = new int(radius);}

SimpleCircle::SimpleCircle(const SimpleCircle & rhs)
{
    int val = rhs.GetRadiusO;
    itsRadius = new int(val);
}
SimpleCircle& SimpleCircle::operator=(const SimpleCircle & rhs)
{
    if (this == &rhs)
        return *this;
    *itsRadius = rhs.GetRadiusO;
    return *this;
}

const SimpleCircle& SimpleCircle::operator++()
{
    ++(itsRadius);
    return *this;
}

// Operator ++(int) postfiks.
// Prvo vra}a a onda inkrementira
const SimpleCircle SimpleCircle::operator++ (int)
{
// deklariše lokalni SimpleCircle i i inicijalizuje na vrednost *this
    SimpleCircle temp(*this);
    ++(itsRadius);
    return temp;
}
int SimpleCircle::GetRadius() const
{
    return *itsRadius;
```



```
)
int main()
{
    SimpleCircle CircleOne, CircleTwo(9);
    CircleOne++;
    ++CircleTwo;
    cout « "CircleOne: " « CircleOne.GetRadiusO « endl;
    cout « "CircleTwo: " « CircleTwo.GetRadiusO « endl;
    CircleOne = CircleTwo;
    cout « "CircleOne: " « CircleOne.GetRadiusO « endl;
    cout « "CircleTwo: " « CircleTwo.GetRadiusO << endl;
return 0;
```

9. ISTERIVAČI BAGOVA: Proverite šta je pogrešno sa ovom implementacijom operatora dodeljivanja:

```
SQUARE SQUARE ::operator=(const SQUARE & rhs)
{
    itsSide = new int;
    *itsSide = rhs.GetSide();
    return *this;
}
```

Morate proveriti, da li je rks jednako thi s, ili će poziv za a jednako a oboriti Vaš program.

10. ISTERIVAČI BAGOVA: Šta je pogrešno sa ovom implementacijom operatora plus:

```
VeryShort VeryShort::operator+ (const VeryShort* rhs)

  itsVal += rhs.GetItsVal();
  return *this;
}
```

Ovaj operator plus je promenio vrednost jednom od operanata, umesto da je kreirao novi very short objekat za zbir. Ispravan način da ovo uradite je sledeći:

```
VeryShort VeryShort::operator+ (const VeryShort& rhs)
{
  return VeryShort(itsVal + rhs.GetItsVal());
```

Dan 11

Kviz

1. Koji su prvi i poslednji element u `SomeArray [25]` ?

`SomeArray[0], SomeArray[24]`

2. Kako ćete deklarisati višedimenzionalni niz?

   Napišite set subscript-ova za svaku dimenziju. Na primer, SomeArray [2], [3] i [2] je trodimenzionalni niz. Prva dimenzija ima 2 elementa, druga ima 3 i treća ima 2.

3. Inicijalizujte clanove niza iz pitanja 2?

```
SomeArray[2][3][2] = { { {1,2},{3,4},{5,6} } , { {7,8},{9,10},{11,12} } };
```

4. Koliko elemenata sadrži niz SomeArray [10], [5] i [20].

```
10x5x20=1000
```

5. Koji je maksimalni broj elemenata koji se može dodati u povezanu listu?

   Ne postoji maksimum. Zavisi samo od količine memorije koju imate na raspolaganju.

6. Može li se koristiti subscript notacija u radu sa povezanim listama?

   U radu sa povezanim listama možete koristiti `subscript` notaciju samo ako napišete Vašu sopstvenu klasu koja sadrži povezane liste i izvršite overload subscript operatora.

7. Koji je poslednji karakter u stringu `"Brad is a nice guy?"`

   Null karakter.

Vežbe

Deklarišite 2-dimenzionalni niz koji predstavlja "iks-oks" tablu za igru.

```
int GameBoard[3][3];
```

Napišite kod koji inicijalizuje sve elemente niza koji ste kreirali u Vežbi 1 na vrednost 0.

```
int GameBoard[3][3] = { {0,0,0},{0,0,0},{0,0,0} }
```

Napišite deklaraciju za `Node` klasu koja sadrži `unsigned short integers`.

```
class Node
 {
 public:
    Node ();
    Node (int);
    -Node();
    void SetNext(Node * node) { itsNext = node; }
    Node * GetNext() const { return itsNext; }
    int GetVal() const { return itsVal; }
    void Insert(Node *);
    void DisplayO;
 private:
    int itsVal;
    Node * itsNext;
 };
```

4. ISTERIVAČI BAGOVA: Šta nije u redu sa sledećim delom koda:

```
        unsigned short SomeArray[5][4];
for (int i = 0; i<4; i++)
     for (int j = 0; j<5;
          SomeArray[i] [j] = i+j
```

   Niz je veličine 5x4 elementa, ali kod inicijalizuje 4x5.

5. ISTERIVAČI BAGOVA: Šta nije u redu sa sledećim delom koda

```
unsigned short SomeArray[5][4];
for (int i =0; i<=5; i++)
     for (int j = 0; j<=4; j++)
          SomeArray[i][j] = 0;
```


Dan 12

Kviz

1. Sta je v-table?

   V-table, ili tabela virtuelnih funkcija, je uobičajeni način na koji kompajleri upravljaju virtuelnim funkcijama u C++-u. Tabela sadrži listu adresa svih virtuelnih funkcija i, u zavisnosti od tipa objekta na koji je ukazano u fazi izvršavanja, biće pozvana odgovarajuća funkcija.

2. Šta je virtuelni destruktor?

   Destruktor bilo koje klase može biti deklarisan kao virtuelan. Kada pointer bude obrisan, pristupiće se tipu objekta u fazi izvršavanja i biće pozva* odgovarajući izvedeni destruktor.

3. Kako ćete prikazati deklaraciju virtuelnog konstruktora?

   Ne postoje virtuelni konstruktori.

4. Kako se kreira virtuelni copy konstruktor?

   Kreiranjem virtuelne metode u Vašoj klasi, koja sama poziva copy konstruktor.

5. Kako se poziva bazna funkcija iz izvedene klase, u kojoj ste izvršili override te funkcije?

```
base::FunctionName();
```

6. Kako se poziva bazna funkcija izvedene klase, u kojoj niste izvršili override te funkcije?

```
FunctionNameO;
```

7. Ako je bazna klasa deklarisala funkciju kao virtuelnu i izvedena klasa nije koristila izraz virtuelna kada je vršila override te klase, da li je ona i dalje virtuelna kada se nasledi u klasi treće generacije?

   Da. Virtuelnost se nasleduje i *ne moze se* isključiti.

8. Za šta se koristi ključna reč protected?

   protected clanovima se može pristupiti iz funkcija cianova izvedenih objekata.

Vežbe

1. Prikažite deklaraciju virtuelne funkcije, koja preuzima celobrojni parametar, a vraća void.

```
Virtuel void someFunction(int);
```

2. Prikažite deklaraciju klase square koja se izvodi iz klase rectangle, koja se izvodi iz klase Shape.

```
class Square : public Rectangle
{};
```

3. Ako u vežbi 2 Shape ne uzima parametre, Rectangl e uzima 2 (dužina i širina), ali Square uzima samo jedan (dužina), prikažite inicijalizaciju konstruktora za Square.

```
Square::Square(int lenght):
        Rectangle(lenght lenght){}
```

4. Napišite virtuelni copy konstruktor za klasu square iz vežbe 3.

```
class Square
      {
         public:
         // -
         virtual Square * clone() const { return new Square(*this); }
       // -
     };
```

5. ISTERIVAČI BAGOVA: Šta nije ispravno u sledećem kodu:

```
void SomeFunction (Shape);
Shape * pRect = new Rectangle;
SomeFunction(*pRect);
```

   Cini se da ovde grešaka nema. SomeFunction očekuje objekat Shape. Vi ste joj prosledili Rectangle. Dokle god nemate potrebu za bilo kojim od delova Rectangle, ovo će biti u redu. Ako su Vam potrebni delovi Rectangle, moraćete da promenite SomeFunction, tako da uzima pointer, ili referencu na Shape.

6. ISTERIVAČI BAGOVA: Šta nije ispravno u sledećem kodu:

```
class ShapeO
{
public:
     ShapeO;
     virtual -ShapeO;
     virtual Shape(const Shape*);
};
```

   Copy konstruktor se ne može deklarisati kao virtuelan.

Dan 13

Kviz

1. Sta je podela uloga "nadole"?

   Podela uloga "nadole" je deklaracija u kojoj se pointer na baznu klasu tretira kao pointer na izvedenu klasu.

2. Šta je v-ptr?

   v-ptr je virtuelni pointer na funkciju i to je jedan implementacioni detalj virtuelnih funkcija. Svaki objekat u klasi sa virtuelnim funkcijama ima v-ptr, koji ukazuje na tabelu virtuelnih funkcija te klase.

3. Ako zaobljeni pravougaonik ima prave stanice i oble uglove i Vaša RoundRect klasa nasledi obe osobine, i od Rectangle i od Circle, a zatim ih nasledi i od Shape, koliko će Shapeova biti kreirano kada kreirate RoundRect?

   Ukoliko se ni jedna klasa ne nasleduje korišćenjem ključne reči virtual, biće kreirana dva Shape-a: jedan za Rectangle i drugi za Shape. Ako se koristi ključna reč virtual za obe klase, tada će biti kreiran samo jedan deljivi Shape.

4. Ako se Horse i Bird nasleduju od Animal a, koristeći public virtuelno nasledivanje, da li njihovi konstruktori inicijalizuju Animal konstruktora? Ako se Pegasus nasleduje i iz Horse i iz Bird, kako on inicijalizuje Animalovog konstuktora?

   I Horse i Bird će inicijalizovati njihovu baznu klasu Animal u svojim konstruktorima. To će učiniti i Pegasus. Kada se Pegasus kreira, inicijalizacije Animal a koje izvrše Horse i Bi rd biće ignorisane.

5. Deklarišite klasu Vechi cl e i napravite je da bude apstraktni tip podataka.

```
class Vehicle
{
    virtual void MoveQ = 0;
}
```

6. Ako je bazna klasa neki ADT i ima tri čiste virtuelne funkcije, nad koliko od njih mora da se izvrši override u njenim izvedenim klasama?

   Ni nad jednom ne mora da se izvrši override, osim ako ne želite da napravite klasu koja nije apstraktna, u kom slučaju override mora da se izvrši nad sve tri funkcije.

Vezbe

1. Prikažite deklaraciju za klasu JetPlane, koja se nasleduje iz Rocket i Airplane,

```
class JetPlane : public Rocket, public Airplane
```

2. Prikažite deklaraciju za **747,** koja se nasleduje iz JetPlane klase, opisane u Vežbi **1.**

```
class 747 : public JetPlane
```

3. Napišite program koji izvodi Car i Bus iz klase Vechicle. Neka Vechicle bude ADT sa dve čiste virtuelne funkcije. A neka Car i Bus ne budu ADT

```
class Vehicle
{
     virtual void MoveQ = 0;
     virtual void HaulQ = 0;
};

class Car : public Vehicle
{
     virtual void MoveQ;
     virtual void HaulQ;
};

class Bus : public Vehicle
{
     virtual void MoveQ;
     virtual void HaulQ;
};
```

4. Izmenite program iz vežbe 3, tako da Car bude ADT i izvedite SportCar, Wagon i Coupe iz Car. U klasi Car obezbedite implementaciju za jednu čistu virtuelnu funkciju u Vechicle, koja neće biti čista (virtuelna funkcija).

```
class Vehicle
{
     virtual void MoveQ = 0;
     virtual void HaulQ = 0;
};

class Car : public Vehicle
{
     virtual void MoveQ;
```



```
class Bus : public Vehicle
{
     virtual void MoveQ;
     virtual void Haul();


class SportsCar : public Car
{
     virtual void Haul();


class Coupe : public Car
{
     virtual void Haul();
```

Dan 14

Kviz

1. Mogu li statičke promenljive članovi da budu private?

   Da. One su promenljive članovi i pristup njima može biti kontrolisan. Ako su private, može im se pristupiti samo uz pomoć funkcija cianova, ili, što je vise uobičajeno, uz pomoć statičkih funkcija cianova.

2. Prikažite deklaraciju za statičke promenljive clanove.

```
static int itsStatic;
```

3. Prikažite deklaraciju za statičke pointere funkcija.

```
static int SomeFunctionQ;
```

4. Prikažite deklaraciju za pointere na funkciju koja vraća long, a uzima integer kao parametar.

```
long (* function)(int);
```

5. Izmenite pointer iz vežbe 4, tako da on bude pointer na funkciju člana klase Car.

```
long ( Car::*function)(int);
```

6. Prikažite deklaraciju za niz od 10 pointera, kao što je definisano u pitanju 5.

```
long ( Car::*function)(int) theArray [10];
```

Veibe

1. Napišite kratak program koji deklariše klasu sa jednom promenljivom članom i jednom statičkom promenljivom članom. Neka konstruktor inicijalizuje promenljivu člana i poveća statičku promenljivu člana. Uključite destruktor, koji će umanjiti promenljivu člana.

```
1:    class myClass
2:    {
3:     public:
4:       myClassO;
5:       -myClassO;
6:     private:
7:       int  itsMember;
8:       static  int  itsStatic;
9:    };
10:
11:   myClass::myClass():
12:     itsMember(l)
13:   {
14:       itsStatic++;
15:   }
16:
17:   myClass::-myClassO
18:   {
19:       itsStatic-;;
20:   }
21:
22:    int myClass::itsStatic = 0;
23:
24:    int main()
25:    {}
```

2. Koristeći program iz vežbe **1,** napišite kratak drajver program, koji će da kreira tri objekta i da, zatim, prikaže tri promenljive člana i statičku promenljivu člana. Zatim, uništite sve objekte i prikažite efekat na statičkoj promenljivoj članu.

```
1:    #include <iostream.h>
2:
3:    class myClass
4:    {
5:     public:
6:       myClassO;
7:       -myClassO;
8:       void ShowMemberO;
9:       void ShowStaticQ ;
10:    private:
11:      int  itsMember;
12:      static  int  itsStatic;
13:   };
```


```
14:
15:   myClass::myClass():
16:    itsMember(l)
17:
18:      itsStatic++;
19:
20:
21:   myClass::-myClassO
22:
23:      itsStatic-;;
24:      cout « "In destructor. ItsStatic: " « itsStatic « endl;
25:
26:
27:   void myClass::ShowMemberO
28:
29:      cout « "itsMember: " « itsMember « endl;
30:
31:
32:    oid myClass::ShowStatic()
33:
34:      cout « "itsStatic: " « itsStatic « endl;
35:
36:   int myClass::itsStatic = 0;
37:
38:   int  main()
39:   {
40:      myClass  objl;
41:      objl.ShowMemberO;
42:      objl.ShowStaticO;
43:
44:      myClass obj2;
45:      obj2.ShowMemberO;
46:      obj2.ShowStatic()^
47:
48:      myClass obj3;
49:      obj3.ShowMemberO;
50:      obj3.ShowStatic();
51:    return 0;
52:
```

3. Izmenite program iz vežbe 2, tako da koristite statičku funkciju člana za pristupanje statičkoj promenljivoj članu. Neka statička promenljiva clan bude private (privatna).

```
#include <iostream.h>

class myClass
{
public:
   myClassO;
```

```
7:       -myClassO;
8:       void ShowMemberO;
9:       static int GetStaticQ;
10:   private:
11:      int itsMember;
12:      static int itsStatic;
13:   h
14:
15:   myClass::myClass():
16:    itsMember(l)
17:   {
18:      itsStatic++;
19:
20:
21:   myClass::-myClassO
22:   {
23:      itsStatic-;;
24:      cout « "In destructor. ItsStatic: " « itsStatic « endl;
25:   }
26:
27:   void myClass::ShowMemberO
28:   {
29:      cout « "itsMember: " « itsMember « endl;
30:
31:
32:   int myClass::itsStatic = 0;
33:
34:   void myClass::GetStatic()
35:   {
36:      return itsStatic;
37:   }
38:
39:   int main()
40:   {
41:      myClass objl;
42:      objl.ShowMemberO;
43:      cout « "Static: " « myClass::GetStatic() « endl;
44:
45:      myClass obj2;
46:      obj2.ShowMemberO;
47:      cout « "Static: " « myClass::GetStatic() « endl;
48:
49:      myClass obj3;
50:      obj3.ShowMemberO;
51:      cout « "Static: " « myClass::GetStatic() « endl;
52:    return 0;
53:
```

4. Napišite pointer na funkciju člana za pristup nestatičkom podatku članu u programu iz vežbe 3 i upotrebite taj pointer za prikaz vrednosti tog podatka.


```
   linclude <iostream.h>

   class myClass
   {
   public:
      myClassO;
      -myClassO;
      void ShowMemberO;
      static int GetStaticO;
10: private:
      int itsMember;
12:   static int itsStatic;
13: };
14:
15: myClass::myClass():
16:  itsMember(1)
17: (
18:    itsStatic++;
19: }
20:
21: myClass::-myClassO
22: {
23:    itsStatic-;;
24:    cout « "In destructor. ItsStatic: " « itsStatic « endl·
25: }
26:
27: void myClass::ShowMemberO
28: {
29:    cout « "itsMember: " « itsMember « endl;
30: }
31:
32: int myClass: .-itsStatic = 0;
33:
34: int myClass::GetStatic()
35: {
36:    return itsStatic;
37:
38:
39: int main()
40: {
41:    void (myClass::*PMF) ();
42:
43:    PMF=myClass::ShowMember;
44:
45:    myClass objl;
46:    (objl.*PMF)();
47:    cout « "Static: " « myClass::GetStatic() « endl;
48:
49:    myClass obj2;
50:    (obj2.*PMF)();
```

```
51:       cout « "Static:  " « myClass::GetStatic()  « endl;
52:
53:       myClass obj3;
54:       (obj3.*PMF)();
55:       cout « "Static:  " « myClass::GetStatic()  « endl;
56:     return 0;
57:    }
```

5. Dodajte još dve promenljive člana klasi iz prethodnog pitanja. Dodajte funkciju koja će preuzeti vrednost i dodeliti svim funkcijama clanovima iste povratne vrednosti i potpise. Upotrebite pointer na funkciju člana za pristup ovim funkcijama.

```
1:     linclude <iostream.h>
2:
3:     class myClass
4:     {
5:     public:
6:        myClassO;
7:        -myClassO;
8:        void ShowMemberO;
9:        void ShowSecond();
10:       void  ShowThirdO;
11:       static  int  GetStaticQ;
12:    private:
13:       int  itsMember;
14:       int itsSecond;
15:       int  itsThird;
16:       static  int  itsStatic;
17:    };
18:
19:    myClass::myClass():
20:     itsMember(l),
21:     itsSecond(2),
22:      itsThird(3)
23:    {
24:      itsStati C++;
25:    }
26:
27:    myClass::-myClass()
28:    {
29:       itsStatic-;;
30:       cout « "In destructor.  ItsStatic:  " « itsStatic « endl;
31:    }
32:
33:    void myClass::ShowMemberO
34:    {
35:       cout « "itsMember:  " « itsMember « endl;
36:    }
37:
```



```
38:    void myClass::ShowSecond()
39:
40:       cout « "itsSecond:  " « itsSecond « endl;
41:
42:
43:    void  myClass::ShowThird()
44:
45:       cout « "itsThird:  " « itsThird « endl;
46:
47:    nt myClass::itsStatic = 0;
48:
49:    nt  myClass::GetStatic()
50:
51:       return itsStatic;
52:
53:
54:    nt  main()
55:
56:       void (myClass::*PMF) ();
57:
58:       myClass  objl;
59:       PMF=myClass::ShowMember;
60:       (objl.*PMF)();
61:       PMF=myClass::ShowSecond;
62:       (objl.*PMF)();
63:       PMF=myClass::ShowThird;
64:       (objl.*PMF)();
65:       cout « "Static:  " « myClass::GetStatic()  « endl;
66:
67:       myClass obj2;
68:       PMF=myClass::ShowMember;
69:       (obj2.*PMF)();
70:       PMF=myClass::ShowSecond;
71:       (obj2.*PMF)();
72:       PMF=myClass: :ShowThird;
73:       (obj2.*PMF)();
74:       cout « "Static:  " « myClass::GetStatic()  « endl;
75:
76:       myClass obj3;
77:       PMF=myClass::ShowMember;
78:       (obj3.*PMF)();
79:       PMF=myClass::ShowSecond;
80:       (obj3.*PMF)();
81:       PMF=myClass: .-ShowThird;
82:       (obj3.*PMF)();
83:       cout « "Static:  " « myClass::GetStatic()  « endl;
84:    return 0;
85:
```

Dan 15

Kviz

1. Kako ćete ostvariti *je* relaciju?

   Uz pomoć javnog nasledivanja.

2. Kako ćete ostvariti *ima* relaciju?

   Kontejnerima; to je kada jedna klasa ima clan koji je objekat drugog tipa.

3. Koja razlika je između kontejnera i delegacija?

   Kontejneri opisuju ideju, da jedna klasa ima podatak clan koji je objekat drugog tipa. Delegacija izražava ideju da jedna klasa koristi drugu klasu za izvršenje zadataka ili ciljeva. Delegacija se, obično, završava uz pomoć kontejnera.

4. Koja je razlika između delegacije i implementiran-u-uslovima-od?

   Delegacija izražava ideju da jedna klasa koristi drugu klasu, kako bi izvršila zadatke, ili ciljeve. Implementiran-u-uslovima-od izražava ideju nasledivanja implementacije neke druge klase.

5. Sta je prijateljska funkcija?

   Prijateljska je funkcija koja je deklarisana da ima pristup Protected i Private clanovima Vase klase.

6. Šta je prijateljska klasa?

   Prijateljska je klasa koja je deklarisana tako da su sve njene funkcije članovi prijateljske funkcije Vaše klase.

7. Ako je Dog prijatelj od Boy, da li je Boy prijatelj Dogu?

   Ne. Prijateljstvo nije komutativno.

8. Ako je Dog prijatelj Boyu, i Terri er se izvodi iz Doga, da li je Terri er prijatelj Boya?

   Ne. Prijateljstvo se ne nasleduje.

9. Ako je Dog prijatelj Boya, i Boy je prijatelj Housea, da li je Dog prijatelj Housea?

   Ne. Prijateljstvo nije asocijativno.

10. Gde bi deklaracija prijateljske funkcije morala da se pojavi?

    Bilo gde unutar deklaracije klase. Nije bitno da li ćete deklaraciju smestiti u public:, protected:, ili private zonu.

Vežbe

1. Prikažite deklaraciju klase Animal koja sadrži podatak clan koji je string objekat.

```
class Animal:
{
private:
   String itsName;
};
```

2. Prikažite deklaraciju klase BoundedArray, koja je niz.

```
class boundedArray : public Array
{
//...
}
```

3. Prikažite deklaraciju klase Set, koja je deklarisana u uslovima od jednog niza.

```
class Set : private Array
{
// ...
}
```

4. Modifikujte listing 15.1, kako biste omogućili klasi Stri ng da koristi operator za ekstrakciju (»).

```
1:      #include <iostream.h>
2:      linclude <string.h>
3:
4:      class String
5:      {
6:         public:
7:            // konstruktori
8:            StringO;
9:             String(const char *const);
10:            String(const String &);
11:           -StringO;
12:
13:           // preklopljeni operatori
14:           char & operator[](int offset);
15:           char operator[](int offset) const;
16:           String operator+(const String*);
17:           void operator+=(const String*);
18:           String & operator= (const String &);
19:           friend ostream* operator«
20:               ( ostream*  _theStream,String* theString);
21:           friend istream* operator»
22:              ( istream* _theStream,String* theString);
23:           // opšte metode pristupa
24:           int GetLenOconst { return itsLen; }
25:           const char * GetStringO const { return itsString; }
```

```
        // static int ConstructorCount;

      private:
         String (int);         // privatni konstruktor
         char * itsString;
         unsigned short itsLen;


ostream* operator«( ostream* theStream,String* theString)
{
    theStream « theString.GetStringO;
    return theStream;
}

istream* operator»( istream* theStream,String* theString)
{
    theStream » theString.GetStringO;
    return theStream;


int main()
{
   String theString("Hello world.");
   cout « theString;
return 0;
```

5. ISTERIVAČI BAGOVA: šta nije u redu sa ovim programom:

```
linclude <iostream.h>

class Animal;

void setValue(Animal& , int);


class Animal
{
public:
   int GetWeightOconst { return itsWeight; }
   int GetAgeO const { return itsAge; }
private:
   int  itsWeight;
   int  itsAge;


void setValue(Animal& theAnimal, int theWeight)
{
   friend class Animal;
   theAnimal.itsWeight = theWeight;
```



```
int main()
{
   Animal peppy;
   setValue(peppy,5);
 return 0;
```

Ne možete smestiti friend deklaraciju u funkciju. Morate deklarisati da je funkcija prijateljska u klasi.

6. Ispravite listing iz vežbe 5, tako da prode kompilaciju.

```
linclude <iostream.h>

class Animal;

void setValue(Animal* , int);


class Animal
{
public:
   friend void setValue(Animal&, int);
   int GetWeight()const { return itsWeight; }
   int GetAgeO const { return itsAge; }
private:
   int  itsWeight;
   int  itsAge;


void setValue(Animal& theAnimal, int theWeight)
i
   theAnimal.itsWeight = theWeight;
}

int main()
{
   Animal peppy;
   setValue(peppy,5);
 return 0;
```

7. ISTERIVAČI BAGOVA: Šta nije u redu sa sledećim kodom:

```
linclude <iostream.h>

class Animal;

void setValue(Animal& , int)
```

```
6:    void setValue(Animal& .int.int);
7:
8:    class Animal
9:    {
10:   friend void setValue(Animal& ,int); // ovde je promena!
11:   private:
12:      int itsWeight;
13:      int itsAge;
14:   };
15:
16:   void setValue(Animal& theAnimal, int theWeight)
17:   {
18:       theAnimal.itsWeight = theWeight;
19:   }
20:
21:
22:   void setValue(AnimaU theAnimal, int theWeight, int theAge)
23:   {
24:      theAnimal.itsWeight = theWeight;
25:      theAnimal.itsAge = theAge;
26:   }
27:
28:   int main()
29:   {
30:      Animal peppy;
31:      setValue(peppy,5);
32:      setValue(peppy,7,9);
33:     return 0;
34:   }
```

Funkcija SetValue(Animal&,int) je deklarisana kao prijateljska, ali, funkcija nad kojom je izvršen overload SetValue(Animal&,int,int) nije deklarisana kao prijateljska.

8. Ispravite vežbu 7, tako da prode kompilaciju.

```
1:    linclude <iostream.h>
2:
3:    class Animal;
4:
5:    void setValue(Animal& , int);
6:    void setValue(Animal& ,int,int); // ovde je promena!
7:
8:    class Animal
9:    {
10:   friend void setValue(Animal& ,int);
11:   friend void setValue(Animal& ,int,int);
12:   private:
13:      int itsWeight;
14:      int itsAge;
15:   };
```



```
16:
17:   void setValue(AnimaH theAnimal, int theWeight)
18:   {
19:       theAnimal.itsWeight = theWeight;
20:   }
21:
22:
23:   void setValue(Animal& theAnimal, int theWeight, int theAge)
24:   {
25:      theAnimal.itsWeight = theWeight;
26:      theAnimal.itsAge = theAge;
27:
28:
29:   int main()
30:   {
31:      Animal peppy;
32:      setValue(peppy,5);
33:      setValue(peppy,7,9);
34:    return 0;
35:   }
```

Dan 16 /

Kviz

1. Šta je operator za Insert i čemu on služi?

   Insert operator («) je operator član ostream objekta i koristi se za pisanje na izlaznu jedinicu.

2. Sta je operator za ekstrakciju i čemu on služi?

   Operator za ekstrakciju (») je operator istream objekta i koristi se za pisanje u Vaše programske promenljive.

3. Koje su tri forme cin.getO i koja je razlika između njih?

   Prva forma get() je bez parametara. Ona vraća vrednost pronadenog karaktera i vratiće EOF, ako je dostignut kraj datoteke.

   Druga forma ci n.get () uzima karakter referencu za parametar. Taj karakter će biti popunjen sledećim karakterom u ulaznom strem-a. Vraćena vrednost je iostream objekat.

   Treća forma cin.getO uzima za parametre niz, maksimalni broj karaktera koji treba da preuzme i karakter za terminaciju. Ova forma get() popunjava niz sa za jedan manjim brojem karaktera od maksimuma, osim ako naide na karakter za terminaciju, u kom slučaju upisuje Nul 1, a karakter za terminaciju ostavlja u baferu.

4. Koja je razlika između cin.read() i cin.getline()?

```
cin.read() se koristi za čitanje binarnih struktura podataka
getline() se koristi za čitanje iz istream bafera.
```

5. Koja je podrazumevana širina za prikaz long integer-a kada se koristi Insert operator?

To je širina dovoljna za prikaz celog broja.

6. Šta vraća insert operator?

eferencu na istream objekat.

7. Koje parametre uzima konstruktor za ofstream objekat?

Ime datoteke koju treba otvoriti.

8. Šta radi argument ios::ate?

Smešta Vas na kraj datoteke, ali niste u mogućnosti da pišete nigde u datoteci.

Vežbe

Napišite program koji piše na četiri standardna iostream objekta: cin, cout, cerr i clog.

```
1:    linclude <iostream.h>
2:    int main()
3:    {
4:       int x;
5:       cout « "Enter a number: ";
6:       cin » x;
7:       cout « "You entered: " « x «  endl;
8:       cerr « "Uh oh, this to cerr!" « endl;
9:       clog « "Uh oh, this to clog!" « endl;
10:    return 0;
11:   }
```

2. Napišite program koji od korisnika traži da unese svoje ime i prezime i zatim ih prikazuje na ekranu.

```
linclude <iostream.h>
int main()
{
   char name[80];
   cout « "Enter your full name: ";
   cin.getline(name,80);
   cout « "\nYou entered: " « name « endl;
return 0;
}
```


3. Prepravite listing 16.9 da radi isti "posao", ali bez korišćenja putback(), ili ignoreQ.

```
1:     // Listing
2:     linclude <iostream.h>
3:
4:     int main()
5:     {
6:        char ch;
7:        cout « "enter a phrase: ";
8:        while ( cin.get(ch) )
9:        {
10:          switch (ch)
11:          {
12:            case '!':
13:               cout « '$';
14:               break;
15:            case '#':
16:               break;
17:             default:
18:               cout « ch;
19:               break;
20:         }
21:       }
22:    return 0;
23:    }
```

4. Napišite program koji uzima ime datoteke kao parametar i otvara datoteku za čitanje. Pročitajte sve karaktere iz datoteke, prikažite samo slova i interpunkciju na ekranu. (Ignorišite sve neprintabilne karaktere.) Zatim zatvorite datoteku.

```
1:    linclude <fstream.h>
2:    enum BOOL { FALSE, TRUE };
3:
4:    int main(int argc, char**argv)    // vraja 1 za grešku
5:    {
6:
7:       if (argc != 2)
8:       {
9:          cout « "Usage: argv[0] <infile>\n";
10:          return(l);
11:      }
12:
13:    // otvara ulazni tok
14:      ifstream fin (argv[l],ios::binary);
15:      if (!fin)
16:      {
17:         cout « "Unable to open " « argv[l] « " for reading.\n";
18:         return(l);
```

```
    char ch;
    while ( fin.get(ch))
       if ((ch > 32 && ch < 127) jj ch == '\n' jj ch == '\t')
          cout « ch;
    fin.closeO;
}
```

5. Napišite program koji prikazuje njegove argumente iz komandne linije u obrnutom redosledu, a ne prikazuje ime programa.

```
linclude <fstream.h>

int main(int argc, char**argv)    // Vra}a 1 za grešku
{
   for (int ctr = argc; ctr ; ctr-))
     cout « argv[ctr] « • ";
```

Dan 17

Kviz

1. Šta je zaštita od uključivanja?

   Zaštita od uključivanja se koristi kako bi se izbeglo uključivanje datoteke zaglavlja u jedan program vise puta.

2. Kako ćete objasniti Vašem kompajleru da prikaže sadržaj pomoćne datoteke, da biste prikazali efekte pretprocesora?

   Na ovo pitanje morate odgovoriti sami, s obzirom da odgovor zavisi od kompajlera koji koristite.

3. Koja je razlika između `#def i ne debug 0` i `lundef debug`?

   #define debug 0 definiše da izraz debug bude jednak 0. Gde god se pronađe, reč debug biće zamenjena sa `0`. `#undef debug` će ukloniti sve definicije `debug-a`; kada se pronađe reč debug u datoteci, ona će biti ostavljena neizmenjena.

4. Navedite četiri predefinisana makroa.

   `_DATE_, _TIME_,_FILE_,_LINE_`

5. Zašto ne možete pozvati `i nvari ants ()` u prvoj liniji Vašeg konstruktora?

   Zadatak Vašeg konstruktora je da kreira objekat. Različite varijante klase ne mogu i ne bi trebalo da postoje, pre nego što je objekat u potpunosti kreiran, pa će, stoga, smisleno koriscenje `i nvari ants ()` vraćati `FALSE`, sve dok konstruktor ne završi sa radom.

Vežbe

1. Napišite naredbe za zaštitu od uključivanja za datoteku zaglavlja `STRING.H`.

```
#1fndef STRING_H
ldefine STRING_H

lendif
```

2. Napišite assert () makro, koji prikazuje poruku o grešci, datoteku i broj linije, ako je nivo debagovanja 2, samo poruku (bez datoteke i broja linije), ako je nivo 1, i ništa, ako je nivo 0.

```
 1:    linclude <iostream.h>
 2:
 3:    lifndef DEBUG
/4:    ldefine ASSERT(x)
 5:    leiif DEBUG == 1
 6:    ldefine ASSERT(x) \
 7:      if (! (x)) \
 8:      { \
 9:         cout « "ERROR!! Assert " « lx « " failed\nM; \
10:              }
11:    leiif DEBUG == 2
12:    ldefine ASSERT(x) \
13:      if (! (x) ) \
14:      { \
15:       cout « "ERROR!! Assert " « lx « " failed\n"; \
16:      cout « " on line " «____LINE____« "\n"; \
17:       cout « " in file " «____FILE___« "\n";  \
18: }
19:    lendif
```

3. Napišite makro `Dprint`, koji proverava da li je definisan `debug` i, ako jeste, prikažite vrednost koja je prosledena kao parametar.

```
lifndef DEBUG
ldefine DPRINT(string)
lelse
ldefine DPRINT(STRING) cout « ISTRING ;
lendif
```

4. Napišite funkciju koja prikazuje poruku o grešci. Funkcija bi trebalo da prikaže broj linije i ime datoteke u kojoj se greška desila. Imajte u vidu da se broj linije i ime datoteke prosleduju ovoj funkciji.

```
1:    linclude <iostream.h>
2:
3:      void ErrorFunc(
4:         int LineNumber,
5:         const char * FileName)
```

```
6:    {
7:        cout « "An error occurred in file ";
8:        cout « FileName;
9:        cout « " at line "
10:       cout « LineNumber « endl;
11:   }
```

5. Kako biste nazvali prethodnu funkciju za obradu greške?

```
1:    // demonstracioni program za testiranje ErrorFunc
2:    int mainQ
3:    {
4:        cout « "An error occurs on next line!";
5:        ErrorFunc(__LINE__,__FILE__);
6:    return 0;
7:    }
```

Primetićete da se _LINE_ i _FILE_ makroi koriste tamo gde se desila greška, a ne u funkciji za obradu greške. Ako ih koristite u funkciji za obradu greške, oni će prijaviti liniju i datoteku same funkcije za obradu grešaka.

6. Napišite assert () makro, koji koristi funkciju za obradu greške iz vežbe 4, i napišite drajver program, koji poziva taj assert () makro.

```
1:    linclude <iostream.h>
2:
3:    Idefine DEBUG // uključuje rukovanje greškama
4:
5:    lifndef DEBUG
6:    Idefine ASSERT(x)
7:    lelse
8:    Idefine ASSERT(X) \
9:       if (! (X)) \
10:      {  \
11:         ErrorFunc(__LINE__,__FILE__); \
12:      }
13:   lendif
14:
15:   void ErrorFunc(int LineNumber, const char * FileName)
16:   {
17:       cout « "An error occurred in file ";
18:       cout « FileName;
19:       cout « " at line ";
20:       cout « LineNumber « endl;
21:   }
22:
23:   // demonstracioni program za testiranje ErrorFunc
24:   int main()
25:   {
26:      int x = 5;
27:      ASSERT(x >= 5);  // nema greške
28:      x = 3;
29:      ASSERT(x >= 5); // greška!
30:   return 0;
31:   }
```

Primetićete da u ovom slučaju _LINE_ i _FILE_ makroi mogu biti pozvani iz assert () makroa, a da i dalje daju ispravnu liniju (linija 29). Razlog za ovo je što je assert () makro ekspandovan na mestu gde je i pozvan. Stoga, ovaj program će se izvrsavati baš kao da je main() bila napisana na sledeći način:

```
1:    // demonstracioni program za testiranje ErrorFunc
2:    int main()
3:    {
4:       int x = 5;
5:       if (! (x >= 5)) {ErrorFunc(__LINE__,__FILE__);}
6:       x = 3;
7:       if (! (x >= 5)) {ErrorFunc(__LINE__, _FILE_);}
8:    return 0;
9:    }
```

Dan 18

1. Kakva je razlika izmedu objektno-orijentisanog i proceduralnog programiranja?

   Proceduralno programiranje se fokusira na funkcije koje su nezavisne od podataka. Objektno-orijentisano programiranje povezuje podatke i funkcionalnost u objekte i fokusira se na interakciju između objekata.

2. Na šta se odnosi sintagma "event-driven"(dogadajima upravljan)?

   "dogadajima upravljani" programi se razlikuju po činjenici da se akcija sprovodi samo kao odgovor na neki dogadaj, kao što su unos sa tastature, ili dogadaji vezani za rad sa mišem.

3. Nabrojte faze razvojnog ciklusa.

   Karakteristično je da razvojni ciklus uključuje analizu, dizajn, kodiranje, testiranje, programiranje i interakciju, kao i medusobnu komunikaciju između ovih faza.

4. Šta je korena hijerarhija?

   Korena hijerarhija je ona u kojoj se sve klase jednog programa direktno, ili indirektno izvode iz jedne bazne klase.

5. Šta je drajver program?

Drajver program je funkcija dizajnirana za ispitivanja različitih objekata i funkcija koje programirate.

6. Šta je enkapsulacija?

Enkapsulacija predstavlja (željene) pokušaje spajanja u jednu klasu svih podataka i funkcionalnosti jednog entiteta.

Vežbe

1. Pretpostavimo da treba da simulirate raskrsnicu Avenije Massachusetts i pete ulice - dve tipično dvosmerne saobraćajnice, na kojima su postavljeni semafori i pešački prelazi. Smisao ove simulacije je da odredite da li vreme na saobraćajnom znaku (semaforu) obezbeduje lagano odvijanje saobraćaja.

Koje vrste objekata bi trebalo modelirati za potrebe simulacije? Koje bi klase trebalo da postoje u ovoj simulaciji?

Automobili, motocikli, kamioni, bicikli, pešaci i razna vozila hitnih službi koriste raskrsnicu. Takođe, postoji i semafor za pešake.

Da li bi površina puta trebalo da bude uključena u simulaciju.? Naravno. Kvalitet puta može imati uticaja na saobraćaj, ali u prvom dizajnu bilo bi jednostavnije izostaviti ovaj uslov.

Prvi objekat je, verovatno, sama raskrsnica. Pretpostavimo da objekatraskrsnica održava listu kola koja čekaju da produ kroz semafor u svakom pravcu, kao i listu ljudi koji čekaju da predu ulicu. Ovo će zahtevati metode za izbor koja i koliko kola i ljudi treba da produ kroz raskrsnicu. Ovde će biti samo jedna raskrsnica, tako da bi trebalo da razmislite na koji način ćete se uveriti da je samo jedan objekat instantiniran (razmislite o statičkim metodama i zaštićenom pristupu).

Ljudi i kola su korisnici raskrsnice. Oni dele veći broj karakteristika: mogu se pojaviti u bilo kom trenutku, može ih biti bilo koji broj i svi čekaju na signal (iako u različitim linijama). Ovo nam nalaže da razmislimo o zajedničkoj baznoj klasi za pešake i za kola.

Stoga bi klase trebalo da uključe:

```
class Entity;    // klijent intersekcije

// osnova automobila, kamiona, biciklova, i vozila za hitne slučajeve
class Vehicle : Entity ...;

// osnova svih ljudi
class Pedestrian : Entity...;

class Car : public Vehicle...;
```



```
class Truck : public Vehicle...;
class Motorcycle : public Vehicle...;
class Bicycle : public Vehicle...;
class EmergencyJ/ehicle : public Vehicle...;

// sadrži listu automobila i ljudi koji žekaju na prolaz
class Intersection;
```

2. Pretpostavimo da se raskrsnica iz vežbe **1** nalazila u predgradu Bostona, koji, složićete se, ima "najneprijateljskije" ulice u SAD. U svako vreme, tim ulicama se kreću tri vrste bostonskih vozača:

lokalni, to su oni koji prolaze kroz raskrsnicu, iako se na semaforu upalilo crveno svetlo turisti, koji voze lagano i obazrivo (kolima rent-a-car-a, što je vrlo tipično), i tajcsisti, kod kojih se može zapaziti široka paleta stilova vožnje, u zavisnosti od vrste putnika u vozilu.

Takode, Bostonom se kreću dve vrste pešaka: lokalni, koji prelaze ulicu gde god im se svidi i gotovo nikad ne koriste dugme (na semaforu) za prelaz pešaka preko ulice i turisti, koji uvek koriste pomenuto dugme i prelaze preko ulice samo na zeleno (svetlo). Na kraju, Boston ima i bicikliste, koji nikada ne obraćaju pažnju na crveno svetlo. Kako će ovi uslovi uticati na model?

Sasvim razuman početak bi bio da kreirate izvedene objekte koji modeliraju činjenice prikazane u problemu:

```
class Local_Car : public Car...;
        class Tourist_Car : public Car...;
        class Taxi : public Car...;
        class Local_Pedestrian : public
Pedestrian...;
        class Tourist_Pedestrian : public
Pedestrian...;
        class Boston_Bicycle : public Bicycle...;
```

Korišćenjem virtuelnih metoda, svaka klasa može da modifikuje generičko ponašanje, kako bi zadovoljila sopstvenu specifikaciju. Na primer, bostonski vozači mogu reagovati na crveno svetlo na drugačiji način nego što to rade turisti, a da i dalje nasleduju generičko ponašanje koje se primenjuje.

3. Zamoljeni ste da dizajnirate plan rada grupe. Softver Vam omogućava da ogranizujete sastanke između pojedinaca, ili grupa i da rezervišete određeni broj sala za konferencije. Identifikujte glavne podsisteme.

Za ovaj projekat je potrebno napraviti dva konkretna programa: program-klijent, koji će koristiti korisnici, i program-server, koji će se izvršavati na odvojenoj mašini. Dodatno, mašina klijenta će imati administrativnu komponentu, koja će omogudti administratoru sistema da dodaje nove ljude i prostorije.

Ako ste odlučili da implementaciju izvršite u klijent/server modelu, klijenti bi trebalo da prihvataju ulaz od korisnika i da generišu zahteve serveru. Server bi trebalo da servisira zahteve i da vraća rezultate nazad klijentima. Na osnovu ovog modela, mnogo ljudi može da organizuje sastanke u istom trenutku.

Sa klijentove strane, uz administativni modul, postoje još dva glavna podsistema: korisnički interfejs i komunikacioni podsistem. Serverova strana se sastoji od tri glavna podsistema: komunikacija, raspodela i mail interfejs - oni će obaveštavati korisnike o izmenama koje su se "dogodile" u rasporedu.

4. Dizajnirajte i prikažite interfejs za klase u delu za rezervaciju soba iz programa o kome je bilo reči u vežbi 3.

Sastanak je definisan kao grupa ljudi koja je rezervisala prostoriju na odredeni period vremena. Osoba zadužena da pravi raspored može da zahteva odredenu prostoriju, ili odredeno vreme, ali u rasporedu uvek mora biti rečeno koliko dugo će sastanak trajati i ko će biti prisutan.

Objekti će, verovatno, sadržati korisnike sistema, kao i prostorije za sastanke. Nemojte zaboraviti da uključite klase za kalendar i, recimo, meeting klasu, koja vrši enkapsulaciju svega što je poznato o odredenom dogadaju.

Prototipovi klasa bi mogli da sadrže:

```
class Calendar_Class;          // ranija referenca
class Meeting;                 // ranija referenca
class Configuration
{
public:
     Configuration();                          ^
     -Configuration();
     Meeting Schedule( ListOfPerson*, Delta Time
duration );
     Meeting Schedule( ListOfPerson&, Delta Time
duration, Time );
     Meeting Schedule( ListOfPerson*, Delta Time
duration, Room );
     ListOfPerson*      People();      // javna
accessors
     ListOfRoom*      Rooms();      // javne metode pristupa
protected:
     ListOfRoom      rooms;
     ListOfPerson      people;
};
typedef long       Room_ID;
class Room
(
public:
     Room( String name, Room_ID id, int capacity,
String directions = "", String description = "" );
     -Room();
     Calendar_Class Calendar();

protected:
     Calendar_Class      calendar;
     int            capacity;
     Room_ID      id;
     String            name;
     String            directions;          // gde je
ovaj prostor?
     String            description;
I;
typedef long Person_ID;
class Person
{
publi c:
     Person( String name, Person_ID id );
     -Person();
     Calendar_Class Calendar();          // pristupna
tačka za dodavanje sastanaka
protected:
     Calendaredass      calendar;
     Person_ID      id;
     String            name;
};
class Calendar_Class
{
public:
     Calendar_Class();
     -Calendar Class();

     void Add( const Meeting* );               // dodaje
sastanak u kalendar
     void Delete( const Meeting* )
     Meeting* Lookup( Time );               // proverava da li
se sastanak održava u
                                   // dato vreme

     Block( Time, Duration, String reason = "" );
// alocira sebi vreme...

protected:
     OrderedListOfMeeting meetings;
};
class Meeting
{
publi c:
     Meeting( ListOfPerson*, Room room,
          Time when, Duration duration, String purpose
= "" );
```

```
    -MeetingO;
protected:
    ListOfPerson     people;
    Room             room;
    Time             when;
    Duration         duration;
    String           purpose;
```

Dan 19

1. Koja je razlika izmedu templejta i makroa?

   Templejti su ugradeni u C++ jezik i vode računa o tipovima. Makroi su implementirani uz pomoć pretprocesora i ne vode računa o tipovima.

2. Koja je razlika između parametara u templejtu i u funkcijama?

   Parametri za templejte kreiraju instancu templejta za svaki tip. Ako kreirate šest instanci templejta, kreiraće se šest različitih klasa, ili funkcija. Parametri funkcija menjaju ponašanje, ili podatke funkcije, ali se kreira samo jedna funkcija.

3. Koja je razlika između templejta specifičnog tipa prijateljskih klasa i opšteg templejta prijateljske klase?

   Opšte templejt prijateljske funkcije kreiraju po jednu funkciju za svaki tip parametarizovane klase; funkcije specifičnog tipa kreiraju instance specifičnog tipa za svaku instancu parametarizovane klase.

4. Da li je moguće obezbediti specijalno ponasanje za jednu instancu templejta, ali ne i za ostale?

   Da. Kreirajte specijalizovanu funkciju za odredenu instancu. Kada kreirate `Array<t>::SomeFunction()` takode kreirajte `Array<int>::someFunction()`, kako biste promenili ponasanje za celobrojne nizove.

5. Koliko statičkih promenljivih će biti kreirano, ako stavite jedan statički član u definiciju templejt klase?

   Jedan za svaku instancu klase.

Vežbe

1. Kreirajte templejt na osnovu sledeće `List` klase:

```
class List
{
```



```
private:

public:       -*
    List():head($,tai1(0).theCount(0) {}
    virtual -List();

    void insert( int value );
    void append( int value );
    int is_present( int value ) const;
    int is_empty() const { return head ==0; }
    int count() const { return theCount; }
private:
    class ListCell
    {
    public:
        ListCell(int value, ListCell *cell = 0):val(value),next(cell){}
        int val;
        ListCell *next;            X
    };
    ListCell *head;
    ListCell Hail;
    int theCount;
```

h

Evo jednog načina za implementaciju ovog templejta:

```
template <class Type>
class List
{

publi c:
    List():head(0),tail(0),theCount(0) { }
    virtual -List();

    void insert( Type value );
    void append( Type value );
    int is_present( Type value ) const;
    int is_empty() const { return head == 0; }
    int count() const { return theCount; }

private:
    class ListCell
    {
    publi c:
        ListCell(Type value, ListCell *cell = 0):val(value),next(cell){}
        Type val;
        ListCell *next;
    };

    ListCell *head;
    ListCell 4ail;
```

```
    int  theCount;
};
```

2. NapiSite implementaciju za List klasu (netemplejt).

```
void  List::insert(int  value)
{
    ListCell *pt = new ListCel1 ( value, head );
    assert (pt != 0);

    // ova linija je dodana za rukovanje repom
    if ( head == 0 ) tail = pt;

    head = pt;
    theCount++;
}

void List::append( int value )
{
    ListCell *pt = new ListCell( value );
    if ( head == 0 )
        head = pt;
    else
        tail->next = pt;

    tail = pt;
    theCount++;
}

int List: :is_present( int vahte ) const
{
    if ( head == 0 ) return 0;
    if ( head->val == value |j tail->val == value )
        return 1;

    ListCell *pt = head->next;
    for (; pt != tail; pt = pt->next)
        if ( pt->val == value )
            return 1;

    return 0;
}
```

3. Napišite templejt verziju implementacije.

```
template <class Type>
List<Type>::-Li st()
{
    ListCell *pt = head;

    while ( pt )
    {
        ListCell *tmp = pt;
```



```
        pt • pt->next;
        delete tmp;
    }
    head = tail = 0;
}

template <class Type>
void  List<Type>::insert(Type  value)
{
    ListCell *pt = new ListCell( value, head );
    assert (pt != 0);

    // ova linija je dodana za rukovanje repom
    if ( head == 0 ) tail = pt;

    head = pt;
    theCount++;
}

template <class Type>
void List<Type>::append( Type value )
{
    ListCell *pt = new ListCel1( value );
    if ( head == 0 )
        head = pt;
    else
        tail->next = pt;

    tail = pt;
    theCount++;
)

template <class Type>
int List<Type>::is_present( Type value ) const
{
    if ( head == 0 ) return 0;
    if ( head->val == value || tai1->val == value )
        return 1;

    ListCell *pt = head->next;
    for (; pt !- tail; pt = pt->next)
        if ( pt->val == value )
            return 1,

    return 0;
}
```

4. Deklarišite tri objekta za listanje: `list of strings, list of Cats i 1 ist of ints.`

```
List<String>  string_list;
List<Cat> Cat_List;
```

```
List<int>  intList;
```

5. Lovci na greške: šta nije dobro u sledećem kodu: (imajte u vidu da je `List` templejt definisan, a da je Cat klasa, koja je već ranije definisana u knjizi).

```
List<Cat> Cat_List;
Cat Felix;
CatList.append( Felix );
cout « "Felix is " «
     ( Cat_List.is_present( Felix ) ) ? " " : "not " « "present\n";
```

Savet: (Ovo je teško): Šta čini `Cat` različitim od `int`?

`Cat` nema definisan operator==; sve operacije koje uporeduju vrednosti u ćelijama `List`-e, kao što je `is_present`, prouzrokovaće grešku u kompilaciji. Da biste smanjili šanse za ovo, smestite komentare pre definicije templejta, počinjući sa operacijama koje moraju biti definisane, kako bi prošla kompilacija.

6. Deklarišite prijateljski operator== za `List`.

```
friend int operator==( const Type& lhs, const Type& rhs );
```

7. Implementirajte prijateljski operator== za `List`.

```
template <class Type>
int List<Type>::operator==( const Type& lhs, const Type& rhs )
{
     // poredi prvo dužine
     if ( lhs.theCount != rhs.theCount )
          return 0;      // dužine se razlikuju

     ListCell *lh = lhs.head;
     ListCell *rh = rhs.head;

     for(; lh != 0; lh = lh.next, rh = rh.next )
          if ( lh.value != rh.value )
               return 0;

     return 1;           // ako se ne razlikuju moraju se podudarati
}
```

8. Da li operator== ima isti problem kao i u vežbi 5?

Da, s obzirom da operator za poredenje nizova uključuje poredenje elemenata, operator != mora biti definisan i za elemente.

9. Implementirajte templejt funkciju za swap, koja vrši zamenu vrednosti dve promenljive.

```
// šablonska swap:
// mora imati dodelu i konstruktor kopije definisane za Type (tip)
template <class Type>
void swap( Type& lhs, Type& rhs)
{
     Type temp( lhs );
     lhs = rhs;
     rhs = temp;

}
```


Dan 20

Kviz

1. Šta je izuzetak?
Izuzetak je objekat, koji je kreiran kao rezultat upotrebe ključne red throw. On se koristi za signalizaciju izuzetka i prosleduje se "na stek" poziva prvoj catch naredbi, koja podržava njegov tip.

2. Šta je try blok?

Try blok je set naredbi koje mogu da generišu izuzetak.

3. Šta je catch naredba?
Catch naredba ima potpis tipa izuzetka koji podržava. Ona sledi try blok i ponaša se kao prijemnik za izuzetke nastale u try bloku.

4. Koje informacije izuzetak može da sadrži?

Izuzetak je objekat i može da sadrži bilo koju informaciju definisanu unutar korisnički definisanih klasa.

5. Kada se kreira izuzetak objekat?

Ovaj objekat se kreira kada upotrebite ključnu reč throw.

6. Mogu li se izuzeci proslediti po vrednosti, ili po referenci?

Uopšteno, izuzeci bi trebalo da se prosleduju po referenci. Ako ne nameravate da izmenite sadržaj objekta izuzetka, trebalo bi da prosledite const referencu.

7. Da li će catch naredba uhvatiti izvedeni izuzetak, ako ona traži baznu klasu?

Da, ako prosledite izuzetak po referenci.

8. Ako postoje dve Catch naredbe, jedna za baznu, a druga za izvedenu klasu, koja će se prva izvršiti?

Catch naredbe se ispituju prema redosledu po kome se pojavljuju u izvornom kodu. Izvršiće se prva catch naredba, dji potpis odgovara izuzetku.

9. Sta znad catch (...) ?

Catch (...) će uhvatiti bilo koji izuzetak, bilo kojeg tipa.

10. **Šta je prekidna tačka?**

Prekidna tačka je mesto u kodu gde će debager prekinuti izvršavanje programa.

Vežbe

1. Kreirajte try blok, catch naredbu i jednostavan izuzetak.

```
#include <iostream.h>
class OutOfMemory {};
int main()
{

     try
     {
          int *myInt = new int;
          if (myInt == 0)
               throw OutOfMemory();
     }
     catch (OutOfMemory)
     {
          cout « "Unable to allocate memory!\n";
     }
return 0;
 }
```

2. Modifikujte odgovor iz vežbe 1, tako što ćete smestiti podatke u izuzetak, zajedno sa pristupnom funkcijom, i što ćete to iskoristiti u catch bloku.

```
linclude <iostream.h>
linclude <stdio.h>
linclude <string.h>
class OutOfMemory
{
public:
     OutOfMemory(char *);
     char* GetStringO { return itsString; }
private:
     char* itsString;
};

OutOfMemory: .-OutOfMemory(char * theType)
{
     itsString = new char[80];
     char warnings = "Out Of Memory! Can't allocate room for:
     strncpy(itsString,warning,60);
     strncat(itsString,theType,19);
}

int main()
```



```
{

     try

          int *myInt = new int;
          if (myInt == 0)
               throw OutOfMemory("int");
     }
     catch (OutOfMemory& theException)
     {
          cout « theException.GetStringO;
     }
return 0;
```

3. Modifikujte klasu iz vežbe 2, tako da bude hijerarhija izuzetaka; modifikujte catch blok, tako da koristi izvedene objekte i bazne objekte.

```
linclude <iostream.h>

// Abstraktni tip podataka za izuzetak
class Exception
{
public:
   Exception(){}
   virtual -Exception(){}
   virtual void PrintErrorQ = 0;


// Izvedena klasa za rukovanje memorijskim problemima.
// Uočite da u ovoj klasi nema alokacije memorije!
class OutOfMemory : public Exception
{
public:
   OutOfMemory(){}
   -OutOfMemory(){}
   virtual void PrintError();
private:


void OutOfMemory::PrintError()
{
   cout « "Out of Memory!!\n";


// Izvedena klasa za rukovanje lošim brojevima
class RangeError : public Exception
{
public:
   RangeError(unsigned long number){badNumber = number;}
```

```
33:       -RangeErrorQQ
34:       virtual void PrintErrorQ;
35:       virtual unsigned long GetNumberQ { return badNumber; }
36:       virtual void SetNumber(unsigned long number) {badNumber = number;}
37:    private:
38:       unsigned long badNumber;
39:    };
40:
41:    void RangeError::PrintErrorQ
42:    {
43:       cout « "Number out of range. You used " « GetNumberQ « "!!\n";
44:    }
45:
46:    void MyFunctionQ;  // funkc. prototip
47:
48:    int mainQ
49:    {
50:       try
51:       {
52:          MyFunctionQ;
53:       }
54:       // Potrebno samo jedno hvatanje, koriste se virtuelne funkcije da bi se
55:       // postiglo ono što je potrebno.
56:       catch (Exception* theException)
57:       {
58:          theException.PrintErrorQ;
59:       }
60:       return 0;
61:    }
62:
63:    void MyFunctionQ
64:    {
65:       unsigned int *myInt = new unsigned int;
66:       long testNumber;
67:       if (myInt == 0)
68:          throw OutOfMemory();
69:       cout « "Enter an int: ";
70:       cin » testNumber;
71:       // ovaj čudan test bi trebalo zameniti serijom
72:       // testova za izveštavanje o lošem korisničkom ulazu
73:       if (testNumber > 3768 1! testNumber < 0)
74:          throw RangeError(testNumber);
75:
76:       *myInt = testNumber;
77:       cout « "Ok. myInt: " « *myInt;
78:       delete myInt;
79:    }
```

4. Modifikujte program iz vežbe 3, tako da ima tri nivoa poziva funkcija.


```
1:    linclude <iostream.h>
2:
3:    // Abstraktni tip podataka za izuzetke
4:    class Exception
5:    {
6:    public:
7:       ExceptionQQ
8:       virtual -ExceptionQQ
9:       virtual void PrintErrorQ = 0;
10:   };
11:
12:   // Izvedena klasa za rukovanje memorijskim problemima
13:   // Uočite da u ovoj klasi nema alokacije memorije!
14:   class OutOfMemory : public Exception
15:   {
16:   public:
17:      OutOfMemory ()0
18:      -OutOfMemoryQQ
19:      virtual void PrintErrorQ;
20:   private:
21:   };
22:
23:   void OutOfMemory::PrintError()
24:   {
25:      cout « "Out of Memory!!\n";
26:   }
27:
28:   // Izvedena klasa za rukovanje lošim brojevima
29:   class RangeError : public Exception
30:   {
31:   public:
32:      RangeError(unsigned long number){badNumber = number;}
33:      -RangeErrorQQ
34:      virtual void PrintErrorQ;
35:      virtual unsigned long GetNumberQ { return badNumber; }
36:      virtual void SetNumber(unsigned long number) {badNumber = number;}
37:   private:
38:      unsigned long badNumber;
39:   };
40:
41:   void RangeError::PrintErrorQ
42:   {
43:      cout « "Number out of range. You used " « GetNumberQ « "!!\n";
44:   }
45:
46:   // funkc. prototipovi
47:   void MyFunctionQ;
48:   unsigned int * FunctionTwoQ;
49:   void FunctionThree(unsigned int *);
50:
```

```
51:    int main()
52:   {
53:     try
54:     {
55:        MyFunctionQ;
56:     }
57:     // Samo jedno hvatanje je potrebno, koriste se virtuelne funkcije za
58:     // postizanje željenog cilja.
59:     catch (Exception* theException)
60:     {
61:        theException.PrintErrorQ;
62:     }
63:     return 0;
64:   }
65:
66:    unsigned int * FunctionTwoQ
67:   {
68:      unsigned int *myInt = new unsigned int;
69:     if (myInt == 0)
70:       throw OutOfMemoryQ;
71:     return myInt;
72:  }
73:
74:    void MyFunctionQ
75:   {
76:        unsigned int *myInt = FunctionTwoQ;
77:
78:         FunctionThree(myInt);
79:        cout « "Ok. myInt: " « *myInt;
80:        delete myInt;
81:  }
82:
83:    void FunctionThree(unsigned int *ptr)
84:   {
85:        long testNumber;
86:        cout « "Enter an int: ";
87:        cin » testNumber;
88:        // ovaj čudan test bi trebalo da bude zamenjen serijom
89:        // testova za obaveštenje o lošem korisničkom ulazu
90:        if (testNumber > 3768 || testNumber < 0)
91:           throw RangeError(testNumber);
92:        *ptr = testNumber;
93:  }
```

5. ISTERIVAČI BAGOVA: Šta je neispravno u sledećem kodu:

```
#include "stringc.h"          // naša string klasa

class xOutOfMemory
{
public:
    xOutOfMemory( const String* where ) : location( where ){}
    -xOutOfMemory(){}
    virtual String where(){ return location };
private:
    String location;
}

mainQ
{
    try {
        char *var = new char;
        if ( var == 0 )
            throw xOutOfMemory();
    }
    catch( xOutOfMemory* theException )
    {
        cout « "Out of memory at " « theException.locationQ « "\n";
    }
}
```

U procesu obrade greške "out of memory" konstruktor xOutOfMemory je kreirao string objekat. Ovaj izuzetak se mogao dogoditi samo kada je program ostao bez memorije, tako da je i ova alokacija morala da proizvede grešku.

Vrlo je verovatno da će pokušaj kreiranja ovog stringa dovesti do istog izuzetka, koji će dovesti do beskonačne petlje i do pada programa. Ako Vam je ovaj string zaista potreban, možete alocirati prostor u statičkom baferu, pre početka programa, i zatim ga koristiti kada dode do izuzetka.

Dan 21

Kviz

1. Koja je razlika između strcpyO i strncpy()?

   strcpy(char* destination, char* source) kopira source u destination i smešta null karakter na kraj odredišta. Odredište mora da bude dovoljno veliko, kako bi prihvatilo source, ili će strcpyO, jednostavno, nastaviti da piše iza kraja niza. strncpy(char*destination, char*source, int howmany) će upisati howmany bajtova source-a u destination, ali neće staviti null karakter.

2. Šta radi ctime()?

   ctimeO uzima time_t promenljivu i vraća ASCII string sa tekućim vremenom. Promenljiva time_t se, obično, popunjava prosledivanjem njene adrese funkciji time().

3. Koju funkciju ćete pozvati da biste izvršili konverziju ASCII stringa u l ong?

   atolQ.

4. Šta radi komplement operator?

   On menja svaki bit u broju.

5. Koja je razlika izmedu **OR** i exclusive **OR?**

   OR vraća **TRUE,** ako su oba bita postavljena, ili ako je bilo koji od njih; exl usi ve **OR** će vratiti **TRUE,** samo ako je jedan od bitova tačno postavljen.

6. Koja je razlika između & i &&?

   & je AND operator za rad sa bitovima, a && je logički AND operator.

7. Koja je razlika između j i 11 ?

   j je **OR** operator za rad sa bitovima, a 11 su logički **OR** operator.

Vežbe

1. Napišite program koji će na siguran način kopirati sadržaj 20-bajtnog stringa u 10-bajtni string, odsecajući visak.

```
linclude <iostream.h>
linclude <string.h>

int  main()
{
   char bigString[21] = "12345678901234567890";
   char smallString[10];
   strncpy(smallString,bigString,9);
   smallString[9]='\0';
   cout « "BigString:  " « bigString « endl;
   cout « "smallString:  " « smallString « endl;
   return 0;
```

2. Napišite program koji će tekući datum prikazati u formi 7/28/94.

```
linclude <iostream.h>
linclude  <time.h>

int  main()
{
   time_t currentTime;
   struct  tm  *timeStruct;
   time (&currentTime);
   timeStruct  =  localtime(&currentTime);

   cout  «  timeStruct->tm_mon+l  «  "/";
   cout  «  timeStruct->tm_mday  «  "/";
   cout  «  timeStruct->tm_year  «  "  ";
   return 0;
}
```

3. Napišite definiciju klase koja koristi polja bitova da li je kompjuter mono ili kolor, PC ili Mac, laptop ili desktop i da li ima CD-ROM.

```
linclude <iostream.h>
enum Boolean { FALSE = 0, TRUE = 1

class Computer
{
public:   // tipovi
     enum Machine { Mac = 0, PC };

public:   // metode
     Computer( Boolean color, Boolean laptop, Machine kind, Boolean cdrom )
          : Color( color ), Laptop( laptop ), Kind( kind ), CDRom( cdrom ){}
     -Computer(){}

     friend ostream& operator«( ostream& os, const Computers computer );

pri vate:
     Boolean Color : 1;
     Boolean Laptop : 1;
     Machine Kind : 1;
     Boolean CORom : 1;


     ostream&
operator«( ostream& os, const Computers computer )
{
     OS « "[";
     ( computer.Col or ) ? os « "color" : os « "monochrome";
     os « ", ";
     ( computer.Laptop ) ? os « "laptop" : os « "desktop";
     os « ", ";
     ( computer.Kind ) ? os « "PC" : os « "Mac";
     os « ", ";
     ( computer.CDRom ) ? os « "" : os « "no ";
     os « "CD-Rom";
     os « "]";
     return os;
}
 int  main()
{
     Computer pc( TRUE, TRUE, Computer :: PC, TRUE );

     cout << pc « '\n';
```

```
     return 0;
}
```

4. Napišite program koji kreira 26-bitnu masku. Tražite od korisnika da unese reč, zatim kratak izveštaj koja su slova upotrebljena i čitanje bitova (jedan bit po karakteru). Program treba da tretira velika i mala slova kao da su iste veličine.

```
linclude <ctype.h>
linclude <iostream.h>
linclude <string.h>

class Bits
{
public:
     enum { BITS_PER_INT = 16 };
     Bits( int cnt );
     virtual -Bits();

     void clear();
     void set( int position );
     void reset( int position );
     int is_set( int position );
private:
     unsigned int * bits;
     int count;
     int Ints_Needed;
};

class AlphaBits : private Bits
{
public:
     AlphaBitsO : Bits( 26 ){}
     -AlphaBitsQO

     void clear() { Bits::clear(); }
     void set( char );
     void reset( char );
     int is_set( char );
};

Bits :: Bits( int cnt ) : count( cnt )
{
     Intsjeeded = count / BITS_PER_INT;

     // ako postoji ostatak, potreban vam je još jedan član u nizu
     if ( 0 != count % BITS_PER_INT )
          Ints_Needed++;

     // kreira niz celobrojnih promenljivih za čuvanje svih bitova
     bits = new unsigned int[ Ints_Needed ];
     clear();
}

Bits :: -BitsQ
{
     delete [] bits;
}

void Bits :: clearQ
{
     // izbriši bitove
     for ( int i = 0; i < Ints_Needed; i++ )
          bits[ i ] = 0;


void Bits :: set( int position )
{
     // pronalazi bit koji treba postaviti
     int Int_Number = position / BITS_PER_INT;
     int Bit_Number = position % BITS_PER_INT;

     // kreira masku sa tim jednim postavljenim bitom
     unsigned int mask = 1 « Bit_Number;

     // postavlja bit
     bits[ Int_Number ] |= mask;
}

// čisti bit
void Bits :: reset( int position )
{
     int Int_Number = position / BITS_PER_INT;
     int Bit_Number = position % BITS_PER_INT;

     unsigned int mask = -( 1 « Bit_Number );

     bits[ Int_Number ] &= mask;
}

int Bits :: is_set( int position )
{
     int Int_Number = position / BITS_PER_INT;
     int Bit_Number = position % BITS_PER_INT;

     unsigned int mask = 1 « Bit_Number;

     return ( 0 != ( bits[ Int_Number ] & mask ) );
}
```

```
void AlphaBits :: set( char s )
{

     // proverava da li je traženi karakter abecedni karakter
     // ako je tako, pretvara ga u malo slovo, zatim oduzima ascii vrednost
     // od 'a' da bi se dobijo njegov redni broj (gde je a = 0, b = 1) i postavlja
taj bit
     if ( isalpha( s ) )
          Bits :: set( tolower( s ) - 'a' );
}

void AlphaBits :: reset( char s )
{
     if ( isalpha( s ) )
          Bits :: reset( tolower( s ) - 'a' );
}

int AlphaBits :: is_set( char s )
{
     if ( isalpha( s ) )
          return Bits :: is_set( tolower( s ) - 'a' );
     else
          return 0;
}

int  main()
{
     AlphaBits  letters;

     char buffer[512];

     for (;;)
     {
          cout « "\nPlease type a word (0 to quit): ";
          cin  »  buffer;

          if (strcmp(buffer,"0") == 0)
             break;

          //  postavlja  bitove
          for ( char *s = buffer; *s; s++ )
               letters.set( *s );

          //  štampa  rezultate
          cout « "The letters used were:  ";
          for ( char c = 'a'; c <= 'z'; C++ )
               if ( letters.is_set( c ) )
                    cout « c « '
          cout « '\n';
          // briše  bitove
          letters.clear();
     }
    return 0;
```


5. Napišite program koji sortira komandne linije parametara. Ako svi korisnici unesu SortFunc cat bird fish dog, program štampa bird cat dog fish.

```
linclude <string.h>
linclude <iostream.h>

void swap ( char* &s, char* &t )
{
     char* temp = s;
     s = t;
     t = temp;
}

int main( int argc, char* argv[] )
{
     // Kako je argv*0* ime programa,
     //mi neželimo da ga sortiramo ili odštampamo
// sortiranje počinjemo od elementa 1 (ne 0).

     // "Ključalo sortiranje" se koristi zbog malog broja elemenata
     int  i,j;
     for ( i = 1 ; i < argc; i++ )
          for ( j = i + 1; j < argc; j++ )
               if ( 0 < strcmp( argv[i], argv[j] ) )
                    swap( argv[i], argv[j] );


     for ( i = 1; i < argc; i++ )
          cout « argv[i] « ' ';

     cout « '\n';
    return 0;
}
```

6. Napišite program koji će sabrati dva broja, bez korišćenja operatora + za sabiranje. Ako pogledate sabiranje dva bita, primetićete da će odgovor sadržati dva bita: rezultujući i bit za prenos. Stoga, sabiranjem binarnih 1 i 1, dobićete 1 i prenos 1. Ako saberete 101 i 001 dobićete rezultat

```
101  // 5
001  И\
110  I/6
```

Ako saberete dva postavljena bita (svaki ima vrednost 1), rezultat je takav da je rezultujući bit 0, a prenosni 1. Ako dodate dva poništena bita i rezultat i prenos

će biti **0.** Ako sabirate dva bita od kojih je jedan postavljen, a drugi poništen, rezultat će biti **1** i prenos će biti **0.** Evo tabele koja prikazuje ova pravila:

```
lhs   rhs   j   carry   result

0     0    )    0       0
0     1     j   0       1
1     0     !   0       1
1     1     1    1      0
```

Ispitajte logiku prenosnog bita. Ako su oba bita koja treba sabrati (lhs i rhs) 0, ili je bilo koji 0, odgovor je 0. Samo ako su oba bita 1, odgovor je 1. Ovo je potpuno isto kao i **AND** operator (&).

Na isti nadn rezultat je XOR operator (^), ako je bilo koji od bitova 1 (ali ne oba), odgovor je 1. Inače, odgovor je 0.

Kada dobijete prenos, on se dodaje sledećem bitu sa leve strane. Ovo ukazuje bilo na iteraciju kroz sve bitove, bilo na rekurziju.

```
linclude <iostream.h>

unsigned int add( unsigned int lhs, unsigned int rhs )
{
     unsigned int result, carry;

     while ( 1 )
     {
          result = lhs ^ rhs;
          carry = lhs & rhs;

          if ( carry == 0 )
               break;

          lhs = carry « 1;
          rhs = result;
     I;

     return  result;
}

int  main()
{
     unsigned long a, b;
     for (;;)
     {
          cout « "Enter two numbers.  (0 0 to stop): ";
          cin » a » b;
          if (!a && !b)
               break;
          cout «a « " + " « b « " =   " « add(a.b)  « endl;
```



```
     )
    return 0;
}
```

Alternatively, you can solve this problem with recursion:

```
linclude <iostream.h>

unsigned int add( unsigned int lhs, unsigned int rhs )
{
     unsigned int carry = lhs & rhs;
     unsigned int result = lhs ^ rhs;

     if ( carry )
          return add( result, carry « 1 );
     el se
          return  result;
}

int  main()
{
     unsigned long a, b;
     for (;;)
     {
          cout « "Enter two numbers.  (0 0 to stop): ";
          cin » a » b;
          if (!a && !b)
               break;
          cout «a «•* + "« b « " =   " « add(a.b)  « endl;
     }
    return 0;
}
```