Prof.dr Milošević Borivoje

Objektno orijentisano programiranje

C++

2020
Beograd

OOP

?

OBJEKTNO ORIJENTISANO PROGRAMIRANJE (OOP)

"Object-oriented programing will be in the 1980's what structured programing was in the 1970's. Everyone will be in favor of it. Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programer will practice it (differently). And no one will know just what it is."

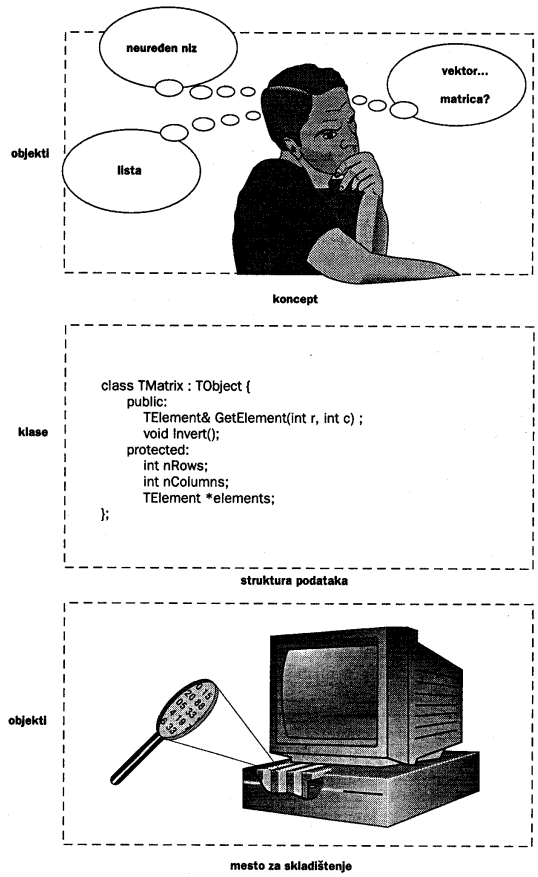
Tim Rentch, Computer Science Department, UCLA

"Objektno-orijentisano programiranje biće osamdesetih ono što je strukturno programiranje bilo sedamdesetih. Svako će biti za njega. Svaki proizvođač će promovisati svoje proizvode kao takve. Svaki menadžer će davati prazna obećanja u vezi sa njim. Svaki programer će ga praktikovati (na svoj način). I niko neće znati šta je to ustvari. "

Tim Rentch, Odsek kompjuterskih nauka, UCLA

OOP predstavlja nov način razmišljanja u oblasti razvoja softvera. U ovom poglavlju upoznaćete osnovne principe i veći deo terminologije OOP-a, preko jednostavnog primera programa. Cilj nije da proizvedemo savršen program za prikazivanje fajlova na ekranu (engl. *file viewer)* ili nešto sasvim drugo, već da radimo sa jednostavnim primerima koji ilustruju osnovne koncepte OOP.

Iako su u žiži poglavlja ideje na kojima se zasniva objektno orijentisano programiranje, ono sadrži i brojne fragmente C++ koda. Na ovom mestu u knjizi ne treba da brinete o svim detaljima C++ sintakse. Detalji će biti pokriveni u daljim poglavljima. Savladaćete sintaksu pošto ih pročitate. U ovom poglavlju dobićete opštu sliku, detalje ostavite po strani.

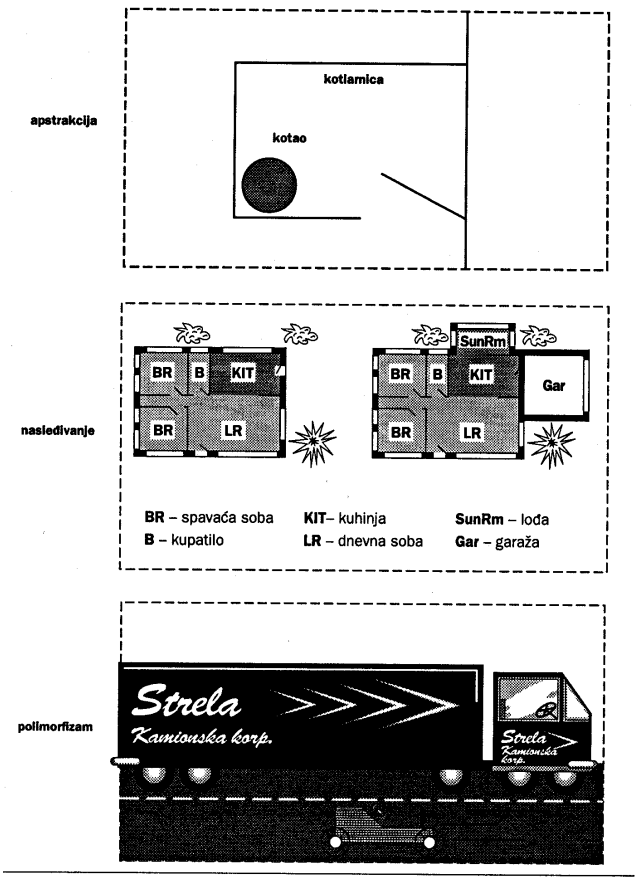


Pojava objektno-orijentisanih jezika i njihova sve šira primena možda je najvažniji događaj u razvoju programskih jezika od trenutka njihovog prerastanja u kategoriju viših programskih jezika. Radi se o konceptu koji se sa dosta misterije pojavljuje negde početkom osamdesetih godina. Danas je objektni koncept podrazumevan kod svih novih programskih jezika i predstavlja osnovu softverske industrije. U ovom poglavlju, biće najpre razmotrene osnovne karakteristike objektnih jezika, a zatim na primerima jezika C++ pokazano kako je ovaj koncept implementiran u ovom za nas najvažnijem objeketno - orijentisanom jeziku.

Klase, kao apstraktan pojam, su nacrti objekata. Reč "objekat" (engl. *object)* postala je marketinška čarolija, kao materinstvo i pita od jabuka.

Medutim, u ovoj knjizi marketinški slogani moraju da budu provereni na ulazu. Konfuzija oko "objekta" delimično nastaje i zbog činjenice što reč tehnički ima dva značenja. Prvo značenje pojma objekat je princip visokog nivoa, koncept koji nam omogućava da razmišljamo o organizovanju softvera. Termin "objektno orijentisano programiranje" govori o ovom prvom značenju.

Drugo značenje pojma objekat je određenije. Objekat je deo memorije u nekom izvršnom programu koji poštuje grupu korisnički definisanih pravila. U matematičkom programu koji ima matrice i nizove, svaka matrica i niz u memoriji bili bi jedan objekat. Konceptualno, projektant softvera smišlja koje su osobine neophodne jednom niz-objektu; za izvršni program, svaki niz kreiran u memoriji je objekat. Instanca (engl. *instance)* je drugi pojam kojim se nazivaju objekti u memoriji; nastao je od reči "aktivirati" (engl. *instantiate)* što označava tehniku oživljavanja objekta kreiranjem njegovog mesta u memoriji, da bi se zatim memorija inicijalizovala prema pravilima za dati objekat.

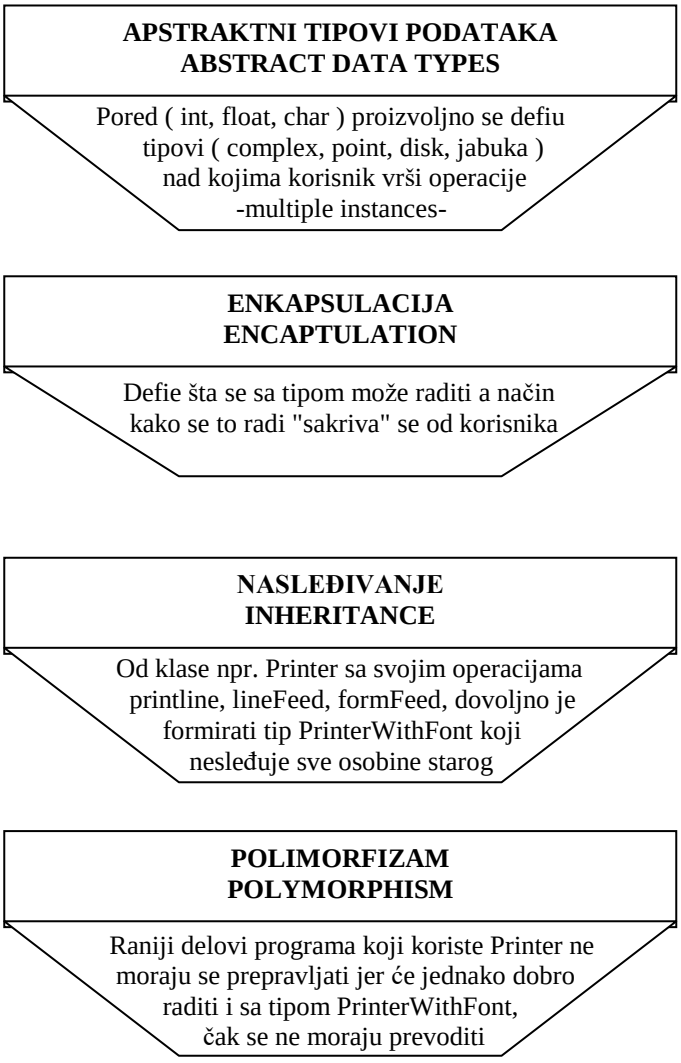


Dva prethodna pasusa opisuju objekte kao koncept i objekte kao mesto u memoriji. Treba da bude jasno da objekat kao koncept kreiramo razmišljanjem. Ali, kako kreiramo objekat kao mesto u memoriji? Ili, drugim rečima, koja osobina programskog jezika omogućava da opisujemo i gradimo objekte? Odgovor je *klasa* (engl. *class)* koja je u većićni objektno orijentisanih jezika, pa i u jeziku C++, korisnički definisan tip podataka za kreiranje objekta. Klase su kao nacrti - precizno opisuju kako objekat može da se izgradi i kako će se ponašati. Klasama je posvećeno celo poglavlje.

OSNOVNE KARAKTERISTIKE OBJEKTNIH JEZIKA

Može se reći da je u konvencionalnim, proceduralnim programskim jezicima (FORTRAN, PL/I npr.) akcenat na konceptima za definisanje algoritma. Program se defie tako što se opisuju koraci algoritma koji će se izvršavati nad podacima. Dalji razvoj programskih jezika išao je u pravcu jezika sa eksplicitnim opisima tipova podataka (Pascal), u okviru kojih se veća pažnja posvećuje predstavljanju podataka. Sledeći korak u tom pravcu razvoja jezika bili su jezici sa modulima kao osnovnim konceptom (Modula, Ada, C) koji se

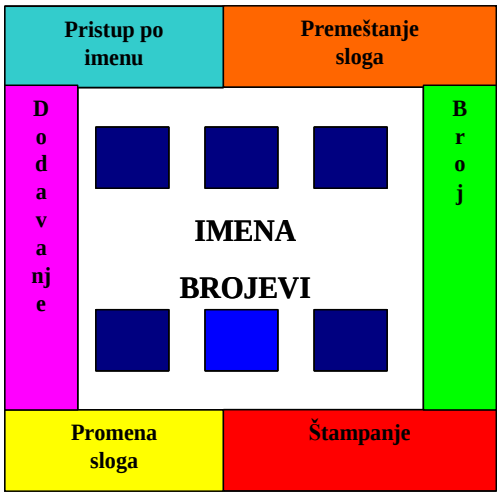
razvija do koncepta apstraktnih tipova podataka kao struktura koje inkapsuliraju podatke i procedure koje se mogu izvršavati nad njima. Objektno orijentisani jezici su nastali kao rezultat razvoja u tom pravcu. Tako možemo reći, da su nova rešenja koja nude OOPJ:



Apstraktni tipovi podataka - klase, objekti i poruke

Apstrakcija podataka predstavlja prvi korak ka objektno-orijentisanom programiranju. Ključni pojmovi u ovim jezicima su pojam objekta i pojam klase. Klasa je slična apstraktnom tipu podataka po tome što defie strukturu objekata i eksterni interfejs, funkcije i procedure (u terminologiji objektnih jezika metode) kojima se može delovati na objekte. Može se reći da klasa opisuje strukturu i ponašanje objekata. Objekti su pojedninačni primerci, instance klase. Svi objekti jedne klase imaju strukturu definisanu klasom i nad njima se mogu izvršavati samo operacije definisane klasom kojoj pripadaju. Na primer, na sledećoj slici je prikazan telefonski imenik kao objekat. Imena korisnika i

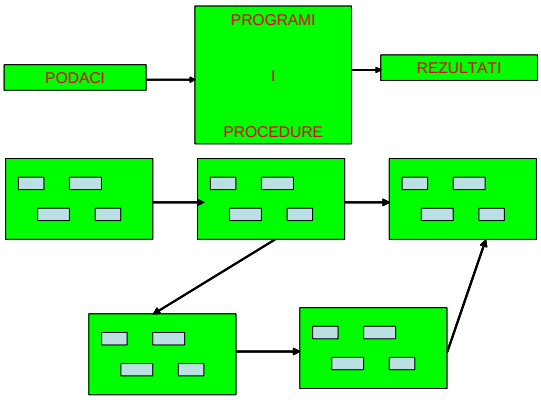
telefonski brojevi su podaci klase nad kojima se mogu izvršavati metodi klase kao što su: dodavanje novog broja, traženje po broju ili imenu i slično.



Donekle se može reći da se klasom opisuju objekti na sličan način na koji se tipovima podataka opisuje struktura podataka i skup operacija koje se nad njima mogu izvršavati. Klasa može da se posmatra kao šablon objekata *(template)* kojim se opisuje model po kome će se kreirati novi objekat. Klase se ponekad nazivaju fabrikama objekata, čime se naglašava algoritamska priroda kreiranja objekata.

Još jedan ključni pojam u terminologiji objektnih jezika su poruke. Kažemo da na objekte delujemo porukama, pri čemu svaka poruka predstavlja poziv metoda (funkcije ili procedure) definisanog u klasi kojoj objekat pripada. Objekat odgovara na poruku time što se izvršava odgovarajući metod - vidi sledeću sliku. Sintaksa poruka razlikuje se od jezika do jezika i najsličnija je pozivu procedura, stim što se obično ovaj poziv vezuje za objekat na koji se porukom deluje. Na primer, sledećom porukom u Smalltalk-u deluje se na objekat krug i defie njegov centar u tački (x,y) i r kao njegov poluprečnik:

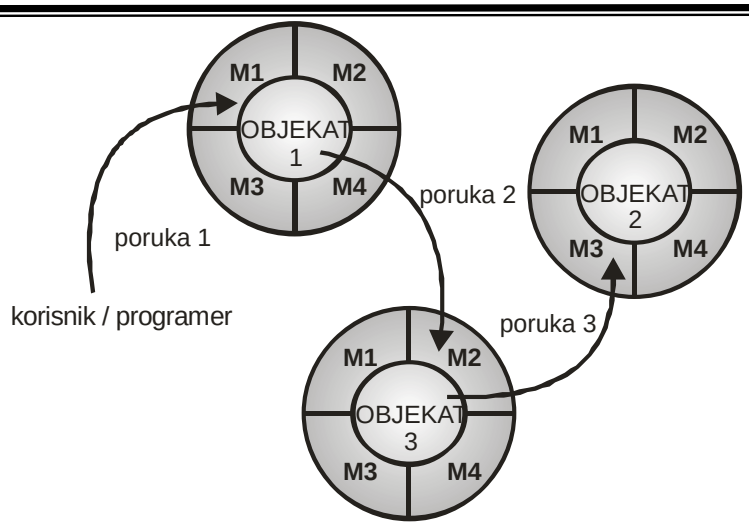
krug center: x @ Y radius:r:



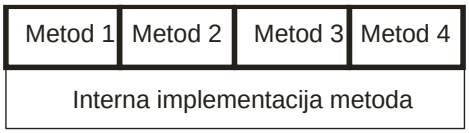
Koncepti klasa, objekat i poruka su dovoljni da shvatimo razliku između klasičnih (proceduralnih) programskih jezika i objektnih. Ova razlika je ilustrovana na sledećoj slici. Dok se kod proceduralnih jezika program može posmatrati kao skup potprograma (procedura i funkcija) kojima se prosleđuju podaci, kod objektnih jezika program se razvija kao skup objekata koji deluju jedni na druge slanjem poruka. Odgovor objekta na određenu poruku zavisi od klase u kojoj je definisan.

Klasa je slična apstraktnom tipu podataka po tome što defie strukturu i eksterni interfejs objekata. Postoji međutim suštinska razlika između klasa i tipova. Dok je klasa svojstvo objekta tip je svojstvo promenljivih i izraza.

U Pascal-u i Moduli 2 moramo upotrebiti opis val : Integer ; defiući na taj način promenljivu val tipa Integer. Ova informacija je važna kompilatoru, jer je preko tipa definisano koje su operacije dozvoljenje nad promenljivom.



Objekti šalju poruke jedan drugome



Korisnik klase izabira metod pomoću poruke dok klasa određuje stvarni sadržaj svojih metoda

Na jednom jednostavnom primeru pokazaćemo način pisanja programa u nekim objektnim jezicima zasnovanim na konceptu klasa:

Naš problem, za koji treba da napišemo objektni program, sastojaće se u tome da treba da nacrtamo krug sa krstom u sredini. Krug treba da ima poluprečnik r , a njegov centar treba da se pojavi na poziciji x@y. Da bi smo nacrtali traženi crtež potrebna su nam tri objekta:

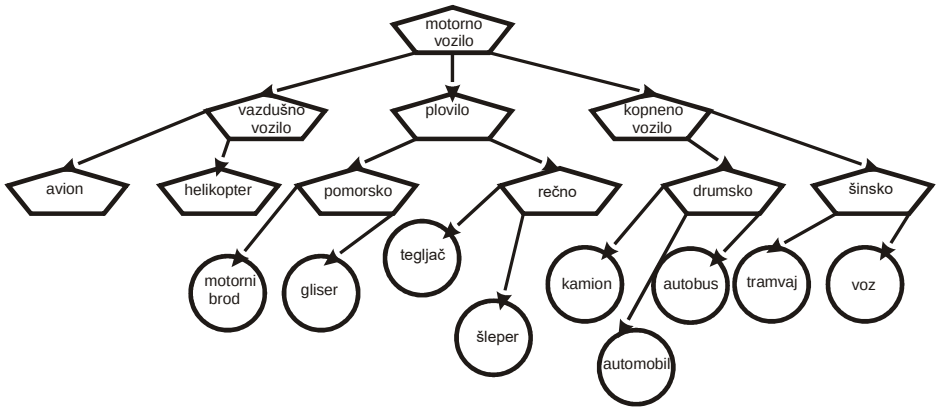
- krug: krug (pripada klasi Circle)
- vert : vertikalna linija (pripada klasi Line)
- hori : horizontalna linija (pripada klasi Line) ;

Očigledno je da su nam potrebne klase Circle i Line da bi kreirali objekat koji će biti prikazan. Za svaku klasu postoji elementaran konstruktor koji se može koristiti za kreiranje objekata sa nekom inicijalnom strukturom. Za inicijalizaciju novokreiranih objekata mogu se od strane programera definisati dodatni konstruktori sa isitim imenom ali sa različitim brojem i/ili tipovima argumenta. Sledeći program ilustruje kako se gore zadati problem rešava u C++ jeziku.

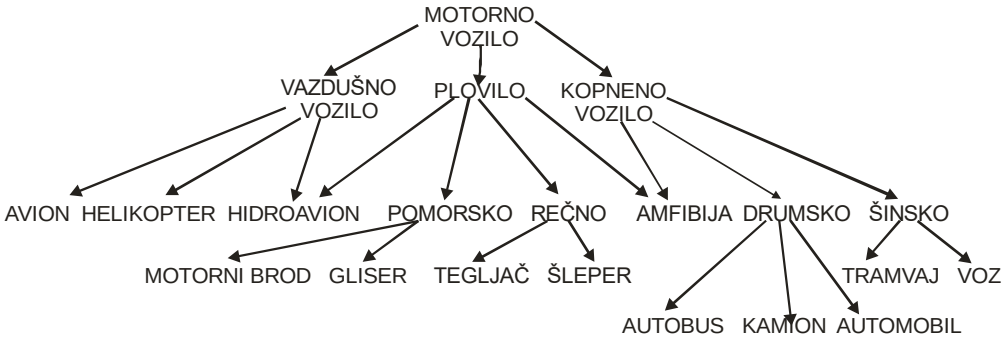
krug= new Circle (x, y, r)
vert=new Line(x, x-r, x, y+r)
hori=new Line(x-r, y, x+r, y)
krug-> display()
vert-> display()
hori-> display()

Klasa je dakle porodica objekata, a objekti stvoreni tokom izvršenja programa su primerci klase. Klasa može imati jedan ili više primeraka, a svaki primerak je pripadnik jedne klase (prosto nasleđivanje),

ili pripada većem broju klasa istovremeno:



Generičko stablo klase motornih vozila



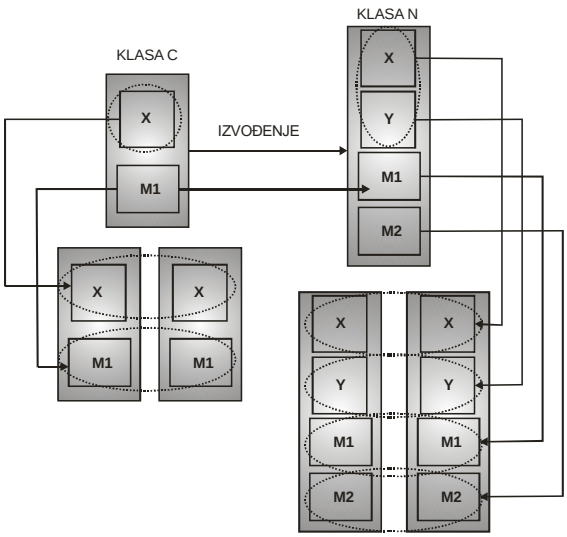
Stablo klase motornih vozila sa višestrukim nasleđivanjem.

Grupisanjem sličnih objekata stvara se klasa kao novi tip podataka koji sadrži zajednička svojstva objekata iz svog sastava.

Nasleđivanje

Kako smo već rekli klase se po strukturi ne razlikuju od apstraktnih tipova podataka i jasno je da one same po sebi ne donose neki novi kvalitet u odnosu na apstraktne tipove koji postoje i kod nekih jezika koji nisu objektni. Ono što bitno razlikuje klase od apstraktnih tipova je koncept nasleđivanja klasa. Nove klase se mogu definisati tako što naleđuju sva svojstva određene klase i tim svojstvima pridodaju nova. U okviru nove klase može da se izmeni struktura objekata koje klasa opisuje ali i njihovo ponašanje. Struktura objekata menja se tako što se nasleđuju svi podaci koje sadrži polazna klasa i njima pridodaju novi. Ponašanje objekata menja se time što se nasleđenim metodama klase pridodaju novi ali i promenom koda vezanog za postojeće metode, što je dodatno svojstvo objektnih jezika.

Radi boljeg razumevanja nasleđivanja razmotrirno primer prikazan na sledećoj slici. Klasa C opisuje objekte čija je struktura definisana promenljivom x i na koje se može



delovati metodom m1. Na levoj strani slike su prikazana dva objekta nastala kao primerci klase C. Nasleđivanjem klase C nastaje klasa N. Objekti koje opisuje klasa N imaju sve elemente objekata koje opisuje klasa C (nasleđuju sve elemente klase C) ali je njihova struktura definisana dodatnom promenljivom Y, i metodom m2. Objekti koji nastaju kao primerci klase N imaju tu izmenjenu strukturu u odnosu na objekte klase C. Objekti klase N mogu da odgovore na metode m1 i m2 dok objekti klase C samo na metode m1.

Nasleđivanje smanjuje vreme potrebno za razvoj programa, jer programer stalno i namerno pozajmljuje metode i osobine objekata koji postoje u sistemu. Dakle, jedan te isti kod se može više puta koristiti i deliti između raznih modula. Zbog nasleđivanja, klase su razvijene u hijerarhije koje predstavljaju stabla čiji su čvorovi klase, a nasleđivanje je nevidljivo vezivo između njih. Hijerarhija se stvara na vrlo prirodan način: jedna klasa nasleđuje drugu, koju opet može nasleđivati neka treća itd.

Primer:

Neka je klasa **Osoba** definisana kao tip objekata koji ima sledeće podatke:

((IME, char),(PREZIME, char),(DAT_RO\, date))

nad kojima se može izvršiti sledeći skup operacija (metoda):

(štampaj_ime, štampaj_prezime, štampaj dat_rođ, promeni_prez)

Jedan konkretan primerak ove klase bi bio:

((IME, Ana), (PREZIME, Kiš), (DAT_RO\, 250355))

Kada hoćemo da generišemo objekte koji se odnose samo na zaposlene osobe, onda kreiramo klasu **Zaposleni**, kao podklasu klase **Osoba,** koja ima sledeća obeležja:

((ODELJENJE, char), (RAD_MES, char), (PLATA, money),
(DAT_ZAP, date))

Nad kojima se može vršiti sledeći skup operacija (metoda):

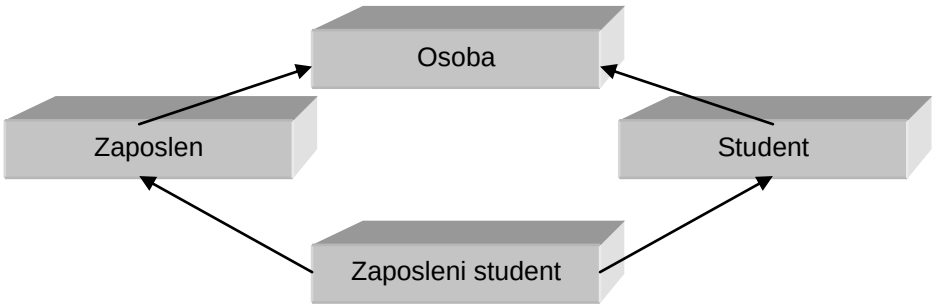
(štampaj_odeljenje, povećaj_platu, štampaj_dat_zap, promeni_odeljenje)

Svi elementi klase **Zaposleni** automatski će naslediti i sva obeležja : IME, PREZIME i DAT_RO\, a tipu objekata **Zaposleni** treba dodati obeležja: RAD_MES, ODELJENJE, PLATA i DAT_ZAP. Objekti klase **Zaposleni** nasleđuju i sve metode iz klase **Osoba** , a dodaju im se još neke **povećaj_platu, štampaj_platu, promeni_rad_mes, promeni_odeljenje** itd.

Jedan konkretan objekat klase **Zaposleni** bi bio:

((IME, Ana), (PREZIME, Kiš), (DAT_RO\, 250355), (ODELJENJE, AOP), (RAD_MES, Programer), (PLATA, 990), (DAT_ZAP, 290179))

Klasa može istovremeno naslediti svojstva više drugih klasa (višestruko nasleđivanje), a takođe, može naslediti više puta jednu te istu nadklasu (ponovljeno nasleđivanje).



Polimorfizam

Jedno od bitnih svojstava objektnih jezika je polimorfizam, koji se jednom rečenicom formuliše kao sposobnost različitih objekata da odgovore na istu poruku. Polimorfizam ćemo najlakše razumeti na primeru jedne heterogene strukture podataka, kolekcije koja se sastoji od objeketa različitog tipa npr. (a (i) i=1,2,..., n). Da bi smo prikazali te objekte koristimo recimo metod display koji se u jednoj petlji poziva za svaki od objekata iz kolekcije a(i):

a(i).display

Kako se kolekcija sastoji od različitih objekata moramo raspolagati sa mehanizmom koji će omogućiti da se u svakom konkretnom slučaju metod display "prilagodi" objektu koji treba da bude prikazan. Objektni jezici raspolažu ovim svojstvom i mehanizam funkcioe tako što se vezivanje metoda za objekat vrši u fazi izvršavanja programa, kada se u okruženju u kome je definisan objekat (klasa) traži metod koji treba da se izvrši. Svaki objekat sa sobom nosi informaciju o tome da li se određeni metod može i kako će se izvršiti nad njim. Očigledno je da ovaj koncept funkcioe samo ako se dozvoli povezivanje metoda i objekta tek u fazi izvršavanja programa. Zbog toga kažemo da polimorfizam zahteva dinamičku implementaciju jezika i takozvano kasno povezivanje, što se suštinski razlikuje od statičke implementacije i ranog povezivanja koje obično danas postoji kod jezika sa konceptom jakih tipova podataka. Naime, kod ovih jezika imamo da se već u fazi prevođenja (kompiliranja) programa koristi informacija o tipu i na osnovu nje vrši usklađivanje izraza i dodeljivanja po tipu, tako da se još u fazi prevođenja zna koji je izraz korektno napisan i koje dodeljivanje može da se izvrši. U terminologiji objektnih jezika rano povezivanje bi značilo da se informacija o pripadanju objekta određenoj klasi koristi još u fazi prevođenja objekta, a

praktično to znači da se već u tom trenutku zna da li se oddređeni metod može izvršiti ili ne nad određenim objektom. Ako se vratimo na naš primer, to bi značilo da bi objekti naše kolekcije morali da pripadaju jednom istom tipu podataka i da se nad njima može izvršavati samo jedan konkretan metod display, definisan klasom kojoj ti objekti pripadaju. Ovom analizom dolazimo do značajnog zaključka da "pravi" polimorfizam podrazumeva dinamičku implementaciju jezika i kasno povezivanje. Napomenimo da je to ujedno i najveći nedostatak samog koncepta jer kasno povezivanje obično slabi pouzdanost jezika i može da bude rizično za izvršavanje programa, posebno kod rada u realnom vremenu. Zbog svega toga danas u praksi najčešće srećemo objektne jezike sa statičkom implementacijom kod kojih se polimorfizam postiže kroz poseban, eksplicitno definisan koncept virtuelnih metoda za koje važi dinamička implementacija. Takav je slučaj sa programskim jezikom C++ o kome će biti reči u jednom od narednih odeljaka.

Zašto OOP?

Objektno orijentisano programiranje (engl. *Object Oriented Programming, OOP)* je odgovor na tzv. krizu softvera. OOP pruža načine za rešavanje nekih problema softverske proizvodnje.

Softverska kriza je posledica sledećih problema u proizvodnji softvera:

- Zahtevi korisnika su se *drastično* povećali. Za ovo su uglavnom "krivi" sami programeri: oni su korisnicima pokazali šta sve računari mogu, i da mogu mnogo više nego što korisnik može da zamisli. Kao odgovor, korisnici su počeli da traže mnogo više nego što su programeri mogli da postignu.

- Zato je bilo neophodno povećati produktivnost programera da bi se odgovorilo na zahteve korisnika. To je moguće ostvariti najpre povećanjem broja ljudi u timu. Tradicionalno proceduralno programiranje dopuštalo je projektovanje softvera u modulima sa relativno jakom interakcijom, a jaka interakcija između delova softvera koga pravi mnogo ljudi može da stvori probleme u razvoju softvera.

- Produktivnost se može povećati i tako što se neki delovi softvera, koji su ranije već negde korišćeni, mogu ponovo iskoristiti, bez mnogo ili imalo dorade. Tradicionalni način programiranja nije omogućavao laku ponovnu upotrebu koda (engl. *Software reuse*).

- Drastično su povećani i troškovi održavanja. Trebalo je naći način da projektovani softver bude čitljiviji i lakši za nadogradnju i modifikovanje. Primer: često se dešava da ispravljanje jedne greške u programu generiše mnogo novih problema, potrebno je lokalizovati realizaciju nekog dela tako da se promene u realizaciji ne šire po ostatku sistema.

- Proceduralno programiranje nije moglo da odgovori na ove probleme, pa je nastala kriza proizvodnje softvera. Zato je OOP došlo kao odgovor.

Šta daju OOP i **C++ kao odgovor?**

C++ je trenutno najpopularniji objektno orijentisani programski jezik. Osnovna rešenja koja pruža OOP , a C++ podržava, jesu:

l. ***Apstraktni tipovi podataka*** (engl. *abstract data types).* Kao što u C-u ili nekom drugom jeziku postoje ugrađeni tipovi podataka (int, float, char , ...), u jeziku C++ korisnik može proizvoljno definisati svoje tipove i potpuno ravnopravno ih koristiti (Complex, Point, Disk, Printer, Jabuka, BankovniRacun, Klijent itd.). Korisnik može kreirati proizvoljan broj primeraka tog tipa i vršiti operacije nad njima (engl. *multiple instances,* višestruke instance ili pojave).

2. ***Enkapsulacija*** (engl. *encapsulation).* Realizacija nekog tipa može (i treba) da se sakrije od ostatka sistema (od onih koji ga koriste). Korisnicima tipa treba precizno definisati samo *šta* se sa tipom može raditi, a način *kako* se to radi sakriva se od korisnika (defie se interno).

3. ***Nasledivanje*** (engl. *inheritance).* Pretpostavimo da je već formiran tip Printer koji ima operacije kao što su printLine, lineFeed, formFeed, gotoxy itd. i da je njegovim korišćšćenjem već realizovana velika količina softvera. Novost je da je firma nabavila i štampače koji imaju bogat skup stilova štampe i valja ih ubuduće iskoristiti. Nepotrebno je iz početka praviti novi tip štampača ili prepravljati stari kod. Dovoljno je kreirati novi tip PrinterWithFonts koji je "baš kao i običan" štampač, samo još može da menja stilove štampe. Novi tip će *naslediti sve* osobine starog, ali će moći da uradi još ponešto.

4. ***Polimorfizam*** (engl. *polymorphism).* Pošto je PrinterWithFonts već ionako Printer, i ostatak programa treba da ga posmatra kao i običan štampač, sve dok mu nisu potrebne nove mogućnosti štampanjča. Ranije napisani delovi programa koji koriste tip Printer *ne moraju se uopšte prepravljati,* oni će jednako dobro raditi i sa novim tipom. Pod određenim uslovima, stari delovi ne moraju se čak ni ponovo prevoditi! Osobina novog tipa da se "odaziva" na pravi način, iako ga je korisnik "prozvao" kao da je stari tip, naziva se *polimorfizam.*

Sve navedene osobine mogu se, na ovaj ili onaj način, pojedinačno realizovati i u proceduralnom jeziku (kakav je i C), ali je realizacija svih koncepata ili teška, ili sasvim nemoguća. U svakom slučaju, realizacija nekog od ovih principa u proceduralnom jeziku drastično povećava troškove i smanjuje čitljivost programa.
Jezik C++ podržažava sve navedene koncepte, oni su ugrađeni u sam jezik.

Šta se menja uvođenjem OOP-a?

Jezik C++ nije "čisti" objektno orijentisani programski jezik (engl. *Object-Oriented Programming Language, OOPL)* koji bi korisnika "naterao" da ga koristi na objektno orijentisani (OO) način. C++ može da se koristi i kao"malo bolji C", ali se time ta ne dobija (čak se i gubi). C++ treba koristiti kao sredstvo za OOP i kao smernicu za razmišljanje. C++ ne sprečava pisanje lakih programa, već samo omogućava pisanje mnogo boljih programa.

OOP uvodi *drugačiji način razmišljanja* u programiranje!

- U OOP-u, *mnogo* više vremena troši se na *projektovanje,* a mnogo manje na samu implementaciju (kodovanje).

- U OOP-u, razmišlja se najpre o *problemu,* a tek naknadno o programskom rešenju.

- U OOP-u, razmišlja se o delovima sistema (objektima) koji nešto rade, a ne o tome kako se nešto radi (algoritmima). Drugim rečima, OOP prvenstveno koristi *objektnu dekompoziciju* umesto isključivo *algoritamske dekompozicije.*

- U OOP-u, pažnja se prebacuje sa realizacije na medusobne veze između delova. Teži se što većoj redukciji i strogoj kontroli tih veza. Cilj OOP-a je da smanji interakciju između softverskih delova.

1. Uvod

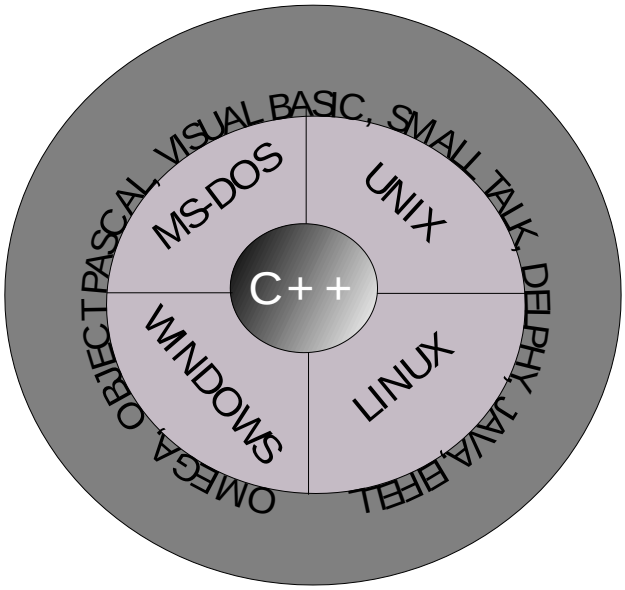
1.1 O programskom jeziku *C++*

Programski jezik C je jezik opšte namene, srednjeg nivoa, koji omogućava dosta intiman kontakt sa hardverom računara. Poseduje strukturirane tipove podataka i upravljačke strukture što je karakteristika viših programskih jezika. S druge strane, podržava manipulaciju sa bitovima, korišćenje procesorskih registara, pristup podacima pomoću adrese i operatore orijentisane ka hardveru računara. Ovo su karakteristike nižih programskih jezika kao što su simbolički mašinski jezici. Sintaksa jezika omogućava koncizno izražavanje i pisanje strukturiranih programa.

Programski jezik C je projektovao *Dennis Ritchie* 1972. godine u *Bell-ovim* laboratorijama. Osnovni cilj je bio sastavljanje jezika nezavisnog od računara, sa karakteristikarna viših programskih jezika, koji će moći da zameni simboličke mašinske *(assemblerske) jezike koji su, i te kako, zavisni od računara.*

Dugi niz godina osnovna definicija jezika C bio je referentni priručnik *(Reference Manual)* u sastavu prvog izdanja knjige *The* C *Programming Language* čiji su autori *Brian W. Kernighan* i *Dennis M. Ritchie.* Varijanta jezika C koja je opisana u toj knjizi danas se naziva *Klasični* C. Zvanični standard za jezik *C,* takozvani *ANSI* C, izdao je Američki nacionalni institut za standarde *(American National Stalldards Institute)* 1989. godine pod brojem X3.159-1989.

Pojavom novih tehnika u programiranju, prvenstveno pojavom objektno orijentisanog programiranja osetila se potreba za novim mogućnostima jezika C. Tako su 1980. godine dodate klase, provera i konverzije tipova argumenata prilikom pozivanja funkcija i još neke druge novine. Tako dobijeni jezik nazivao se C *sa klasama.*

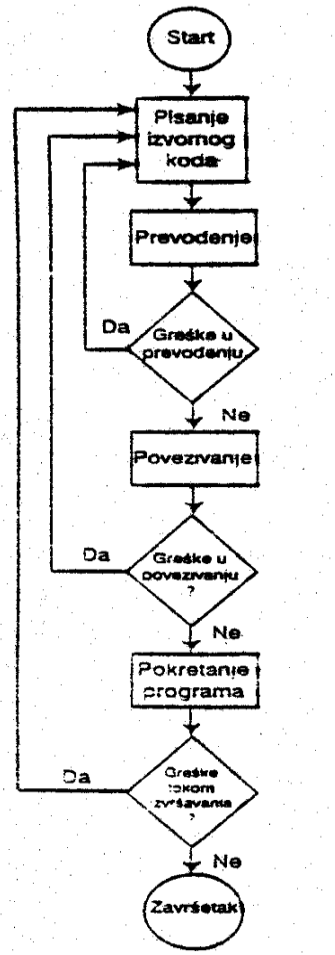


Glavnu novinu predstavlja mogućnost definisanja novih tipova podataka u pravom smislu te reči. Dok strukture (struct) u jeziku C defiu samo moguće vrednosti za definisane podatke, novouvedene klase (class) defiu i moguće operacije nad tim podacima. Šta više, jedine operacije nad podacima date klase mogu da budu samo one koje su predviđene definicijom te klase.

Posle daljeg proširivanja, na prvom mestu dodavanjem virtuelnih funkcija i preklapanja operatora, jezik je 1983/84. godine konačno dobio današnje ime *C++*. lme treba

da sugeriše da se ne radi o novom jeziku, već o proširivanju jezika C . C ++ je u jeziku C operator povećavanja ! Preko 95% jezika C usvojeno je bez izmena i u jeziku *C++*. Promenjeni su samo detalji koji su morali da budu promenjeni radi obezbeđivanja konzistentnosti novih koncepcija u jeziku *C++*. Te izmene se odnose na vrlo suptilne detalje koji dolaze do izražaja samo kod krajnje profesionalnog programiranja. U nedostatku zvaničnog standarda, kao osnovna definicija jezika, u početku, koristila se knjiga The *C++ Programming Language* čiji je autor *Bjarne Stroustrup.* On se smatra i autorom samog jezika *C ++* . Kasnije su dodate nove mogućnosti kao što su višestruko nasleđivanje, apstraktne klase, mehanizmi za sastavljanje generičkih klasa i za rukovanje izuzecima (obradu grešaka). Osnovnu definiciju jezika *C++* sa početka 1991. godine predstavlja knjiga *The Annotated C++ Reference Manual* čiji su autori *Margaret A. Ellis* i *Bjarne Stroustrup.* To je istovremeno bio i jedan od osnovnih dokumenata grupe za izradu *ANSI* standarda za jezik *C ++* .

ANSI stnadard za jezik *C++* usvojen je l4. II. 1997. godine. Jezik *C ++* danas je jedan od najmoćnijih jezika za objektno orijentisano programiranje. Operativni *sistem UNIX* predstavlja prirodno okruženje za jezik *C++*, kao što je to slučaj i sa jezikom C. Ali, s obzirom da se radi o jeziku opšte namene, nudi se i pod drugim operativnim sistemima kao što je *MS-DOS* i *MS-Windows* na danas vrlo populamirn ličnim računarima *tipa IBM-PC.* Akreditovani komitet za standard, koji radi pod procedurama Američkog instituta za standarde priprema međunarodni standard za *C++*, čiji nacrt je publikovan i dostupan na Internetu, na adresi www.libertyassociates.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 proizvođača. 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 proizvodači ga podržavaju. Učinili smo maksimalne napore da budemo sigurni da sav programski kod u ovom izdanju knjige odgovara ANSI standardu.



1.2 Obrada programa na jeziku *C++*

Obrada programa sastoji se od sledeća četiri koraka:

- unošenje izvornog teksta programa u datoteku na disku,
- prevođenje izvornog teksta programa,
- povezivanje prevedenog oblika programa sa potrebnim korisničkim i sistemskim potprogramima u izvodljivi oblik, i
- izvršavanje programa.

U narednim tačkama prikazano je kako se gornji koraci izvode u dva najčešća oktuženja u kojima se koristi programski jezik *C++*. Prvo je operativni sistem *UNIX* koji predstavlja prirodno okruženje za jezik *C++*. Drugo često okruženje u kome se koristi jezik

C++ su operativni sistemi MS-DOS i Windows na ličnim računarima tipa *IBM-PC* i njima sličnim računarima.

1.2.1 Rad pod operativnim sistemom *MS-DOS* i prevodiocem *Borland C++*

Prilikom rada na ličnim računarima tipa *lBM-PC* pod operativnim sistemom *MS-DOS* postoji veći izbor mogućnosti nego pod operativnim sisternom *UNIX*. Pored raznih urednika teksta na raspolaganju su i nekoliko prevodilaca za jezik *C++*. Svakog od tih prevodilaca prati i odgovarajuće okruženje u kome radi. U današnje vreme kada se uglavnom koristi operativni sistem *MS- Windows* ovde opisani način rada može da se sprovodi u *DOS* prozoru.

Za prikazivanje odabran je rad sa prevodiocem *Borland C++* iz razloga što taj prevodilac radi u otvorenom okruženju operativnog sistema *MS-DOS*, što se sa njim radi vrlo slično kao i pod operativnim sistemom *UNIX* i što se sa njim lako obrađuju veliki programski sistemi čiji se izvorni tekst nalazi u više datoteka. Za potrebe većeg dela ove knjige zadovoljava bilo koja verzija prevodioca *Borland C++* počev od verzije *3.0*. Jedino za rad sa izuzecima je neophdna verzija *4.0* ili neka novija.

Za sastavljanje izvornog teksta programa može da se koristi bilo koji urednik teksta. To može da bude standardni urednik teksta EDIT koji se nudi pod operativnim sistemom *MS- DOS*, ali može da se koiisti i NOTEPAD operativnog sisterna *Windows*. U slučaju rada na jeziku *C++*, ime datoteke mora da ima proširenje .CPP. Komanda upravljačkog jezika operativnog sistema *MS-DOS* za unošenje i obradu izvornog teksta prograrna u datoteku *imeprog* .CPP je:

EDIT imeprog.CPP

Po završetku sastavljanja izvornog teksta programa komandom: BCC *imeprog* vrši se prevođenje izvorng teksta prograrna na jezik-u *C++* iz datoteke *imeprog.CPP*. Rezultat prevođenja smešta se u datoteku imeprog .OBJ i odmah se stvara i povezani oblik programa koji se smešta u datoteku sa imenom *imeprog* .EXE.

U slučaju sastavljanja velikog programskog sistema vrlo često se izvomi tekst g1avnog programa i svih potrebnih potprograma smešta u više datoteka. U ovom slučaju potrebno je da se preskoči povezivanje prilikom prevodenja sadržaja datoteka u kojima se nalaze samo potprogrami. To se postiže dodavanjem opcije */C* u komandi BCC: BCC */C imeprog*. U komandi BCC može da bude naveden i čitav niz imena datoteka, od kojih neke mogu da sadrže izvorne tekstove, a neke prevedene oblike programskih modula. Pomoću komande: BCC *ime1 ime2.0BJ imeN.CPP* prevodiće se sadržaji datoteka čija imena nemaju proširenje ili imaju proširenje .CPP, i rezultati prevođenja smeštaće se u odgovarajuće datoteke sa proširenjem imena .OBJ.

Posle toga povezaće se programski moduli iz svih novostvorenih datoteka kao i iz datoteka čija imena u grnjoj komand imaju proširenja .OBJ. Povezani oblik celokupnog programskog paketa smešta se u datoteku sa imenom *ime1.EXE*. Na kraju, izvršavanje programa postiže se komandom koja se sastoji samo od imena datoteke, bez proširenja .EXE, koja sadrži povezani oblik programa. Komanda u slučaju datoteke *imeprog.EXE je:* - *imeprog*

Posle gornje komande počinje izvršavanje korisnikovog programa. U toku rada korisnikovog programa korisnik treba da preko tastature unosi podatke po redosledu kako ih očekuje program sa "glavnog ulaza" računara. Na ekranu pojaviće se podaci koji se unose preko tastature kao i podaci koje program piše na "glavni izlaz" računara. Pod operativnim sisternom *MS-DOS* postoji vrlo elegantan način za "skretanje" glavnog ulaza i/ili glavnog izlaza. Time se postiže da se podaci čitaju iz neke datoteke umesto sa tastature,

odnosno da se upisuju u neku datoteku umesto ispisivanja na ekranu. To se postiže izvršavanjem programa pomoću komande oblika: *imeprog* <*podaci* >rezult. Znak < ispred imena označava da se čitanje sa glavnog ulaza vrši iz te datoteke, (strelica koja "izlazi" iz imena datoteke), a znak > ispred imena da se pisanje na glavni izlaz vrši u tu datoteku (strelica koja "ulazi" u ime datoteke). Datoteke *podaci* i *rezult* su tekstualne datoteke, što znači da mogu da se obrađuju urednikom teksta ili drugim programima za obradu teksta.

U prethodnom izlaganju je pretpostavljeno da je obezbeđeno neophodno okruženje za pristup do svih potrebnih programa (EDIT, BCC i TLINK) kao i do potrebnih potprogramskih biblioteka (CS .LIB i MATHS .LIB). Operativni sistem *MS-DOS* ne pravi razliku između malih i velikih slova. Zbog toga, mala i velika slova u gornjim komandama mogu da se koriste na potpuno proizvoljan način. To važi iza imena datoteka, pa ALFA.CPP i aLfa.cPp predstavljaju istu datoteku koja sadrži tekst izvornog programa najeziku *C++*.

Na kraju ovog izlaganja neophodno je da se napomene da firma *Borland* pored gore pomenutog prevodioca za jezik *C++* nudi i takozvano "integrisano okruženje" za razvoj programa na jeziku *C++* koje se pokreće komandom BC ili WBC. To okruženje u sebi sadrži urednik teksta, prevodilac, alate za otkrivanje grešaka, alat za rukovanje razvojem velikih programskih sistema i još neke druge alate koji čine rad udobnijim od gore opisanog rada u otvorenom okruženju. Iskustvo, međutim pokazuje da dobijeni prevedeni oblici (.OBJ) i povezani oblici (.EXE) obrađivanih programa zauzimaju znatno više prostora kada se radi u integrisanom okruženju nego kada se radi sa prevodiocem u otvorenom okruženju. Integrisano okruženje firme *Borland* počev od verzije *4.0* je u obliku aplikacije za operativni sistem *Windows.* Komunikacija sa korisnikom se vrši pomoću grafičke sprege. Podržava razvoj programa kako za *MS-DOS,* tako i za *Windows.* Naravno, što je neki alat moćniji, to je on glomazniji i složeniji za korišćenje. Međutim, u ovom slučaju potrebno je voditi računa jer se sada velika i mala slova razlikuju i različito prevode.

1.3. Šta daju OOP i C++ kao odgovor?

C++ je trenutno najpopularniji objektno orijentisani programski jezik. Osnovna rešenja koja pruža OOP , a C++ podržava, jesu:

1. Apstraktni tipovi podataka (engl. *abstract data types*). Kao što u C-u ili nekom drugom jeziku postoje ugrađeni tipovi podataka (int, float, char, ...), u jeziku C++ korisnik može proizvoljno definisati svoje tipove i potpuno ravnopravno ih koristiti (Complex, Point, Disk, Printer, Jabuka, BankovniRacun, Klijent itd.). Korisnik može kreirati proizvoljan broj primeraka tog tipa i vršiti operacije nad njima (engl. *multiple instances*, višestruke instance ili pojave).

2. Enkapsulacija (engl. *encapsulation).* Realizacija nekog tipa može (i treba) da se sakrije od ostatka sistema (od onih koji ga koriste). Korisnicima tipa treba precizno definisati samo šta se sa tipom moze raditi, a način kako se to radl sakriva se od korisnika (defie se interno).

3. Nasledivanje (engl. *inheritance).* Pretpostavimo da je već formiran tip Printer koji ima operacije kao što su printLine, lineFeed, formFeed, gotoxy itd., i da je njegovim korišćenjem već realizovana velika količina softvera. Novost je da je firma nabavila i štampače koji imaju bogat skup stilova štampe i valja ih ubuduće iskoristiti. Nepotrebno je iz početka praviti novi tip štampača ili prepravljati stari kod. Dovoljno je kreirati novi tip PrinterWithFonts koji je "baš kao i običan" štampač, samo "još može da" menja stilove štampe. Novi tip će *naslediti sve* osobine starog, ali će i moći da uradi još ponešto.

4. Polimorfizam (engl. *polymorphism).* Pošto je PrinterWithFonts već ionako Printer , i ostatak programa treba da ga posmatra kao i običan štampač, sve dok mu nisu potrebne nove mogućnosti štampača. Ranije napisani delovi programa koji koiste tip Printer *ne moraju se uopšte prepravljati,* oni će jednako dobro raditi i sa novim tipom. Pod odredenim uslovima, stari delovi ne moraju se čak ni ponovo prevoditi! Osobina novog tipa da se "odaziva" na pravi način, iako ga je korisnik "prozvao" kao da je stari tip, naziva se *polimorfizam.*

Sve navedene osobine mogu se, na ovaj ili onaj način, pojedinačno realizovati i u proce- duralnom jeziku (kakav je i C), ali je realizacija svih koncepata ili teška, ili sasvim nemoguća. U svakom slučaju, realizacija nekog od ovih principa u proceduralnom jeziku drastično povećava troškove i smanjuje čitljivost programa. Jezik C++ podržava sve navedene koncepte, oni su ugradeni u sam jezik.

2. Ulaz – Izlaz

Ulaz i izlaz podataka treba da omogućava komunikaciju programa sa spoljnim svetom radi unošenja podataka za obradu i prikazivanje ili odlaganje rezultata. Zbog toga ne može da se zamisli program bez ulaza i izlaza podataka. Uprkos tome, ulaz i izlaz podataka nije deo jezika *C++* u tom smislu što ne postoje zasebne naredbe za izvodenje tip radnji. Umesto toga postoje odgovarajuće bibliotečke klase za njihovo ostvarivanje. Pošto su datoteke u jeziku *C++,* kao što je slučaj i u jeziku C, samo dugački nizovi bajtova, nazivaju se "tokovima". Nema suštinske razlike između toka na disku (datoteke) i toka u operativnoj memoriji.

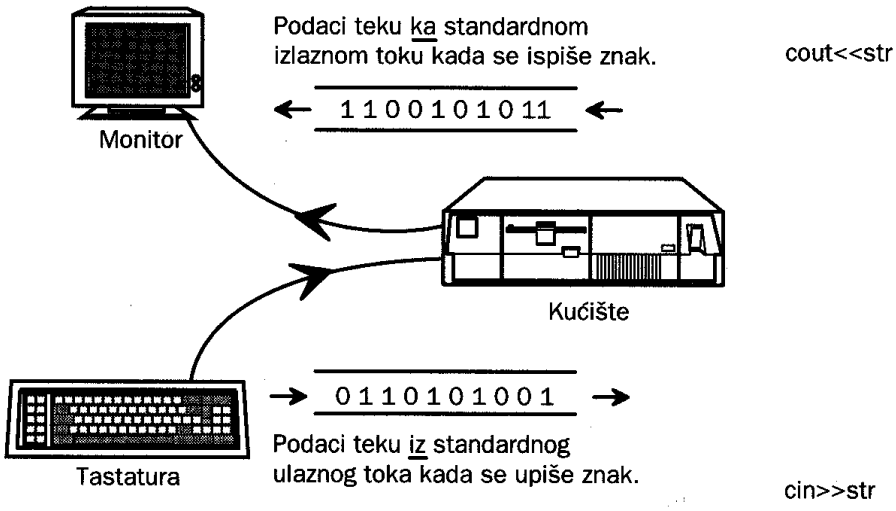
Većina radnji sa tokovima su istovetne, bez obzira gde su oni smešteni. Rad sa tokovima u jeziku *C++* realizuje se odgovarajućim klasama. Konkretni tokovi predstavljaju se pomoću primeraka tih klasa. Za rad sa datotekama (tokovima na magnetnim diskovima) postoje tri klase opisane u zaglavlju fstream.h. Klasa ofstream predviđena je samo za izlazne datoteke, klasa ifstream samo za ulazne datoteke, dok klasa fstream za ulazno-izlazne datoteke. Za rad sa tokovima u operativnoj memoriji, takođe postoje tri klase. Opisane su u zaglavlju strstream.h. Klasa ostrstream služi samo za smeštanje podataka u tokove, klasa istrstream samo za uzimanje podatka iz tokova, dok klasa strstream omogućava kako uzimanje tako i smeštanje podatka u tokove. Sve gore pomenute klase izvedene su iz odgovarajućih klasa ostream i istream koje ostvaruju efektivni prenos podatka sa ili bez ulazno-izlazne konverzije. Ove klase su zajedničke kako za tokove na diskovirna tako i za tokove u operativnoj memoriji. Postoje četiri standardna toka (primeraka klasa ostream ili istream) koji se automatski stvaraju na početku izvršavanja svakog programa:

cin -glavni (standradni) ulaz tipa istream. Predstavlja tastaturu dok se ne izvrši skretanje glavnog ulaza unutar samog programa ili u komandi operativnog sistema za izvršavanje programa.

cout -glavni (standardni) izlaz tipa ostream. Predstavlja ekran dok se ne izvrši skretanje glavnog izlaza unutar samog programa ili u komandi operativnog sistema za izvršavanje programa. Koristi se za ispisivanje podataka koji čine rezultate izvršavanog programa.

cerr - (standardni) izlaz za zabeleške tipa ostream. Predstavlja ekran dok se ne izvrši skretanje izlaza za poruke unutar samog programa. Koristi se obično za ispisivanje poruka o greškama.

clog -(standardni) izlaz za zabeleške tipa cstream. Predstavlja ekran dok se ne izvrši skretanje izlaza za zabeleške unutar samog programa. Koristi se obično za vođenje dnevnika o događajima za vreme izvršavanja programa.



2.1. Upoznavanje sa tokovima :

U ovom delu će se termin *tok* (engl. *stream)* često koristiti, uostalom kao i u sistemskom programiranju uopšte. Šta je tok?
U svakodnevnom životu, tok je na primer protok vode. U programiranju, tok je protok podataka. Glavna odlika tog toka je da teče u jednom smeru. Možda ćete reći da to i nije sasvim tačno (jer i najmanji tok ima vrtloge), ali olakšava razumevanje tokova. Ulazno-izlazni tok je sekvenca bajtova koji neprekidno teku u jednom ili drugom smeru, u zavisnosti od toga da li se radi o ulazu ili izlazu.

Druga odlika toka je neiscrpnost. Tokovi ponekad presuše, ali se to ne dešava često. Izlazni tok će, na primer , uvek prihvatiti još jedan bajt, osim u slučaju prepunjenog diska.

Dva najčešćšće korišćšćena toka u programiranju jesu strandardni ulazni i izlazni tokovi, a oni ne presušuju. Od standardnog ulaznog toka uvek možete tražiti da prihvati još jedan ASCII znak sa tastature i uvek možete ispisati još jedan znak na ekranu.

2.1.1. Letimičan pogled na operatore toka << i >>

Operatori toka << i >> alternativa su funkcijama printf, scanf i ostalima iz datoteke stdio.h. Ovi operatori imaju dve prednosti: prvo, ne morate da koristite oznake formata ako ste zadovoljni podrazumevanim formatom; i drugo, značenje operatora možete proširiti tako da rade i s vašim klasama.

NAPOMENA: U ovom poglavlju naučićete nekoliko ulazno-izlaznih tehnika: printf i scanf, operatore toka i ulaz zasnovan na redu. Standardna biblioteka C++-a koristi različite ulazno-izlazne bafere za svaku tehniku, a njihovo kombinovanje može dovesti do nepredvidivih rezultata. Ako ipak morate da ih kombinujete, radite to pažljivo i imajte na umu da u ulazno-izlaznim klasama C++-a postoji funkcija sync_with_stdio koja služi za koordiniranje funkcija printf i scanf sa ulazno-izlaznim podacima.

Evo jednostavnog programa, koji čita dva broja u formatu pokretnog zareza i ispisuje njihov zbir:

```
#include <iostream.h>
void main()
{
double a,b;
```

```
cout << "Unesite prvi broj: ";
cin >> a;
cout << "Unesite drugi broj: ";
cin >> b;
cout << "Zbir je ";
cout << a + b << endl ;
}
```

Naredni program postupa isto, samo što koristi funkcije printf i scanf:

```
#include <stdio.h>
void main()
{
double a,b;
printf("Unesite prvi broj: ");
scanf("%lf", &a);
printf("Unesite drugi broj: ");
scanf("%lf", &b);
printf("Zbir je %lf\n", a + b);
}
```

Uočite sledeće važne tačke u verziji koja koristi operatore toka cin i cout:

- .Koristi se datoteka zaglavlja iostream.h, a ne datoteka stdio.h.
- Nisu potrebne oznake formata; način prenosa određuje tip objekta (u ovom primeru a i b). U tom pogledu, operatori toka su nalik iskazu PRINT u BASIC-u i malo se lakše koriste nego funkcije printf i scanf.
- Operator adrese (&) ne primenjuje se na operande kad se koristi cin, kao što je slučaj kod funkcije scanf. (Tu ulazni tokovi više liče na BASIC, jer koriste reference argumenata.)
- Podaci teku ka standardnom izlaznom toku (cout), što je obično ekran:

  cout << "Unesite prvi broj: ";
- Podaci teku iz standardnog ulaznog toka (cin) , što je obično tastatura:

  cin >> a;
- U objekte toka se znak za kraj reda upisuje pomoću endl , dok se u C-u koristi izlazna sekvenca /n (što može da se koristi i u C++-u).

U predhodnom primeru, poslednja dva reda :

```
cout << "Zbir je ";
cout << a + b ;
```

mogu se napisati i zajedno:

```
cout << "Zbir je " << a + b ;
```

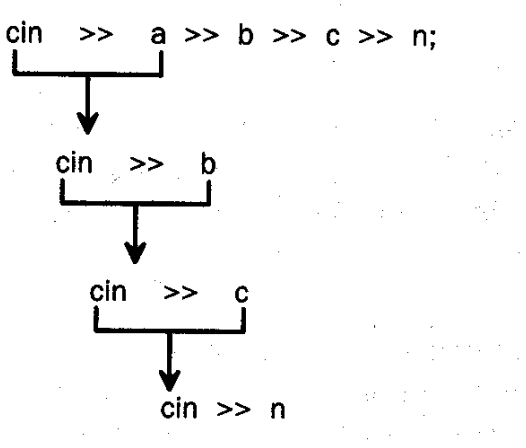
Operatori pomeranja su asocijativni sleva nadesno, što znači da će prvo biti izračunat sledeći izraz: cout << "Zbir je "

Kao što je u jezicima C i C++ uobičajeno, ovaj izraz radi dve stvari: šalje znakovni niz toku cout (što bi se moglo nazvati sporednim efektom), a zatim se izračunava i vraća vrednost. Vrednost izraza u obliku cout << argument jeste sam tok cout. Zbog toga se cout << " Zbir je " zamenjuje tokom cout. Zbog toga se izraz izračunava na sledeći način:

cout <<« "Zbir je " <<« a + b;
(cout <<« "Zbir je ") « << a + b; //ispiši znakovni niz.
cout «<< a + b; // ispiši a + b

To je trik u C++-u koji omogućava da u velikom, složenom izrazu uvek iznova koristite objekat cout. U ovom slučaju se prvo ispisuje znakovni niz, a zatim a + b. Ovaj trik možete upotrebiti i za čitanje nekoliko brojeva u istom iskazu, jer svaki izraz oblika cin >> argument daje rezultat cin:

cin >> a >> b >> c >> n;



U narednim odeljcima prikazane su važnije metode za rad sa tokovima iz spomenutih klasa. Pregled nije potpun, ali je sasvim dovoljan za rešavanje čak i relativno složenih problema pri radu sa tokovima. Treba još napomenuti da funkcije za ulaz i izlaz korišćene u jeziku C, koje su opisane u zaglavlju stdio.h, i dalje stoje na raspolaganju i mogu i dalje da se koriste. Klase za ulaz i izlaz koje se nude u jeziku C++ međutim, omogućavaju efikasnije izvođenje ulazno-izlaznih operacija, pa se preporučuje da se one koriste. U svakom slučaju, nikako se ne preporučuje da se u nekom programu istovremeno koriste obe vrste ulazno-izlaznih funkcija.

Upotreba operatora za ulaz - izlaz biće opisana u nastavku i po svojoj sintaksi i po semantici jezika C++.

2.2. Operator za ulaz - cin >> (dvostruko veće - prevodi se kao jedan znak)

sintaksa je:cin >> <promenljiva>

gde je cin objekat (promenljiva) klase ili tipa istream, unet sa tastature

semantika je: ***cin >> <promenljiva>;***
cin >> <promenljiva> {>><promenljiva>};
cin >> promenljiva$_1$>>promenljiva$_2$>>...>>promenljiva$_N$;

Da bi podatke uneli sa tastature moramo ih ukucati i preneti u operativnu memoriju pritiskanjem tastera ENTER kao što sledi na primeru:

```
double a,b,c;
cin >> a>>b>>c;
```

može biti uneto na razne načine pretpostavljajući da je a=1.1,b=2.2,c=3.3 :

1.1 2.2 3.3 ENTER	1.1 2.2 ENTER 3.3 ENTER	1.1 ENTER 2.2 3.3 ENTER	1.1 ENTER 2.2 ENTER

2.3. Operator za izlaz - cout << (dvostruko manje - prevodi se kao jedan znak)

sintaksa je:***cout*** **<<** **<*izraz*>**

gde je cout objekat (promenljiva) klase ili tipa ostream, unet sa tastature

semantika je: ***cout*<< <*izraz*>;**
***cout*<< <*izraz*> *š*>><*izraz*>*ć*;**
***cout*<< $izraz_1$<<$izraz_2$<<...<<$izraz_N$;**

Jedna od prednosti upotrebe operatora cin i cout, posebno za početnike, jeste u tome što se ne koriste neobični simboli neophodni za funkcije printf i scanf. Operator prepoznaje tip podataka, a za svaki tip podataka vezano je podrazumevano ponašanje. Primera radi, sledeći iskaz standardno ispisuje decimalni prikaz za n, što je uobičajen format.
cout << n;

Šta biva ako broj želite da ispišete u heksadecimalnom ili oktalnom formatu? Funkcijom printf možete da uradite sledeće:

```
int n = 16;
printf("n je %x heksadecimalno, %o oktalno, i %d dekadno.\n", n, n, n) ;
```

Rezultat ovoga primera je:
n je 10 heksadecimalno, 20 oktalno i 16 dekadno.

Sledeći primer koristi operatore toka i ispisuje isti rezultat:

```
int n = 16;
cout << "n  je " << hex << n <<" heksadecimalno, ";
cout << oct << n << " oktalno, i ";
cout << dec << n << " dekadno."  << endl ;
```

hex, oct i dec su objekti posebne vrste (o njima ćete više detalja saznati u daljim poglavljima). Kada se ispisuju u tok cout ili čitaju iz toka cin, oni menjaju format narednog celog broja koji se učita iz toka ili upiše u njega.

3. Delovi C++ programa

C++ program se satoji od objekata, funkcija, promenljivih, konstanti i drugih delova. Ako posmatramo, računajući na to da već imamo dovoljno iskustva u pisanju programa na jeziku C, jednostavan program koji ima zadatak da na ekranu ispiše poruku " Zdravo svete ":

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

ANALIZA:

U liniji 1. datoteka iostream.h je uključena u program. Prvi karakter u liniji je # znak što predstavlja signal predprocesoru. Svaki put kada pokrenete svoj prevodilac predrocesor se izvršava. Predprocesor analizira izvorni kod, tražeći linije koje počinju sa # i deluje na te linije pre nego što se prevodilac pokrene. *Include* je instrukcija predrocesoru koja kaže, "Ono što sledi je naziv datoteke. Nađi tu datoteku i postavi je na ovo mesto." Znači manje i veće oko naziva datoteke ukazuju predprocesoru da potraži tu datoteku na svim uobičajenim mestima. Ukoliko je vaš prevodilac pravilno konfigurisan, znaci < i > će ukazati predrocesoru 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.

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

Datoteka zaglavlja	Pretprocesorska komanda include	Deklariše funkcije
stdio.h	`#include <stdio.h>`	Standardne ulazne i izlazne funkcije jezika C, uključujući i one koje rade s datotekama.
iostream.h	`#include <iostream.h>`	Klase toka (samo za C++) koje mogu da zamene `printf` i `scanf`. U poglavlju 4 (str. 369) objašnjena je upotreba klasa toka.
string.h	`#include <string.h>`	Funkcije za rad sa znakovnim nizovima; na primer, kopiranje jednog znakovnog niza u drugi.
ctype.h	`#include <ctype.h>`	Funkcije za proveravanje i menjanje tipa pojedinačnih znakova u znakovnom nizu.
math.h	`#include <math.h>`	Trigonometrijske, logaritamske, eksponencijalne i ostale zanimljive funkcije s kojima inženjeri vole da se igraju.
malloc.h	`#include <malloc.h>`	Funkcije jezika C za dinamičko zauzimanje i oslobađanje blokova memorije operativnog sistema. (U C++-u su za ovu namenu predviđeni još i operatori `new` i `delete`.)

Linija 3. označava stvarni početak programa funkcijom main(). Svaki C++ program ima main() funkciju. Generalno, funkcija je blok koda, koja obavlja jednu, ili više 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 vrednost će vratiti. Tip povratne vrednosti za main() u ZDRAVO.CPP je void, što znači da ova funkcija neće vratiti nikakvu vrednost.

Sve funkcije počinju otvorenom velikom zagradom ({) i završavaju se zatvorenom velikom zagradom (}). Zagrade za main() funkciju su u linijama 4 i 7. Sve izmedu 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. Evo kako se cout koristi: otkucajte reč cout, iza koje treba da se nađe operator izlazne redirekcije (<<). Sve što se nalazi iza operatora izlazne redirekcije biće odštampano na ekranu. Ukoliko želite da odštampate niz karaktera, postavite ga izmedu dvostrukih navodnika ("... "), kao što je prikazano u liniji 5.

Tekstualni string (eng. *text string)* je serija karaktera koji se mogu odštampati.

Zadnja dva karaktera /n upućuju cout da pređe u novi red posle reči Zdravo svete! Ovaj specijalni kod biće detaljno objašnjen kasnije.

//Kratak pogled na cout

```
#include <iostream.h>
int main()
{
cout << "Zdravo!\n";
cout << "Ovo je 5: " << 5 << "\n";
cout << "Manipulator end1 nas prevodi u novi red" << endl;
cout << "Evo veoma velikog broja: \t" << 10000 << endl;
cout << "Ovo je zbir 8 i 5: \t\t" << 8+5 << end1;
cout << "Evo razlomka: \t\t\t" << (float) 5./8. << endl;
cout << "I veoma velikog broja: \t\t" << (double) 7000*7000 << endl ;
cout << "Student Ime i prezime je C++ programer! \n";
return 0;
}
```

ANALIZA:

U liniji 3. naredba #include<iostream.h> dovodi do uključivanja datoteke iostream.h u izvorni kod. Ovo je potrebno ako konstite 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: "Ovo je 5: ". Obratite pažnju na prazno mesto posle dve tačke. Prazno mesto je deo stringa. Sledeća vrednost je 5, koja se prosleđuje 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 ekranu.

U liniji 9. uvodimo /t novi znak za formatiranje. On umeće tab karakter i koristi se u linijama 8. do l2. 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 prosleđena cout. ali je 13 odštampano.

U liniji 11. za cout je prosleđeno *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 a termin (double) se koristi da ukaže da želite prikaz rezultata u eksponencijalnom zapisu.

U liniji 13. napisali ste uz svoje ime i prezime da je C ++ programer.

Na sledećem primeru vidimo upotrebu naredbi za ulaz - izlaz i komentare:

```
// Program Zad2.cpp
#include <iostream.h>
int main()
{
// Ucitavanje prve promenljive
cout << "a= ";
double a;
cin >> a;
// Ucitavanje druge promenljive
cout << "b= ";
double b;
cin >> b;
// Postavljanje parametra p
double p;
p = 2*(a+b) ;
// Postavljanje parametra s
double s;
s = a*b;
// Stampanje p i s
cout << "p= " << p << "\n";
cout << "s= " << s << "\n";
return 0;
}
```

3.1 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.

3.1.1. 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 naiđe 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 predpostavili, 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.

3.1.2. Korišćenje komentara

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

Sledeći listing demonstrira korišćenje komentara pokazujući da oni ne utiču na izvršavanje programa, niti na njegov izlaz:

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

3.1.3 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 konšćeni pri izradi progama)
- dodatne beleške po potrebi.

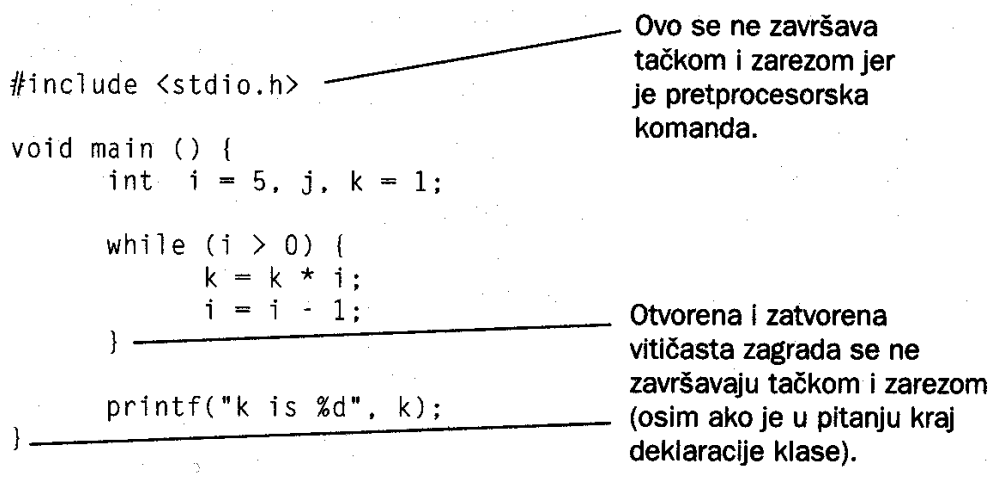
Veoma je važno obezbediti ažurnost beleški i opise ažurnim. Standardni problem sa zaglavljima je što se zanemare posle početnog kreiranja i tokom vremena postanu više smetnja nego pomoć. Ali, ako se propisno ažuriraju, ona mogu biti od neprocenjive pomoći u razumevanju celog programa.

3.1.4. Obratite pažnju na tačku i zarez!

Verovatno najčešćša greška početnika u jezicima C i C++ jeste izostavljanje znaka tačka i zarez (;), ali se dešava da se nađe i tamo gde je izlišan. Pravilo za korišćšćenje tačke i zareza je sledeće: svaki iskaz završite tačkom i zarezom osim u slučaju:

- pretprocesorske komande, kao što su #include i #define,
- Složenog iskaza. U praksi, ovo znači da tačku i zarez ne pišete posle završne vitičaste zagrade ć izuzev ako je to kraj klase ili deklaracije promenljive.

Jednostavan program prikazan u produžetku ilustruje ovo pravilo i oba izuzetka.



U C++-u se za označavanje kraja iskaza koriste tačka i zarez, a ne fizički kraj reda. Kad god vidite da iskaz neće stati na ekran, proširite ga u više redova. Na primer:

printf("Dana %d/%d/%d, temperatura je bila %f,\n",
dan,
mesec,
godina,
temp);

Još jedna osobenost sintakse C++-a jeste mogućnost pisanja nekoliko iskaza u jednom redu, poput narednog primera sa četiri dodele. Ne zaboravite da se tačkom i zarezom završavaju svi iskazi, uključujući i poslednji.

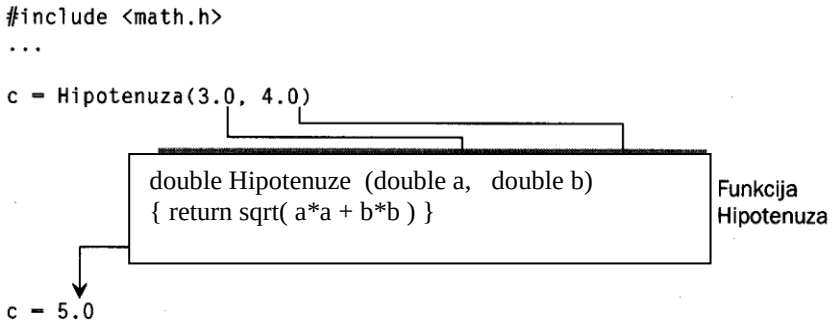
a = 0; b = 0; c = 0; d = 0;

3.2. Funkcije

U C++-u postoji samo jedna vrsta podprogramiranja -funkcija. Funkcije koje ne vraćaju nikakvu vrednost deklarišu se sa *void*, a one koje je vraćaju imaju povratni tip kao što je *int, double* ili *float*. Ovo pojednostavljuje sintaksu C++-a -ne postoje zasebne reči Sub i Function kao u Visual Basicu.

Sigurno ste se ranije sretali s funkcijama. Funkcija može imati jedan argument ili više njih, a ne mora imati nijedan -u zavisnosti od toga kako je deklarisana -i izračunavanjem daje vrednost koja se potom može koristiti u većem izrazu. Na narednoj slici je prikazano kako bi mogao da radi poziv funkcije Hipotenuza.

Rezultat izraza Hipotenuza(3.0. 4.0) je poziv funkcije Hipotenuza i prosleđivanje vrednosti 3.0 i 4.0 dva parametra. Funkcija koristi iskaz return da bi vratila kontrolu onome ko je pozvao funkciju i da bi vratila dužinu hipotenuze -vrednost 5.0.



3.2.1. Opšta sintaksa za funkcije

Opšta sintaksa u C++ programu kada su u pitanju funkcije sledi špo šablonu prikazanom na sledećoj slici.

Pre nego što funkciju pozovete morate je deklarisati; tome služe *prototipi_funkcije.* Prototip funkcije pruža prevodiocu informacije o tipu tako da on zna koji tip argumenata da očekuje.

direktive_include
prototipi_funkcija
void main()
{ deklaracije_i_iskazi }
povratni_tip ime funkcije(argumenti){deklaracije i iskazi }

Format prototipa funkcije izgleda skoro identično kao format prvog reda (zaglavlja) definicije funkcije:

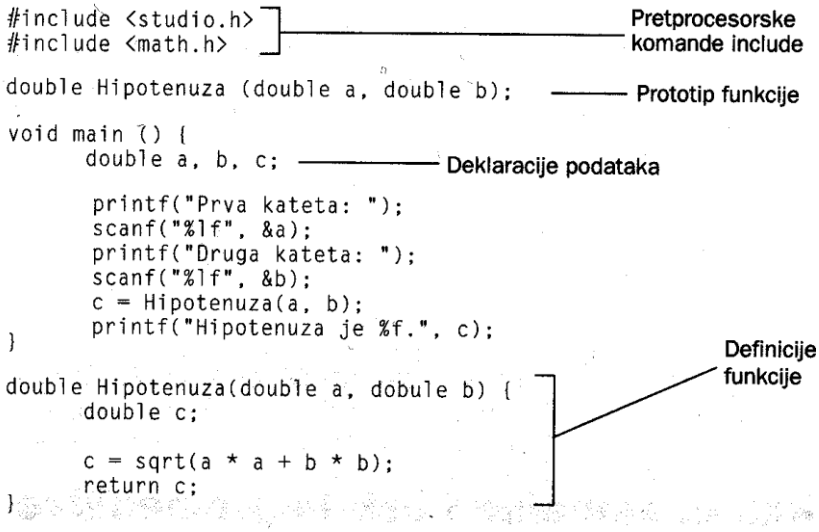
povratni_tip ime_funkcije(argumenti) ;

Prototip funkcije se završava tačkom i zarezom (;). Nemojte stavljati tačku i zarez iza završne vitičaste zagrade ć *definicije* funkcije. Po tome ih i jeste lako razlikovati.

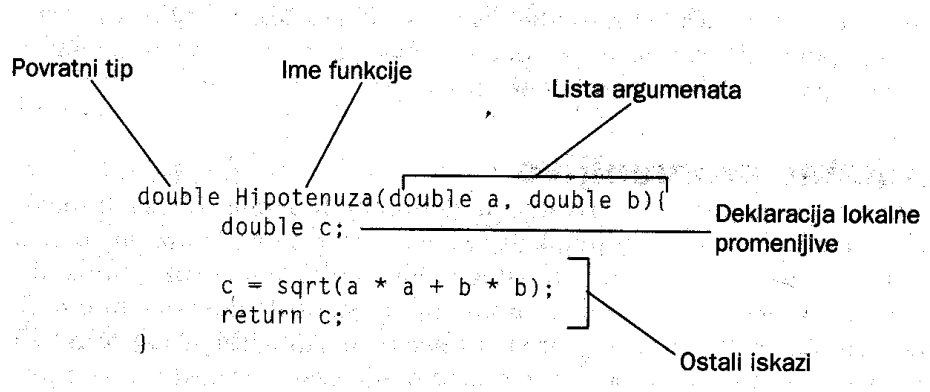
Primer:

Sintaksa funkcije ima više smisla kada se pokaže na primeru. Slika u nastavku ilustruje svaki deo sintakse funkcije uključujući prototip, poziv funkcije i definiciju funkcije. Prototip priprema za poziv funkcije (tako što javlja prevodiocu za koje tipove treba da proverava), poziv funkcije izvršava funkciju, a definicija funkcije govori programu kako da izvrši funkciju.

U ovom primeru morate naznačiti dve datoteke zaglavlja (stdio.h i math.h) jer program koristi ulazno-izlazne funkcije (printf i scanf) kao i matematičku funkciju (sqrt), koja vraća kvadratni koren broja. Datoteke zaglavlja imaju prototipe za ove funkcije.



Na narednoj slici analizirana je sintaksa definicije funkcije Hipotenuza. Povratni tip je double, što znači da funkcija vraća rezultat u obliku dvostruko preciznijeg broja u formatu pokretnog zareza.



Funkcija main( ) nije tipična. Tipične funkcije se pozivaju tokom izvršavanja programa. Program se izvršava liniju po liniju, po redu kojim se pojavijuju 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 potroši, 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 na sledećem listingu:

```
// Demonstiranje pozivanja funkcije
#include <iostream.h>

//Funkcija: Demonstracija funkcije
//stampa korisnu poruku
void DemonstrationFunction()
cout<< "Mi smo unutar Demonstracione funkcije/n";
//Funkcija main -stampa poruku, zatim
//poziva DemonstrationFunction, a zatim, stampa
//druqu poruku.
int main()
{

cout << "Unutar main\n";
DemonstrationFunction();
cout << "Nazad u main\n";
return 0;
}
```

Izlaz je:
Unutar main
Mi smo unutar Demonstracione funkcije
Nazad u main

3.2.1. Korišćenje funkcija

Funkcije vraćaju ili vrednost ili void, što znači da ne vraćaju ta. 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 main(int a, int b)
```

Parametar funkcije (eng. *parameter)* je deklaracija tipa vrednosti, koja će biti prosleđena funkciji: stvarna vrednost prosleđena 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.

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

U listingu je prikazana funkcija sa dva celobrojna parametra, koja vraća celobrojnu vrednost. Nemojte brinuti o sintaksi ili specifičnosti rada sa celobrojnim vrednostima (na primer int x).

```
#include <iostream.h>
int Add (int x, int y)
{
cout << "U Add(), preuzimam " << x << " i " << y << "\n";
return (x+y);
}
int main()
{
cout << "Ja sam u main()!\n";
int a, b, c;
cout << "Unesite dva broja: ";
cin >> a;
cin >> b;
cout << "\nPozivam Add()\n";
c=Add(a,b);
cout << "\nNazad u main().\n";
cout << "c ima vrednost " << c;
cout << "\nIzlazim....\n\n";
return 0;}
```

Izlaz:
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....

PAŽNJA: Funkcija **void** ne vraća nikakvu vrednost i nema iskaz return, na primer:

```
#include <stdio.h>
void ispis(int i1; int i2, int i3);
 void main()
{ int a, b, c;
a = b = c = 1;
ispis(a, b, c);
a =b = c = 2;
ispis(a, b, c) ; }

void ispis(int i1, int i2, int i3) {
printf("Vrednost promenljive param1 je %d.\n", i1);
printf("Vrednost promenljive param2 je %d.\n", i2);
printf("Vrednost promenljive param3 je %d.\n\n", i3); }
```

4. Elementi programskog jezika C++

Skup znakova koji se koriste u jeziku C++ čine mala i velika slova engleskog alfabeta, deset decimalnih cifara i veći broj znakova interpunkcije. Pravi se razlika izmedu malih i velikih slova.

Leksički simboli su nedeljivi nizovi znakova. U jeziku C++ dele se na identifikatore, konstante, službene reči, operatore i separatore. Identifikatori služe za označavanje svih vrsta elemenata programa: promenljivih, simboličkih konstanti, tipova podataka, oznaka, i funkcija. Mogu da se sastoje od slova, cifara i znaka podvučeno (-), stim da prvi znak ne sme da bude cifra. Mogu da imaju proizvoljnu dužinu s tim da je najmanje 31 znak značajan. Izuzetak su "spoljašnja" imena kod kojih su najmanje 6 znakova značajna i to bez razlikovanja malih i velikih slova.

Službene reči jezika C++ su rezervisane reči i ne mogu da se koriste kao identifikatori. Potpuni spisak službenih reči dat je u nastavku:

asm, car, bool, break, marry, catch, to char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, to register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t

Leksički simboli mogu da se pišu, sa nekoliko izuzetaka, spojeno ili medusobno razdvojeno proizvoljnim brojem "belih" znakova. U bele znakove spadaju znak za razmak, tabulacija, vertikalna tabulacija, prelazak u novi red i prelazak na novi list.
U širem smislu, u bele znakove se ubrajaju i kornentari. Komentari su, kako smo već videli, proizvoljni tekstovi koji stavljeni izrnedu /* i * / služe kao objašnjenje čitaocu programa. Mogu da se stave izrnedu bilo koja dva leksička simbola (kao i beli znakovi) i mogu da se protežu i kroz više redova.

4.1. Tipovi podataka

Tipovi podataka određuju moguće vrednosti koje podaci mogu da imaju i moguće operacije koje mogu da se izvode nad tim podacima. Podaci mogu da budu *prosti* (skalarni, nestrukturirani) ili *složeni* (strukturirani). Prosti podaci ne mogu da se dele na manje delove koji bi mogli nezavisno da se obraduju, složeni podaci sastoje se od nekoliko elemenata koji i sami mogu da budu prosti ili složeni.

Od prostih tipova jezik C++ poznaje samo numeričke tipove: Medu njima razlikuju se celi brojevi i realni brojevi.

Osnovne celobrojne tipove čine tipovi *char* i *int*. Varijante tih tipova označavaju se dodavanjem modifikatora ispred ovih oznaka:

unsigned char, signed char, short int, long int, unsigned int, unsigned short int i unsigned long int.

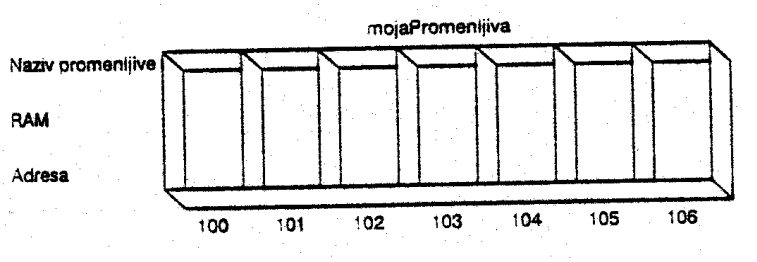
Ako se koristi modifikator, reč int može da se izostavi (na primer: *short* znači *short int*).

Ime/Simbol formata	Dužina u bajtima	Opis	Dijapazon
char/%c	1	Znak ili broj dužine 8 bita	signed:-128 -127 unsigned: 0-255
short/%d	2	Celi broj od 16 bita	-32763 do 32762

			0 do 65535
long/%d	4	Celi broj od 32 bita	-2147483648 do 2147483647 0 do 4294967295
int/%d	16,32,64 ? System	Celi broj	vidi short i long
float/%f	4	Broj sa dec. zarezom	3.4 e±38 (7 cifre)
double/%f	8	Broj sa dec. zarezom dvostruke tačnosti	1.7e±308(15 cifre)
long double/%f	10	Broj sa dec. zarezom višestruke tačnosti	1.2e±4932(19 cifri)
bool	1	logički	true ili false

Osnovne realne tipove čine tipovi *float* (jednostruka tačnost) i *double* (dvostruka tačnost), a jedina varijanta tih tipova označava se sa *long double* (višestruka tačnost).

U programskom jeziku C++ ***promenljiva*** (engl. variable) je mesto za čuvanje informacija određeno svojim imenom i lokacijom u memoriji računara. Ime promenljive, npr. mojaPromenljiva može zauzeti zavisno od veličine (tipa) jednu ili više memorijskih adresa.



Rezervisanje memorije:

Kada defiete 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 vednosti želite da skladištite u svojoj promenljivoj.

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

Veličina celobrojnih promenljivih:

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

Promenljiva tipa char (koristi se za smeštanje znakova) je, najčešćšće, dugaćčka jedan bajt. Celobrojna promenljiva tipa short integer na većini račćunara 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 Vam može pomoći da odredite tačnu veličinu pojedinih tipova podataka na svom računaru. Obratiti pažnju na escape karaktere!

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

```
4:{
5: cout << "Velicina tipa int je: \t\t" << sizeof(int) <<bajta. \n";
6: cout << "Velicina tipa short je: \t\t" << sizeof(short) <<bajta. \n";
7: cout << "Velicina tipa long je: \t\t" << sizeof(long) <<bajta. \n";
8: cout << "Velicina tipa char je: \t\t" << sizeof(char) <<bajta. \n";
9: cout << "Velicina tipa float je: \t\t" << sizeof(float) <<bajta. \n";
10: cout << "Velicina tipa double je: \t\t" << sizeof(double) <<bajta. \n";
11:
12: return 0;
13: }
```

IZLAZ:

```
Velicina tipa int je: 2 bajta.
Velicina tipa short je: 1 bajt.
Velicina tipa long je: 4 bajta.
Velicina tipa char je: 1 bajt.
Velicina tipa float je: 4 bajta.
Velicina tipa double je: 8 bajta.
```

\b	**povratnik**
\f	**form feed**
\n	**nova linija**
\r	**carriage return**
\t	**horizontalni tab**
\v	**vertikalni tab**
**\ **	**obrnuta kosa crta**
\"	**dvostruke navodnice**
\ '	**jednostruke navodnice**
\ (carriage return)	**nastavak linije**
\ nnn	**vrednost karaktera**

Tabela Escape karaktera

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 vednost.

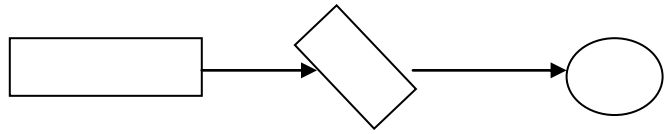
Pošto imamo isti broj bajtova, koje zauzimaju i signed i unsigned celobrojne promenljive najveći broj koji se može skladištiti u unsigned integer je dva puta veći od najvećeg pozitivnog broja koji se može skladištiti u signed integer; unsigned short integer može da prima brojeve od 0 do 65.535. Pola 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.

Definisanje promenljive:

Veličine čije se vrednosti menjaju u toku izvršavanja programa nazivaju se *promenljive.* Promenljivoj se u programu dodeljuje ime, u svakom trenutku ona je definisana svojom vrednošću. Kažemo da je svaka promenljiva u programu povezana sa tri pojma:

imenom -identifikatorom promenljive.
referencom -pokazivačem koji određuje mesto promenljive u memoriji i
vrednošću -podatkom nad kojim se izvršavaju operacije.

Veza između imena, reference i vrednosti promenljive može se predstaviti sledećim dijagramom:



Promenljivu možete da kreirate, ili defiete, navodeći njen tip, unošenjem sa jednog ili više 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; korišćšćenje dobrih naziva čini lakšim razumevanje toka programa. Sledeća naredba defie celobrojnu promenljivu po nazivu mojUzrast: ***int mojUzrast;***

Kao opštšu programersku praksu, nemojte koristiti takve rogobatne nazive, poput J23qrsnf, i zadržite korišćšćenje 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 kol i koJos.

Primer 1:

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

Primer 2:

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

Lokalne, globalne i ostale promenljive:

Jedno od svojstava promenljive je i oblast važenja (engl. *scope).* Ovo svojstvo određuje gde u programu promenljiva može da se vidi. ČČetiri osnovne oblasti važenja nasleđene su iz jezika C: lokalne, globalne, statičke i (spoljašnje). Njima je C++ dodao oblast važenja klase koji povezuje promenljivu s objektom određene klase.

Lokalne promenljive

Lokalna promenljiva je privatna u definiciji funkcije. Svaka funkcija može imati svoju promenljivu koja se zove i , na primer , i može je menjati a da to ne utiče na promenljivu istog imena u bilo kojoj drugoj funkciji. Da biste deklarisali lokalnu promenljivu, smestite deklaraciju u definiciju funkcije. Primera radi, promenljiva c je lokalna u sledećoj definiciji funkcije Hipotenuza, i njene promene ne utiču na vrednost promenljive c u bilo kojoj drugoj funkciji.

```
#finclude <math.h>
.....
double Hipotenuza(double a, double b)
{
double c;
c = sqrt(a * a + b * b);
return c;
}
```

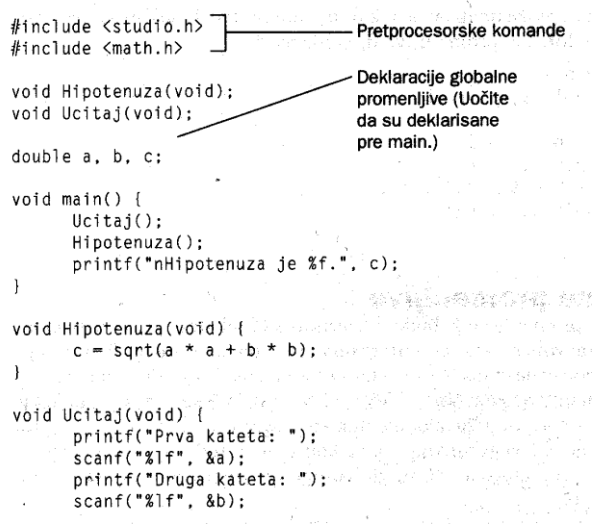
Jedna od razlika između C++-a i ostalih programskih jezika sastoji se u tome što je u C++-u main funkcija kao svaka druga. Ona ima dve jedinstvene karakteristike: ulazna je tačka programa i ne treba joj prototip.
Funkcija main može imati svoje lokalne promenljive. U sledećem primeru su a, b, c i h lokalne promenljive funkcije main.

```
void main()
{
double a, b, c, h;
a = b = c = 1;
ispis(a, b, c);
h = Hipotenuza(a, b);
ispis(a, b, c);
}
```

Globalne promenljive

Po pravilu, promenljivu je bolje deklarisati kao lokalnu, jer se tada može precizno kontrolisati koji deo programa može da je promeni. Ponekad je, međutim, potrebna oblast važenja koja seže van okvira jedne funkcije. Globalne promenljive imaju oblast važenja i životni vek cele izvorne datoteke, tj. svoje vrednosti zadržavaju sve dok se program izvršava. Ovakve promenljive omogućavaju funkcijama da komuniciraju deleći informacije.

NAPOMENA : Globalne promenljive se vide od mesta na kome su deklarisane do kraja izvorne datoteke. One se često deklarišu pri vrhu datoteke.



Statičke promenljive

Statička promenljiva kombinuje vidljivost lokalne promenljive i dužinu trajanja globalne. To je korisno kada vam treba lokalna promenljiva koja zadržava vrednost između poziva funkcije. Na primer:

void ispis(int il. int i2. int i3)
{ static int broj = 0;
printf("Vrednost promenljive i1 je %d.\n", i1);
printf("Vrednost promenljive i2 je %d.\n", i2);
printf("Vrednost promenljive i3 je %d.\n", i3);

broj = broj + 1
printf("Pozvana sam %d put(a).\n\n", broj);}

4.1.1. Konstante

Za sve osnovne tipove, uključujući i njihove varijante postoje odgovarajuće konstante. Celobrojne konstante mogu da budu decimalne (ako prva cifra nije 0), oktalne (ako počinju sa 0) ili heksadecimalne (ako počinju sa 0x ili 0X). Zavisno od vrednosti, one su tipa *int* ili *long*.

int MyAge=48; students=classes*15;

Znakovne konstante su celobrojne konstante (tip *int*) čije su vrednosti kodovi navedenih znakova. Pišu se u jednom od oblika *'z', '\000', '\xhh', '\Xhh'* ili *'\u'*, gde su *z* bilo koji štampajući znak. Nabrojenene konstante su celobrojne (tip *int*) konstante koje se defiu nabrajanjem u naredbama oblika:

enum ime_nabrajanja { ime_konstante=vrednost_konstante , -} ;
enum COLOR { red, blue, green, white, black }
enum COLOR { red=100, blue, green=400, white, black }

Ime nabrajanja
je identifikator pomoću kojeg je moguće kasnije pozivati se na posmatrano nabrajanje. *Imena konstanti* su identifikatori simboličkih konstanti kojima se dodeljuju vrednosti konstantnih celobrojnih izraza označenih sa *vrednost konstante.*

Ako iza imena neke konstante ne stoji vrednost, dodeliće joj se vrednost koja je za jedan veća od vrednosti prethodne konstante u nizu, odnosno koja je nula ako se radi o prvoj konstanti u nizu. Za realne konstante koristi se isključivo decimalni brojevni sistem i podrazumeva dvostruka tačnost (double). Za jednostruku tačnost treba da se koristi sufiks f ili F, a višestruku tačnost l ili L. Najopštiji oblik za sve tačnosti je eksponencijalni oblik mEe, gde se m naziva mantisa, a e eksponent. Vrednost predstavljenog broja je $m*10^e$. Delovi najopštijeg oblika mogu da nedostaju, ali decimalna tačka u mantisi ili eksponent moraju budu prisutni.

Definisanje konstanti može biti različito:

#define studentPerClass=15 (nema određeni tip već je u pitanju čista zamena)
const unsigned short int studentPerClass=15 (ima određeni tip)

4.1.2. Podaci

Podaci se defiu naredbama oblika:

modifikator opis_tipa naziv_podatka =početna_vrednost ,... , naziv_podatka =početna_vrednost ;

Opis tipa može da bude ime bilo koje od gornjih osnovnih tipova sa svim mogućim varijantama, oznaka nekog složenog tipa ili detaljan opis složenog tipa. Za slučaj nabrajanja treba pisati enum *ime_nabrajanja,* ili celokupan opis nabrajanja.

Naziv podatka za proste tipove je jednostavno identifikator podatka za dole opisa izvedene tipove uz identifikator mogu da stoje još i odgovarajući *modifikatori.*

Početna vrednost može da bude proizvoljan izraz odgovarajućeg tipa pod uslovom da svi operandi u izrazu irnaju definisane vrednosti u momentu izvršavanja posmatrane deklarativne naredbe. U nedostatku *=početna_vrednost* odgovarajući podatak ima slučajnu početnu vrednost. Dodela početne vrednosti prilikom definisanja podataka naziva *inicijalizacijom,* a sama početna vrednost *inicijalizatorom.*

Modifikator na početku naredbe, ako postoji, može da bude const iii volatile. Ako nema modifikatora, definisanim podacima se može menjati vrednost u toku izvršavanja programa *(promenljivi podaci).* Modifikator const označava da vrednosti definisanih podataka ne mogu da budu promenjene u toku izvršavanja programa *(nepromenljivi podaci).* U tom slučaju moraju da budu navedene početne vrednosti za sve podatke, tj. moraju da se navedu inicijalizatori. Bez obzira na nepromenljivost, takvi podaci ne mogu da se koriste na mestima gde se izričito zahtevaju konstante. Modifikator volatile označava da se vrednosti definisanih podataka mogu promeniti mimo kontrole programa *(nepostojani podaci).*

Svakom osnovnom tipu, a i dole opisanim izvedenim tipovima, može da se dodeli neko ime naredbom oblika:

typedef oznaka_tipa Naziv_tipa ;

Oznaka tipa podleže istim pravilima kao i u slučaju naredbe za definisanje podataka.

Naziv tipa za slučaj prostih tipova je identifikator koji se dodeljuje odabranom tipu. Za neke izvedene tipove uz taj identifikator treba da stoje i odgovarajući modifikatori. Identifikatori koji se uvode naredbama typedef kasnije mogu da se koriste kao oznaka tipa ravnopravno sa imenima osnovnih tipova.

Polazeći od gore opisanih osnovnih tipova podataka može da se sastavi neograničen broj *izvedenih tipova*. U izvedene tipove u jeziku C++ spadaju pokazivači, nizovi, unije i strukture od bitova.

Pokazivači

su prosti podaci u koje mogu da se smeste adrese podataka ili potprograma u operativnoj memoriji. Broj bajtova koje zauzima neki pokazivač zavisi od mogućeg opsega adresi na datom računaru, a ne od broja bajtova koje zauzima pokazivani podatak.

Pokazivači se deklarišu naredbama za definisanje podataka dodavanjem modifikatora * (promenljivi pokazivač), *const (nepromenljivi pokazivač) ili *volatile (nepostojani pokazivač) ispred identifikatora pokazivača u nazivima podataka. Oznaka tipa na početku naredbi predstavlja tip pokazivanih podataka. Eventualni modifikitor ispred oznake tipa označava nepromeljivost ili nepostojanost pokazivanih podataka, a ne pokazivača samih.

Dva pokazivača su istog tipa ako pokazuju na podatke istih tipova. Specijalan tip za pokazivače je generički pokazivač (tip void *) kod kojeg se ne zna tip pokazivanih podataka. Postoji jedna jedina konstanta pokazivačkog tipa, 0 koja označava da dati pokazivač ne pokazuje ni na jedan podatak. Za simboličko označavanje te vrednosti može da se koristi simbolička konstanta NULL.

Nizovi su složeni podaci koji se sastoje od više elemenata međusobno jednakih tipova. Ako su elementi nekog niza prostog tipa, govori se o jednodimenzionalnom nizu ili vektoru. Niz je dvodimenzionalni (matrica) ako su mu elementi jednodimenzionalni nizovi. Nizovi mogu da budu sa proizvoljnim brojem dimenzija. Podaci tipa niza dek1arišu se naredbama za definisanje podataka dodavanjem modifikatora (dužina) iza identifikatora niza. Oznaka tipa u naredbi označava tip elemenata niza. *Dužina* predstavlja broj elemenata niza i treba da je celobrojni konstantni izraz. Elementi niza se obeležavaju rednim brojevima 0, 1, ..., *dužina-l* koji se nazivaju indeksima elemenata. Za višedimenzionalne nizove potrebno je navesti po jedan modifikator gornjeg oblika za svaku dimenziju. Znakovni nizovi (tip *string)* ne postoje kao posebni tip podataka. Za njih se koriste nizovi znakova (tip char 1) u kojima se iza poslednjeg korisnog znaka nalazi još jedan element sa sadržajem ' /0 ' . Konstante nizovnog tipa postoje samo za znakovne nizove i pišu se u obliku "tekst", gde je *tekst* proizvoljan niz znakova koji čini vrednost konstantnog znakovnog niza. Između znakova navoda mogu da se koriste svi ranije pomenuti oblici predstavljanja znakovnih konstanti.

Početne vrednosti podacima tipa niza mogu da se dodeljuju u obliku niza vrednosti stavljenih između para vitičastih zagrada: š*vrednost* , *vrednost* , ..., *vrednost* ć s tim da *vrednosti* moraju da budu isk1učivo konstantni izrazi. U slučaju niza znakova (tip char 1) može da se koristi i notacija za konstantne znakovne nizove. Za više dimenzionalne nizove svaka vrednost i sama treba da bude niz vrednosti gornjeg oblika.

Strukture

su složeni tipovi podataka koji se sastoje od uredenih nizova elemenata koji mogu da budu međusobno različitih tipova. Ti elementi nazivaju se poljima strukture. Polja struktura se obeležavaju identifikatorima. Opšti oblik opisa strukture u naredbama za definisanje podataka je:

struct ime_strukture { niz_deklaracija }

Ime strukture je identifikator definisane strukture. Ono nema status identifikatora tipa tako da u naredbama za definisanje podataka i u naredbama typedef kao opis tipa mora da se piše Struct *ime_strukture. Niz deklaracija* se sastoji od niza naredbi za definisanje podataka, pri čemu nije dozvoljeno korišćenje modifikatora const i volatile, niti navođenje početnih vrednosti. Identifikatori koji se uvode na taj način predstavljaju identifikatore polja strukture. Bez obzira na svoju složenost, strukture se smatraju pojedinačnim podacima (ne nizovima) i često mogu da se koriste na isti način kao i prosti (skalarni) podaci. Notacija za navođenje početnih vrednosti struktura je slična kao i u slučaju nizova, s tim što navedene vrednosti moraju da se slažu po tipu sa odgovarajućim članovima strukture.

Unije
su složeni tipovi podataka koje omogućavaju da se u isti memorijski prostor, u raznim trenucima, smeštaju podaci različitih tipova. Opšti oblik opisa unija je istovetan opisu struktura, osim što umesto reči struct treba da se koristi reč union. Za razliku od struktura, gde svi članovi istovremeno imaju definisane vrednosti, kod unija svakog momenta samo jedan član ima definisanu vrednost. To je poslednji od članova kome je dodeljena vrednost. Početna vrednost može da bude dodeljena samo prvom članu date unije.

Strukture od bitova
su strukture čija su polja dužine nekoliko bitova koji se, na mašinski zavisan način pakuju u mašinske reči računara. Defiu se opisima struct kod kojih su sva polja nekog celobrojnog tipa (najčešće unsigned int). Iza identifikatora svakog od polja treba da se navede dužina u obliku : d, gde je d broj bitova koliko dati član zauzima.

4.2. Operatori i izrazi

Operatori predstavljaju radnje koje se izvršavaju nad operandima dajući pri tome određene rezultate. Izrazi su proizvoljno složeni sastavi operanada i operatora.

Tip operacije	Operacija	Operator
Multiplikovane operacije	stepenovanje-korenovanje	pow(x,y)
	množenje	*
	deljenje	/
	ostatak deljenja	%
Aditivne operacije	sabiranje	+
	oduzimanje	-
Unarne operacije	plus	+
	minus	-
Relacije	manje	<
	manje ili jednako	<=
	veće	>
	veće ili jednako	>=
	jednako	=
	nejednako	đ=

Aritmetički operatori
služe za izvođenje osnovnih aritmetičkih operacija. U njih spadaju binarni operatori za sabiranje (+), oduzimanje (-), množenje (*), deljenje (/) i nalaženje ostatka deljenja celih brojeva (%). Unarni operator + nema nikakvog efekta, a služi za izmenu algebarskog znaka broja. U jeziku C++ postoje još dva unarna operatora za povećavanje (++) i smanjivanje (--) vrednosti operanda za jedan. Ova dva operatora mogu da budu napisana ispred operanda (prefiksna notacija) kada je vrednost izraza nova vrednost operanda, ili iza operanda (postfiksna notacija) kada je vrednost izraza vrednost operanda pre promene.

Relacijski operatori
služe za upoređivanje numeričkih podataka i daju logičke vrednosti. U jeziku C++ to znači celobrojnu vrednost 0 za logičku neistinu, odnosno celobrojnu vrednost 1 za logičku istinu. Ti operatori se obeležavaju sa == (jednako), ! = (različito), < (manje), <= (manje ili jednako), > (veće) i >= (veće ili jednako).

Logički operatori
služe za izvođenje logičkih operacija nad logičkim podacima dajući logički rezultat. U jeziku C++ nulta vrednost operanda smatra se logičkom neistinom, a bilo koja nenulta vrednost logičkom istinom. Rezultati su 0 za logičku neistinu i isključivo 1 za logičku istinu. Postoje unarni operator za logičku *ne(!)* operaciju i binarni operatori za logičku *i* (&&) i *ili* (||) operaciju. U slučaju operatora && prvo se izračunava vrednost prvog operanda i ako je to =0, rezultat je 0 i vrednost drugog operanda uopšte se ne izračunava. u slučaju operatora đ prvo izračunava se vrednost prvog operanda i ako je to različit od 0, rezultat je 1 i vrednost drugog operanda se uopšte ne izračunava.

Operatori po bitovima
od standardnih viših programskih jezika postoje samo u jeziku C i jezika koji su proizašli iz jezika C *(C++, Java).* Oni služe za manipulacije sa bitovima unutar celobrojnih podataka. Postoje unarni operator za komplementiranje bit po bit (~) i binarni operatori za logičke operacije i (&), *uključivo ili* (|) i *isključivo ili* (^), bit po bit. Pored toga postoje binarni operatori za pomeranje binarne vrednosti prvog operanda ulevo (<<) i udesno (>>) za broj mesta koji je jednak vrednosti drugog operanda.

Uslovni operator (?:)
je specifičan ternarni operator jezika C. lma tri operanda i piše se u obliku *a?b:* C. Prvo se izračunava vrednost izraza a i ako ima vrednost logičke istine (različito od 0), izračunava se vrednost izraza *b* i to predstavlja vrednost celog izraza. Ako a ima vrednost logičke neistine (=0), izračunava se vrednost izraza c i to predstavlja rezultat celog izraza. Bitno je da se od izraza *b* i c uvek izračunava samo jedan.

Adresni operatori
služe za manipulisanje sa adresama podataka. U užem smislu, tu spadaju unarni operatori za nalaženje adrese datog podatka (&) i za posredni pristup podatku pomoću pokazivača (*), koji se naziva i operatorom indirektnog adresiranja). Naime, dozvoljeno je na vrednost nekog pokazivača (adrese) dodati celobrojnu vrednost, odnosno od nje oduzeti celobrojnu vrednost. Jedinica mere promene adrese u tim slučajevima je veličina pokazivanih podataka. Drugim rečima ako pokazivač p pokazuje na neku komponentu datog niza, tada je p+l pokazivač na narednu, a p-l na prethodnu komponemu tog niza. Slično tome, dozvoljeno je oduzeti vrednost dva pokazivača koji pokazuju na elemente istog niza. Pošto je i sada jedinica mere veličina pokazivanih podataka, rezultat je razlika indeksa ta dva elementa niza. Na kraju, dozvoljeno je i upoređivati vrednosti dva pokazivača relacijskim operatorima, s tim da upotreba operatora <, <=, > i >= ima smisla samo ako oba operanda pokazuju na komponente istog niza.

Operatori za dodelu vrednosti
su, takođe, jedna od specifičnosti jezika C. Naime, dok je na drugim jezicima dodela vrednosti naredba, u jeziku C je operator (=) i zbog toga, može da bude i više dodeljivanja vrednosti u sastavu istog izraza (naredbe). Pored toga, postoje još deset operatora dodele vrednosti za slučajeve kada rezultat nekog binarnog operatora treba da se smešta u prvi operand tog operatora.To su operatori +=, -=, *=, /=, %=, &=, ^ =, <<= i >>=. Izraz a@=b se tumači kao a=a@b, osim što se izraz a izračunava samo jednom. Pošto se strukture smatraju pojedinačnim podacima, mogu se međusobno dodeljivati operatorom za dodelu vrednosti (=).

Operator zarez (,) služi za stvaranje niza izraza. U slučaju izraza a, b prvo se izračunava vrednost izraza a, posle toga vrednost izraza b i rezultat celokupnog izraza je b, nezavisno od vrednosti a. Ovo u slučaju niza od više izraza (na primer: a, b, c, d) znači da se izrazi u nizu izračunavaju sleva udesno i vrednost celog izraza je vrednost poslednjeg izraza u nizu.

Operator za konverziju tipa
vrši konverziju tipa vrednosti svog operanda u odgovarajuću vrednost naznačenog tipa. Izraz za konverziju tipa podatka a u tip *T* piše se u obliku (*T) a._T može* da bude oznaka tipa bilo kog osnovnog tipa, identifikator koji je nekom tipu dodeljen naredbom typedef ili oznaka pokazivačkog tipa na podatke proizvoljnih tipova.
Veličina podataka može da se dobije unarnim operatorom sizeof. Operand može da bude naziv tipa podataka unutar para okruglih zagrada ili proizvoljan izraz. U slučaju izraza, dobija se veličina rezultata bez izračunavanja vrednosti izraza. Po definiciji sizeof (char) je uvek 1. Za slučaj nizova rezultat je veličina celog niza (veličina komponenti pomnožena sa brojem komponenti).

Indeksiranje,
kako se naziva pristup elementima nizova, u jeziku C se smatra binarnim operatorom ([]) koji se piše oko svog drugog operanda: a [b] , gde a predstavlja niz (u obliku identifikatora niza ili adresnog izraza), a *b* indeks (redni broj) željene komponente u obliku celobrojnog izraza. Izraz sa indeksiranjem a [b] i izraz adresne aritmetike * (a + b), su međusobno istovetni. Istovetnost važi i za izraze &a [b] i a+b. Identifikator niza u izrazima predstavlja pokazivač na prvu komponentu niza.

Pristup poljima struktura i unija vrši se binarnim operatorima '.' ili '->'. Prvi operand operatora . treba daje podatak, a operatora -> pokazivač na podatak tipa strukture ili unije. Drugi operand oba operatora treba daje identifikator polja strukture ili unije. Izraz a -> istovetan izrazu (*a) .b.

Pozivanje funkcije
u jeziku C smatra se binarnim operatorom (( )) koji se piše svog drugog operanda: f(a,b,c). Prvi operand je pozivana funkcija f predstavljena identifikatorom ili adresnim izrazom. Drugi operand je niz argumenata a, *b*, c. Argumenti a, *b* i c su proizvoljni izrazi potrebnih tipova. Zarez (,) ovde predstavlja separator među argumentima, a ne operator *zarez* za stvaranje niza izraza. Ako neki od argumenata treba je niz izraza, isti mora da se stavlja unutar para okruglih zagrada.

Lvrednost
je izraz koji označava neki podatak ili funkciju u memoriji. lmena podataka predstavljaju *lvrednosti*. Rezultat primene operatora indirektnog adresiranja (*)-i indeksiranja ((I) je *lvrednost.* Operandi unarnih operatora &, ++ i --, kao i operand na levoj strani svih operatora za dodelu vrednosti moraju da budu *lvrednosti.* Kovanica *lvrednost* upravo potiče otuda: to je vrednost koja može da stoji na levoj strani operatora za dodelu vrednosti. Ako su operandi nekog operatora različitih tipova, vrši se automatska konverzija operanda "jednostavnijeg"

tipa u tip operanda ,.složenijeg" tipa i onda se izračunava rezultat operatora. Najjednostavniji mogući tip rezultata je int (tipovi char i short uvek se pretvore u tip int). za njim slede tipovi unsigned int, long, unsigned 1ong, float, double i long double. Kod dodele vrednosti tip numeričkog rezultata koji se dodeljuje pretvara se u tip numeričkog podatka kome se vrši dodeljivanje. U slučaju pokazivača dodela vrednosti dozvoljena samo ako su oba operanda pokazivači na podatke istih tipova, osim ako je jedan od operanada generički pokazivač (tip void*). U ostalim slučajevima mora da bude eksplicitno naznačena konverzija tipa dodeljivanog pokazivača. Redosled izvršavanja susednih operatora u složenim izrazirna određuje se zagradama (( )), prioritetom operatora i smerom grupisanja operatora. Redosled izračunavanja operanada operatora i argumenata funkcija u jeziku C propisan (izuzev za operatore &&, " ? : i *zarez* (,). Zbog toga redosled izvršavanja nesusednih operatora nije jednoznačno određen. Smatra se lošim stilom programiranja pisanje izraza čiji rezultali zavise od redosleda izračunavanja operanada operatora i argumenata funkcija. Potencijalni izvori zavisnosti od redosleda izračunavanja operanada su operatori(funkcije koje daju više od jednog rezultata, tj. koje pored glavnog rezultata daju i dodatne *bočne efekte*. Operatori sa bočnim efektima su ++, --i svi operatori dodele vrednosti.

Nivo	Operator	Opis	Sintaksa
1	`()`	Pozivanje funkcije	*funkcija*(*argumenti*)
1	`[]`	Pristupanje elementu niza	*niz* [*ceobroj*]
1	`->`	Pristupanje članu	*pok*->*član*
1	`.`	Pristupanje članu	*obj*.*član*
1	`::`	Razrešavanje opsega	*klasa*::*simbol* ::*simbol*

Nivo	Operator	Opis	Sintaksa
2D	!	Logička negacija (NOT)	!*ceobroj*
2D	~	Negacija (NOT) nad bitovima	~*ceobroj*
2D	++	Uvećanje za jedan	++*lvred* *lvred*++
2D	--	Umanjenje za jedan	--*lvred* *lvred*--
2D	-	Aritmetička negacija	-*broj*
2D	*	Dereferenciranje pokazivača	**pok*
2D	&	Adresa	&*lvred*
2D	sizeof	Veličina podatka	**sizeof**(*izraz*) **sizeof**(*tip*)
2D	new	Dodeljivanje memorije	**new** *tip* **new** *tip*(*argumenti*) **new** *tip*[*dužina*]
2D	delete	Uklanjanje iz memorije	**delete** *pok* **delete** [] *pok*
2D	typeid	Informacije o tipu	**typeid**(*izraz*)
2D	operatori konverzije	Konverzija tipova	(*tip*) *izraz*
3	.*	Pokazivač-na-član	*obj*.**pok_mem*
3	->*	Pokazivač-na-član	*pok*->**pok_mem*
4	*	Množenje	*broj* * *broj*
4	/	Deljenje	*broj* / *broj*
4	%	Modulo (ostatak pri deljenju)	*ceobroj* % *ceobroj*
5	+	Sabiranje	*izraz* + *izraz*
5	-	Oduzimanje	*izraz* - *izraz*
6	<<	Pomeranje bitova ulevo	*izraz* << *int*
6	>>	Pomeranje bitova udesno	*izraz* >> *int*
7	<	Manje	*izraz* < *izraz*
7	<=	Manje ili jednako	*izraz* <= *izraz*
7	>	Veće	*izraz* > *izraz*
7	>=	Veće ili jednako	*izraz* >= *izraz*
8	==	Jednako	*izraz* == *izraz*
8	!=	Različito	*izraz* != *izraz*

Nivo	Operator	Opis	Sintaksa
9	&	Konjunkcija (AND) nad bitovima	*ceobroj* & *ceobroj*
10	^	Isključiva disjunkcija (XOR) nad bitovima	*ceobroj* ^ *ceobroj*
11	|	Disjunkcija (OR) nad bitovima	*ceobroj* | *ceobroj*
12	&&	Konjunkcija (AND)	*izraz* && *izraz*
13	||	Disjunkcija (OR)	*izraz* || *izraz*
14D	?:	Uslovni operator	*izraz* ? *izraz* : *izraz*
15D	=	Dodeljivanje vrednosti	*lvred* = *izraz*
15D	+=	Dodeljivanje uz sabiranje	*lvred* += *izraz*
15D	-=	Dodeljivanje uz oduzimanje	*lvred* -= *izraz*
15D	*=	Dodeljivanje uz množenje	*lvred* *= *izraz*
15D	/=	Dodeljivanje uz deljenje	*lvred* /= *izraz*
15D	%=	Dodeljivanje uz deljenje sa ostatkom	*lvred* %= *izraz*
15D	>>=	Dodeljivanje uz pomeranje udesno	*lvred* >>= *ceobroj*
15D	<<=	Dodeljivanje uz pomeranje ulevo	*lvred* <<= *ceobroj*
15D	&=	Dodeljivanje uz konjunkciju nad bitovima	*lvred* &= *ceobroj*
15D	^=	Dodeljivanje uz isključivu disjunkciju nad bitovima	*lvred* ^= *ceobroj*
15D	|=	Dodeljivanje uz disjunkciju nad bitovima	*lvred* |= *ceobroj*
16	,	Operator zarez (vraća izraz2)	*izraz1*, *izraz2*

Operatori inkrementiranja i dekrementiranja:

Operator inkrementiranja ++ i dekrementiranja -- su unarni operatori istog prioriteta kao i unarni operator - ili kast i asocijativnosti s desna na levo. Oba operatora primenjuju se isključivo na promenljive i javljaju se u prefiksnom i postfiksnom obliku.

Operator inkrementiranja: + + promenljivoj p dodaje vrednost 1 , pa važi:

p++ je ekvivalentno p=p+1
++ p je ekvlvalentno p=p+1

Operator dekrementiranja --promenljivoj p oduzima vrednost 1, pa važi:

p-- je ekvivalentno p = p-1
--p je ekvivalenlno p = p-1

Ako u nekom izrazu postoji p++ (postfiksni oblik) vrednost promenljive se prvo koristi u izrazu, pa tek onda inkrementira. U slučaju da u izrazu postoji ++p (prefiksni oblik) promenljiva se prvo inkrementira, pa se tek onda njena vrednost koristi u izrazu.

Slično prethodnom, ako u izrazu postoji p-- (postfiksnl oblik) vrednost se prvo koristi u izrazu, pa tek onda dekrementira, a u slučaju --p (prefiksni oblik) vrednost se dekrementira i tako ažurirana koristi u izrazu.

Operatori inkrementiranja i dekrementiranja su predstavljeni u programu:

```
/*Program za prikaz inkrementa i dekremanta */

main()
{
int a = 0, b = 0, c = 0;
printf("\na=%d b=%d c=%d\n", a, b, c);
 a = ++b + ++c;
printf("a=%d b=%d c=%d\n",a, b, c);

a = b++ + c++;
printf("a=%d b=%d c=%d\n", a, b, c);
a = ++b + c++;
printf("a=%d b=%d c=%d\n", a, b, c);
a = ++c + c;
printf("a=%d b=%d c=%d\n", a, b, c);
}
```

Izlaz

```
a=0 b=0 c=0
a=2 b=1 c=1
a=2 b=2 c=2
a=5 b=3 c=3
a=8 b=3 c=4
```

Izraz *a* = + + c + c iz programa 5ukazuje na izvor čestih grešaka. Naime, ovaj izraz zavisi od realizacije C prevodioca pa ga stoga treba izbegavati. Zbog čega? C++ prevodilac može, izračunavajući ++ c + c prvo da inkrementira + + c pa tek onda da preuzme vrednost promenljive c kao vrednost drugog operanda.

Drugi C++ prevodilac može prvo da preuzme vrednost promenljive c kao vrednost drugog operanda, pa tek potom da inkrementira ++ c. Očigledno da ovakve neodređenosti nisu poželjne u programima. Zato izraz ++ c + c treba pisati u obliku ++c i a=c+c čime se eksplicitno navodi redosled izračunavanja gornjih izraza i time izbegava neodređenost redosleda izračunavanja.

Dinamički objekti

• Operator new pravi dinamički objekat, a operator delete utava dinamički objekat nekog tipa T.

• Operator new za svoj operand ima identifikator tipa i eventualne argumente konstruktora. Operator new alocira potreban prostor u slobodnoj memoriji za objekat datog tipa, a zatim poziva konstruktor tipa sa zadatim vrednostima. Operator new vraća pokazivač na dati tip:

```
Complex *pcl = new Cnmplex(1.3,5.6);
Complex *pc2 = new Complex(-1.0,0) ;
*pcl=*pcl+*pc2 ;
```

• Objekat formiran pomoću operatora new naziva se dinamički objekat, jer mu je životni vek poznat tek u vreme izvršavanja programa. Ovakav objekat nastaje kada se izvrši operator new, a traje sve dok se ne ukine operatorom delete (može da traje i po završetku bloka u kome je nastao):

```
Complex *pc;
void f() {
   pc=new Complex (0.1,0.2) ;
}

void main ( ) {
f( );
delete pc;    // ukidanje objekta *pc
}
```

• Operator delete ima jedan operand koji je pokazivač na neki tip. Ovaj pokazivač mora da ukazuje na objekat nastao pomoću operatora new. Operator delete poziva destruktor za objekat na koji ukazuje pokazivač, a zatim oslobađa zauzeti prostor. Ovaj operator *vraća void.*

• Operatorom new može se napraviti i niz objekata nekog tipa. Ovakav niz ukida se operatorom delete sa parom uglastih zagrada:

```
Comlex *pc = new Complex[10] ;
//. . .
delete [ ] pc;
```

• Kada se alocira niz, nije moguće zadati inicijalizatore. Ako klasa nema definisan konstruktor, prevodilac obezbeđuje podrazumevanu inicijalizaciju. Ako klasa ima konstruktore, da bi se alocirao niz, potrebno je da postoji konstruktor koji se može pozvati bez argumenata.

• Kada se alocira niz, operator new vraća pokazivač na prvi element alociranog niza. Sve dimenzije niza, osim prve, treba da budu konstantni izrazi. Prva dimenzija može da bude i promenljivi izraz, ali takav da može da se izračuna u trenutku izvršavanja naredbe sa operatorom new.

4.3. Naredbe

Naredba je osnovna jedinica obrade u programima. U jeziku C ++ naredba može da bude prosta, složena ili upravljačka.

Prosta naredba se sastoji od izraza iz prethodne tačke iza koje se nalazi terminator , tačka-zarez *(izraz;*). Specijalan slučaj proste naredbe je *prazna naredba* koja se sastoji sarno od tačka-zareza (;).

Složene naredbe predstavljaju složene strukture naredbi kojima se određuje redosled izvršavanja naredbi u programu. Nazivaju se i *upravljačkim strukturama.* Upravljačke strukture mogu da se podele u četiri grupe: sekvenca; selekcije i ciklusi ili petlje.

Upravljačke naredbe služe za prenos toka upravljanja na neko mesto u programu. To su razne naredbe skokova. Sekvenca je najjednostavnija upravljačka struktura i predstavlja niz naredbi koje se izvršavaju jedna za drugom. Niz naredbi koje čine sekvencu stavljaju se između para vitičastih zagrada: šnaredba naredba...naredbać . Svaka naredba u nizu može da bude prosta ili složena. Sekvenca može da bude i prazna (š ć), tj da ne sadrži nijednu naredbu. Pošto na početku svake sekvence, pre izvršnih naredbi, mogu da budu i deklarativne naredbe za definisanje podataka i tipova ova struktura naziva se i *blok.* Dejstvo tih deklarativnih naredbi je do kraja bloka u kome se nalaze. Selekcije su složene naredba koje omogućavaju uslovno izvršavanje jedne ili nijedne naredbe iz skupa od jedne ili više naredbi.

Osnovnom selekcijom vrši se uslovno izvršavanje jedne od dve naredbe. U jeziku C ostvaruje se naredbom oblika:

if (izraz) naredba_1 else naredba_2

Ukoliko logički *izraz* ima vrednost logičke istine *(različito od 0),* izvršava se *naredba_1.* Ako je vrednost izraza logička neistina (=0), izvršava se *naredba_2.* Deo else *naredba_2* može da nedostaje. Tada se, u stvari, radi o uslovnom izvršavanju ili preskakanju *naredbe_l.* Takva upravljačka struktura naziva se i *uslovnim preskokom.*
Prilikom uklapanja osnovnih selekcija, tj. kada su *naredba_l* i/ili *naredba_2* i same osnovne selekcije, ponekad se javlja problem u odlučivanju koja reč else pripada kojoj reči if. Opšte pravilo glasi: službena reč else uvek pripada najbližoj reči if koja se nalazi ispred nje i za koju još nije pronađen njen else deo. Od ovog pravila može da se odstupa korišćenjem sekvenci.

Selekcija pomoću skretnice sastoji se od niza naredbi i celobrojnog izraza čija vrednost određuje prvu naredbu u nizu odakle počinje izvršavanje. U jeziku C ostvaruje se naredbom oblika:

switch (izraz) {
case alfa : niz_ naredbi_ 1
case beta : niz_naredbi_2 ;
... ...
default: niz_naredbi_I .
... ...
case omega : niz_naredbi_N }

Oznake case *alfa,* case *beta, ...,* case omega: označavaju mesta u nizu naredbi odakle počinje izvršavanje ako celobrojni *izraz* ima vrednost alfa, beta, ..., omega, respektivno. Specijalna oznaka
default označava mesto na koje se skače ukoliko vrednost izraza nije jednaka nijednoj od vrednosti u oznakama case. Posle izvršenog skoka, naredbe se izvršavaju redom, do kraja celokupne upravljačke strukture, zanemarujući eventualne usputne oznake. U odsustvu znake default može da se desi da nijedna naredba u nizu ne bude izvršena.

Ciklusi
su upravljačke strukture koje omogućavaju ponovljeno izvršavanje neke naredbe (proste ili složene). U jeziku C postoje tri vrste ciklusa. *Osnovni ciklus sa izlazom na vrhu* ostvaruje se naredbom oblika: while (*izraz*) *naredba.* Na početku svakog prolaska kroz ciklus izračunava se vrednost logičkog *izraza* i ako se dobija logička istina *(različito od 0),* izvršava se *naredba.* Ciklus se završava kada vrednost *izraza* postane logička neistina (=0). Može da se desi da *naredba* ne bude nijednom izvršena.

Generalizovani ciklus sa izlazom na vrhu ostvaruje se naredbom oblika:

for (izraz_l ; izraz_2 ; izraz_3) naredba

Prvo se izračunava vrednost *izraza_l,* kao priprema za ulazak u cik1us. Posle toga, na početku svakog prolaska kroz cik1us izračunava se vrednost logičkog *izraza_2* i ako se dobija logička istina *(različito od 0),* izvršava se *naredba* i izračunava se *izraz_3* kao priprema za naredni prolazak kroz ciklus. Ciklus se završava kada vrednost *izraza_2* postane logička neistina (=0). I ovde može da se desi da *naredba* ne bude nijednom izvršena. Bilo koji od izraza može da nedostaje. U slučaju kada nedostaje *izraz_2,* smatra se da uvek ima vrednost Iogičke istine. To znači da tada ciklus nema prirodan završetak.

Ciklus sa izlazom na dnu
ostvaruje se naredbom oblika: do *naredba* while *(izraz 1*). Prvo se izvršava *naredba* koja čini sadržaj cik1usa. Posle toga izračunava se vrednost logičkog *izraza* i ako se dobija logička istina *(TRUE),* skoči se na ponovno izvršavanje *naredbe.* Cik1us se završava kada vrednost *izraza* postane logička neistina (=0 ili FALSE). U ovom slučaju *naredba* se uvek izvršava bar jednom. Skokovi su upravljačke naredbe kojima se tok upravljanja prenosi na neko drugo mesto u programu.

U jeziku C postoje tri vrste skokova:

Iskakanje iz upravljačke strukture izvršava se naredbom break. Time se postiže preskakanje preostalih naredbi unutar selekcije pomoću skretnice (switch) skakanjem na prvu naredbu neposredno iza selekcije, ili prevremeni završetak cik1usa (while, for, do) skakanjem na prvu naredbu neposredno iza cik1usa.

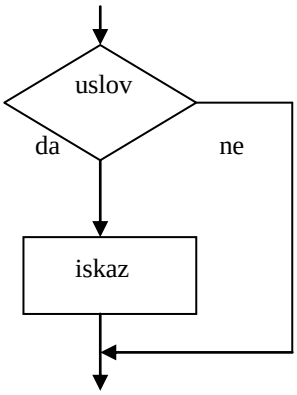
Skok na kraj ciklusa izvršava se naredbom continue. Time se postiže preskakanje preostalih naredbi do kraja cik1usa i izvršavanje radnji vezanih za opsluživanje cik1usa. To u slučaju ciklusa while i do znači ponovno izračunavanje *izraza,* odnosno u slučaju cik1usa for izračunavanje *izraza_3 pa izraza_2.*

Skok sa proizvoljnim odredištem izvršava se naredbom oblika: goto oznaka;
Oznaka predstavlja odredište skoka i po formi je identifikator. Mesto u programu na koje treba skočiti obeležava se umetanjem *oznaka* : ispred naredbe na koju treba da se prenese upravljanje. Odredište skoka može da bude bilo gde u programu, uz jedino ograničenje da ne sme da se uskače u unutrašnjost neke upravljačke strukture.

5. Osnovne strukture upravljanja tokom programa

Obično se prilikom izvršavanja programa on izvršava liniju po liniju kako se one pojavljuju u izvornom kodu. Međutim, postoje službene reči koje omogućuju " skakanje " na različite delove programa (naravno u zavisnosti od rezultata ili od ispunjenja nekog uslova). To su takozvani uslovni operatori i operatori cikličnih struktura.

5.1. Uslovni operator - Iskaz if



Najjednostavniji oblik *if* iskaza je

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

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

Ovaj kod poredi bigNumber i smallNumber. Ako je bigNumber veći, dodeljuje mu se vrednost promenljive smallNumber .
Pošto je blok iskaza unutar vitičastih zagrada ekvivalentan jednom iskazu, sledeći oblik grananja može biti zaista veliki:

```
if (izraz)
{
iskaz1;
iskaz2;
iskaz3);
}
```

Evo jednostavnog primera upotrebe:

```
if( bigNumber > smallNumber)

{
bigNumber = smallNumber;
cout<< "Veliki broj: " <<bigNumber<< "\n";
cout <<"Mali broj: "<<smallNumber << "\n";
}
```

Ovoga puta, ako je bigNumber veći, ne samo da mu se dodeljuje vrednost promenljive smallNumber, već se prikazuje i poruka o tome.

5.1.1. Stilovi uvlačenja

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 ) {
        iskaz1
}
```

- pozicioniranje vitičastih zagrada ispod *if* i uvlačenje iskaza unutar bloka

```
if ( izraz)
{
        iskaz1
}
```

- Uvlačenje i zagrada i iskaza

```
if(izraz)
        {
        iskaz1
        }
```

If iskaz se može pojaviti i u kombinaciji sa *else* službenom reči:

```
if (izraz)
iskaz1;
else
iskaz2;
sledeći iskaz;
```

Znači, ako je izraz tačan izvršava se iskaz1; u suprotnom se izvršava iskaz2. Posle toga program nastavlja sa sledećim iskazom.

5.1.2. Složeni *if* iskazi

Vredno je napomene da se unutar *if* iskaza može koristiti ma koji iskaz pa čak i drugi *if* iskaz. Na taj način se može doći do složenih *if* iskaza u sledećem obliku:

```
if (izraz1)
}
    if(izraz2)
      iskaz1;
else
    {
      if (izraz3)
          iskaz2;
      else
          iskaz3;
    }
}
else
iskaz4;
```

Ovaj složeni *if* iskaz govori: "Ako su izraz1 i izraz2 tačni, izvršite iskaz1. Ako je izraz1 tačan, ali izraz2 nije, tada, ako je izraz3 tačan izvršite iskaz2 a ako nije tačan izvršite iskaz3. I konačno, ako izraz1 nije tačan, izvršite iskaz4".

Primer1: Napisati program u C++ jeziku za izračunavanje funkcije:

$$y = \begin{cases} \lg(x) + 1.82 & x \geq 1 \\ x^2 + 7.x + 8.82 & x < 1 \end{cases}$$

```
// Program Zad20.cpp za izracunavanje vrednosti funkcije
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
```

```
int main()
{cout << "x= ";
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
} // Unesena je validna vrednost za x
double y;
if (x >= 1) y = log10(x) + 1.82;
else y = x*x + 7*x + 8.82;
cout << setprecision(3) << setiosflags(ios :: fixed);
cout << setw(10) << x << setw(10) << y << "\n";
return 0;
}
```

Primer2: Napisati program u C++ jeziku za izračunavanje funkcije:

$$y = \begin{cases} x & x \le 2 \\ 2 & x \in (2,3] \\ x-1 & x > 3 \end{cases}$$

```
// Program Zad22.cpp za izracunavanje vrednosti funkcije
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << "x= ";
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
} // Unesena je validna vrednost za x
double y;
if (x <= 2) y = x; else
   if(x<=3) y = 2; else y=x-1;
cout << setprecision(3) << setiosflags(ios :: fixed);
cout << setw(10) << x << setw(10) << y << "\n";
return 0;
}
```

5.1.3. Operator switch

Često se u problemima nalaže realizacija izbora jedne od mnogo varijanti. U tom slučaju upotreba službene reči *switch* i *case* rešava probleme. Sintaksa operatora *switch* je:

```
switch(<izraz>)     prekidac
{
case <izraz1>:<dejstvo_operatora1>   izbor ili varijanta
case <izraz2>:<dejstvo_operatora2>   izbor ili varijanta
.....................................................................
case <izraz(n-1)>:<dejstvo_operatora(n-1)>    izbor ili varijanta
Šdefault:<dejstvo_operatora(n)>]   opciono
}
```

Naravno vodimo računa da je :
<izraz> je dopustivi tip promenljive bool,int,char (realni tipovi double i float nisu dopustivi)
<izraz1><izraz2>....<izraz(n-1)> su konstantni izrazi koji predstavljaju različite varijante izraza
<dejstvo_operatora> je posledica određenog izbora

Primer1: Napisati program, koji za zadati realni broj x izračunava jedan od sledećih izraza:
1.) y=x-5 2.) y=sin(x) 3.) y=cos(x) 4.) y=exp(x)

```
// program Zad25.cpp
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{
cout<<"====================================\n";
cout<<"|   y = x-5       ->    1       |\n ";
cout<<"|   y = sin(x)    ->    2       |\n ";
cout<<"|   y = cos(x)    ->    3       |\n ";
cout<<"|   y = exp(x)    ->    4       |\n ";
cout<<"====================================\n";
cout<<" l, 2, 3 or 4 ? ";
int i;
cin >> i;
if (!cin)
šcout << "Error. Bad input! \n"; return 1; ć
if (i == 1 đ i == 2 đ i == 3 đ i == 4)
šcout << "x= ";
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n"; return 1; }
double y;
switch (i)
{case 1: y = x-5; break;     //naprimer ako je izabrano i=3
case 2: y = sin(x); break;   //izracunava se y=cos(x) i sa
case 3: y = cos(x); break;   //naredbom break bezulslovno
case 4: y = exp(x); ć        //prekida program
cout << "y= " << y << "\n"; }
else

{cout << "Error. Bad choice! \n"; return 1; }
return 0;}
```

5.2. Operatori cikličnih struktura

Mnogi problemi programiranja se rešavaju ponavljanjem delova na iste podatke. Postoje dva načina da se ovo uradi: rekurzija (vidi funkcije) i iteracija. Iteracija znači ponavljanje iste stvari ponovo i ponovo sve dok ne bude isponjen neki uslov. Glavni metod iteracije je petlja.

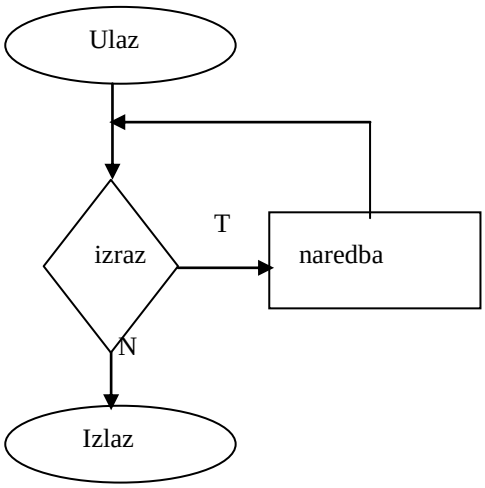
5.2.1. goto iskaz

U vreme rane kompjuterske nauke programi su bili nezgodni, brutalni i kratki. Petlje su se sastojale od labele, nekih iskaza i skoka. U C++ jeziku labela je samo ime, praćeno znakom dve tačke (:) i stavlja se sa leve strane u odnosu na legalan iskaz, a skok se ostvaruje pisanjem *goto* posle koga sledi ime labele.

```
//5.2.1 Upetljavanje sa kljucnom reci goto
// Primer za goto
#include<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;
}
```

IZLAZ:
Brojac: 1
Brojac: 2
Brojac: 3
Brojac: 4
Brojac: 5
Zavrseno. Brojac: 5.

5.2.2. Iskaz while



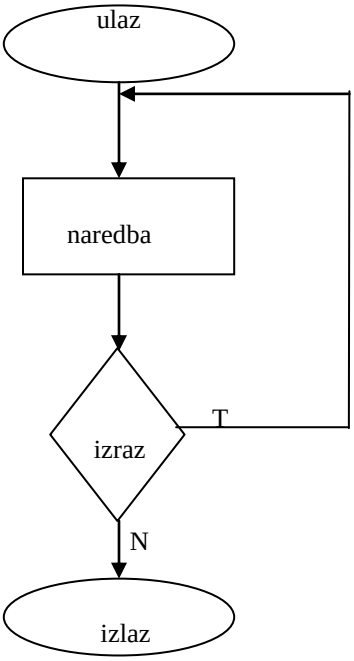
Petlja while prouzrokuje da program ponavlja sekvencu iskaza sve dok je početni uslov istinit. Sintaksa za iskaz while je sledeća:

```
while (uslov)
    iskaz;
```

gde je uslov bilo koji izraz, a iskaz je bilo koji važeći C++ iskaz ili blok iskaza. Kada se uslov proračunava *true*, izvršava se iskaz, a onda se uslov ponovo testira. Ovo se nastavlja sve dok test uslova ne bude *false*, kada se petlja završava i izvršenje programa nastavlja u prvoj liniji ispod iskaza. Predhodni program će mnogo bolje raditi upotrebom while iskaza:

```
//5.2.2 Upetljavanje sa kljucnom reci while
#include<iostream.h>
int main()
{ int counter =0; // inicijalizuje brojac
    while (counter < 5) // testiraj da li je uslov jos istinit
{   counter++;          // telo petlje
    cout << "brojac: " << counter << "\n"; }
    cout << "Zavrseno. Brojac: "<< counter << ".\n";
return 0;}
```

5.2.3. Iskaz do while



Moguće je da se telo petlje nikad neće izvršiti. Iskaz while proverava svoj uslov pre izvršenja svojih iskaza, i ako se uslov proračuna kao *false* celokupno telo petlje while se preskače, tako da je teško postaviti takav uslov da se telo petlje izvrši bar jednom. Tada se upotrebljava do while struktura. Sintaksa do while iskaza je:

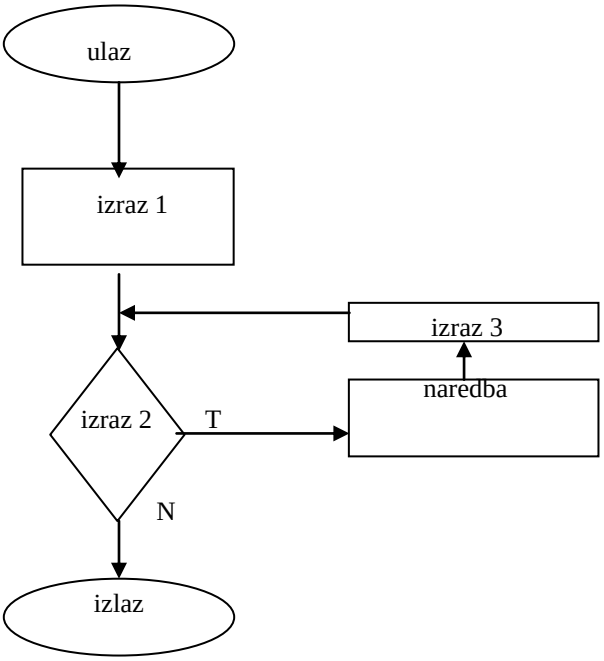
```
do
iskaz
while ( uslov );
```

pa se iskaz izvršava a onda proverava uslov. Ako je uslov true onda se petlja ponavlja, inače petlja se završava.

```
//5.2.3.Demonstracija do while strukture
#include<iostream.h>
int main()
{
int counter;
cout<<"Koliko pozdrava?";
cin>>counter;
do
{     cout<<"Zdravo\n";
     counter--;         }
     while(counter>0);
cout<<"Brojac je: "<<counter<<endl;
return 0;
}
```

IZLAZ:
Koliko pozdrava? 2
Zdravo
Zdravo
Brojac je :0

5.2.4. Petlje for



Prilikom programiranja petlji while, često ćete zateći sebe kako postavljate početni uslov, testirajući da li je on istinit i inkrementirajući, ili menjajući promenljive pri svakom prolasku kroz petlju. Korišćenjem petlje for kombinuju se tri koraka u jedan iskaz. To su

inicijalizacija, test i inkrementiranje. Iskaz for se sastoji od ključne reči *for,* posle koje sledi par zagrada unutar kojih se nalaze tri iskaza rastavljeni znakovima tačka-zarez (;). 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 izraz i proračunava se pri svakom prolasku kroz petlju. Ako test ima vrednost *true* izvršava se akcija u zaglavlju (obično se inkrementira brojač) a onda se izvršava telo petlje for.

//5.2.4. Primer for petlje

```
#include<iostream.h>
int main()
{
    for(int i=0,j=0; i<3; i++,j++)
      cout<<"i:"<<i<<"  j:"<<j<<endl;
      return 0;
}
```

IZLAZ:
i:0 j:0
i:1 j:1
i:2 j:2

//5.2.5. Primer ugnjezdene for petlje

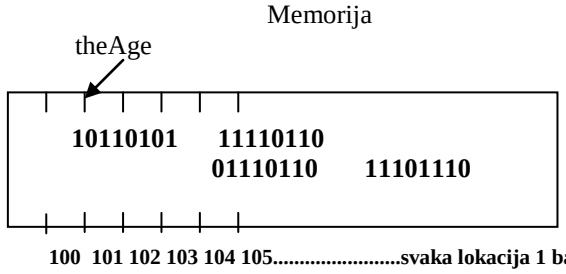
```
#include<iostream.h>
int main()
{
int rows,columns;
char cheChar;
cout<<"Koliko redova?";
cin>>rows;
cout<<"Koliko kolona?";
cin>>columns;
cout<<"Koji karakter?";
cin>>cheChar;

  for(int i=0;i<rows; i++)
    {
     for(int j=0;j<columns; j++)
     cout<<cheChar;
     cout<<"\n";
    }
return 0;
}
```

IZLAZ:
Koliko redova? 4
Koliko kolona? 12
Koji karakter? x
xxxxxxxxxxxx
xxxxxxxxxxxx
xxxxxxxxxxxx
xxxxxxxxxxxx

5.3. Pokazivači:

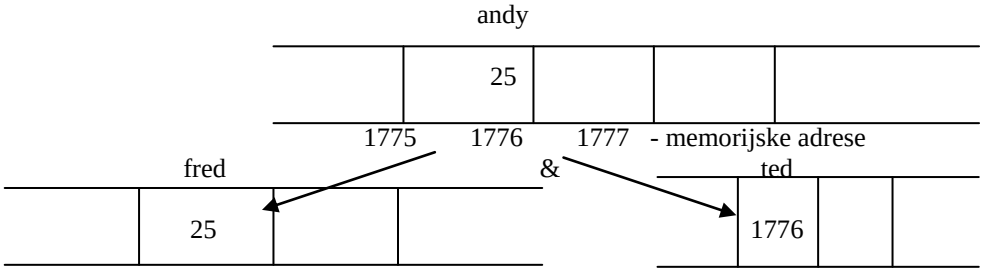
Jedan od najmoćnijih alata koji stoje na raspolaganju C++ programerima su pokazivači, kojima se direktno manipuliše sa memorijom kompjutera. Mogu biti pokazivači adresa ili pokazivači indirekcije.

5.3.1. Pokazivač adresa je promenljiva koja čuva memorijsku adresu ili lvalue. Prikaz promenljive theAge može izgledati:



unsigned long int theAge = 4 bajta = 32 bita, a ime promenljive theAge pokazuje na prvi bajt pa je 102 adresa promenljive theAge

ili:



andy=25; //promenljiva andy postavlja se na vrednost 25 i dodeljuje mem. adresa 1776

fred=andy; //promenljiva fred uzima vrednost promenljive andy tj. 25

ted=&andy //promenljiva ted uzima vrednost adrese promenljive andy

Sintaksa operatora adresa je: ***&<promenljiva>***

Operator & se ne može pisati kao &100 ili &(i+1) !

Različiti kompjuteri označavaju ovu memoriju, koristeći različite, kompleksne šeme. Obično nije potrebno da programeri znaju određenu adresu neke date promenijive, zato što kompajler rukuje detaljima. Ipak, ako želite ovu informaciju, možete upotrebiti operator adresa od (& - ampersand), što je ilustrovano u listingu:

```
//Demonstriranje operatora adresa od, i adresa lokalnih //promenljivih

#include <iostream.h>
int main()
{
unsigned short shortVar=5;
unsigned long longVar=65535;
long sVar = -65535;

cout <<"shortVar:\t" << shortVar;
cout << " Adresa shortVar:\t";
cout << &shortVar<< "\n";

cout << "lonqVar:\t" << longVar;
cout << " Adresa longVar:\t" ;
cout << &longVar << "\n";

 cout << "sVar:\t"<< sVar;
cout << " Adresa sVar:\t" ;
cout << &sVar << "\n";
return 0;
}
```

Izlaz:

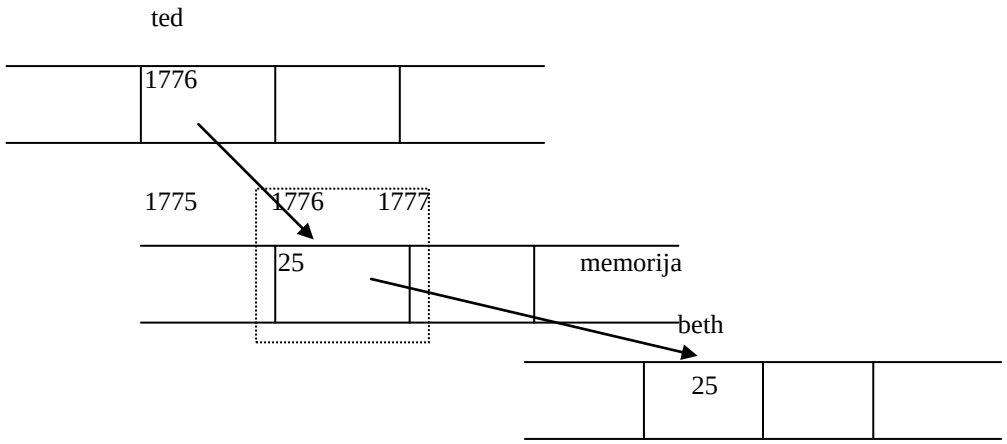
shortVar: 5	Adresa shortVar: 0x1b1a
longVar: 65535	Adresa 1ongVar: 0x1b16
sVar: -65535	Adresa sVar: 0x1b12

5.3.2. Pokazivač indirekcije sa operatorom '*' (naziva se i operator dereferenciranja) omogućuje da i bez znanja specifične adrese date promenljive čuvamo njenu adresu u pokazivaču. Predpostavimo da je how0ld celobrojna promenljiva. Da bi deklarisali pokazivač nazvan pAge za čuvanje njene adrese napisali bi: int *pAge=0. Ovo deklariše promenljivu pAge kao pokazivač na int, što je deklariše i za čuvanje adrese od int. U ovom primeru pAge ima vrednost nula - divlji pokazivači, ali bi uglavnom svi pokazivači morali da imaju vrednost različitu od nule.

```
unsigned short int how01d=50;                  //kreiranje promenljive
unsigned short int * pAge=0        ;           //kreiranje pokazivača
pAge=&how0ld;                                  //stavi adresu od how01d u pAge
```

Prva linija kreira promenljivu how01d čiji je tip unsigned short int, a inicijalizuje je na vrednost 50. Druga linija defie pAge kao pokazivač za tip unsigned short int i inicijalizuje ga na 0. Treća linija dodeljuje adresu promenljive how01d pokazivaču pAge. Znači korišćenjem pAge možemo odrediti vrednost promenljive how01d, pristupajući joj preko pokazivača. Ovakav pristup se naziva indirekcija. Indirekcija znači pristupanje vrednosti preko adrese, koja se čuva u pokazivaču. On obezbeđuje indirektan način za dobijanje vrednosti, koja se čuva na toj adresi. Ako pogledamo sledeći primer:

ted=1776; //ted uzima vrednost 1776
beth=ted; //beth je takođe 1776
beth=* ted; //beth je sada podatak koji se nalazi na adresi 1776 a to je naprimer 25.



// Primer za ukazatelje indirekcije

```
#include <iostream.h>
main ()
{
int value1 = 5, value2 = 15;
int *p1, *p2;
p1 = &value1; // pl = adresa value1
p2 = &value2; // p2 = adresa value2
*p1 = 10; // ukazatelj na p1=10
*p2 = *p1; // ukazatelj p2 jednak je ukazatelju p1
p1 = p2; // p1 = p2
*p1 = 20; // ukazatelj na p1=20
cout << "value1==" << value1 << "/ value2==" << value2 ;
return 0;
}
```

Izlaz:

value1==10 / value2==20

❖ Tip "Pokazivač na tip T oznacava se sa T* . Na primer:

```
int i=0, j=0;        // objekti i i j, tipa int;
int *pi;   // objekat pi je tipa "pokazivač na int" (tip: int*);
pi=&i;               // vrednost pokazivača pi je adresa objekta i, pa pi ukazuje na i.;
*pi=2;               // *pi označava objekat i; i postaje 2;
j=*pi;               // j postaje jedak  objektu na koji ukazuje pi;
pi=&j;               // pi sada sadrži adresu j, tj. ukazuje na j;
```

❖ Na isti način se mogu definisati pokazivači na proizvoljafn tip. Ako je p pokazivač koji ukazuje na objekat klase sa članom m, onda je (*p) .m isto što i p ->m:

```
Osoba otac("Petar Simić", 40);          // objekat otac klase Osoba
Osoba *pc;                              //pc je pokazivač na tip osoba
po=&otac;                               //po ukazuje na objekat otac;
 (*po) .koSi () ;                       // poziv funkcije koSi objekta otac
po->koSi();                             //Isto što i (*po).koSi()
```

```
//Listing za primer sta se cuva u pokazivacu
#include<iostream.h>
typedef unsigned short int USHORT;
int main()
{
unsigned short int myAge=5, yourAge=10;
unsigned short int *pAge=&myAge;          //pokazivac
cout<<"myAge:\t"<<myAge<<"\tyourAge:\t"<<yourAge<<"\n";
cout <<"&myAge:\t" << &myAge << "\t&yourAge:\t" << &yourAge <<"\n";
cout << "*pAge:\t" << pAge <<"\n";
cout << "**pAge:\t" << *pAge <<"\n";
pAge =&yourAge;          // ponovo dodeljuje vrednost pokazivacu
cout << "myAge:\t" << myAge << "\tyourAge:\t" << yourAge <<"\n";
cout << "&myAge:\t" << &myAge << "\t&yourAge:\t" << &yourAge <<"\n";
cout << "pAge:\t" << pAge <<"\n";
cout << "*pAge:\t" << *pAge <<"\n";
cout << "&pAge:\t" << &pAge <<"\n";
return 0;
}
```

Izlaz:

```
myAge: 5                        yourAge:    10
&myAge:     0x1b4e              &yourAge:   0x1b4c
*pAge:      0x1b4e
**pAge:     5
myAge:      5                   yourAge:    10
&myAge:     0x1b4e              &yourAge:   0x1b4c
pAge:       0x1b4c
*pAge:      10
&pAge:      0x1b4a
```

Znači, da bi deklarisali pokazivač, upisaćemo tip promenljive ili objekta, čija će adresa biti čuvana u pokazivaču, praćen operatorom pokazivača (*) i imenom pokazivača. Sagledavajući predhodni primer vidimo kako se korak po korak dodeljuju adrese promenljive pokazivaču. Međutim, u praksi se to svakako ne radi na ovaj način, jer se postavlja pitanje zašto se baratati pokazivačima kada već imamo promenljive i neometan pristup njihovim vrednostima. Zato treba znati da se pokazivači uglavnom koriste za :

- upravljanje podacima na slobodnim skladištima memorije
- pristupanje podacima članovima i funkcijama članicama klase
- predavanje promenljivih funkcijama po referenci

Stek i slobodno skladište:

Pomenućemo pet područja memorije: prostor globalnih imena, slobodno skladište, registri, prostor za kod i 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.

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.

O slobodnom skladištu možete razmišljati kao 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 dao 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 sećate 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 memorju 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 obezbeđuje usko kontrolisan interfejs ka tim podacima i elimie 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 između funkcija. Sledeće sekcije 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 1ong 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;

U svakom slučaju, pPointer sada pokazuje na unsigned short int na slobodnom skladištu. Njega možete korisriti kao i svaki drugi pokazivač na promenljivu i dodeliti vednost području memorije pisanjem.

Ovo znači: "Stavite 72 na vrednost u pPointer", ili: "Dodelite vrednost 72 području na slobodnom skladištu na koje pokazuje pPointer.

UPOZORINJE: Svaki put kada alocirate memoriju, korišćenjem ključne reći new, morate proveriti da biste bili sigurni da pokazivač nije nula.

DELETE

Kada završite sa Vašim područjem memorije, morare pozvati delete za pokazivač - on vraća memoriju slobodnom skladištu. Zapamtite da je sam pokazivač, što je suprotno memoriji na koju on pokazuje, lokalna promanljiva. Po povratku iz funkcije u kom je deklarisan, pokazivač izlazi iz opsega i postaje izgubljen. Menorija alocirana sa newe se ne oslobađa 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, naprimer:

delete pPointer;

KADA BRIŠETE POKAZIVAČ, TO JE STVARNO OSLOBA\ANJE MEMORIJE ČIJA SE ADRESA ČUVA U POKAZIVAČU

6. Klase i pojam klase

- *Klasa* je realizacija apstrakcije koja ima svoju internu predstavu (svoje atribute) i operacije koje se mogu vršiti nad njenim instancama. Klasa defie tip. Jedan primerak takvog tipa (instanca klase) naziva se *objektom* te klase (engl. *class object).*

- Podaci koji su deo klase nazivaju se *podaci članovi* klase (engl. *data members).* Funkcije koje su deo klase nazivaju *sefunkcije članice* klase (engl. *member functions).*

- Članovi (podaci ili funkcije) klase iza ključne reči *private* : zaštićeni su od pristupa spolja (enkapsulirani su). Ovim članovima mogu pristupati samo funkcije članice klase. Ovi članovi nazivaju se *privatnim članovima klase* (engl. *private class members).*

- Članovi iza ključne reči *pub1ic* : dostupni su spolja i nazivaju se *javnim članovima klase* (engl. *public class members).*

- Članovi iza ključne reči *protected* : dostupni su funkcijama članicama date klase, kao i klasa izvedenih iz te klase, ali ne i korisnicima spolja, i nazivaju se *zaštićenim članovima klase* (engl. *protected class members)*

.

- Redosled sekcija *public, protected i private* proizvoljan je, ali se preporučuje baš navedeni redosled. Podrazumeva se da su članovi privatni (ako se ne navede specifikator ispred).

- Objekat klase ima unutrašnje stanje, predstavljeno vrednostima atributa, koje menja pomoću operacija. Funkcije članice nazivaju se još i *metodima* klase, a poziv ovih funkcija -*upućivanje poruke* objektu klase. Objekat klase menja svoje stanje kada se pozove njegov metod, odnosno kada mu se uputi poruka.

- Objekat unutar svoje funkcije članice može pozivati funkciju članicu neke druge ili iste klase, odnosno može uputiti poruku drugom objektu. Objekat koji šalje poruku (poziva funkciju) naziva se *objekat-klijent*, a onaj kojije prima (čija je funkcija članica pozvana) je objekat-server.

- Preporučuje se da se klase projektuju bez javnih podataka članova. Podaci članovi treba da budu privatni, osim ukoliko postoje jaki razlozi sa suprotnu odluku. Javne treba da budu samo funkcije članice koje predstavljaju operacije date apstrakcije koje su na raspolaganju korisnicima klase. Zaštićene su obično jednostavne operacije klase koje ne predstavljaju operacije interfejsa date klase, nego su ili proste funkcije za pristup do podataka članova, ili pomoćne funkcije koje služe za implementaciju javnih operacija jer se koriste na više mesta (nastale su algoritamskom dekompozicijom implementacije operacija i lokalizacijom zajedničkih delova). Ovakve jednostavnije operacije su pomoćne (engl. *helper functions)* i često su potrebne i izvedenim klasama, pa su zbogtoga zaštićene.

- Unutar funkcije članice klase, članovirna objekta čija je funkcija pozvana pristupa se direktno, samo navodenjem njihovog imena.

- Kontrola pristupa članovima nije stvar objekta, nego klase: jedan objekat neke klase iz svoje funkcije članice može da pristupi privatnim članovima drugog objekta iste klase. Kontrola pristupa članovima potpuno je odvojena od provere oblasti važenja: najpre se, na osnovu oblasti važenja, odreduje entitet na koji se odnosi dato ime na mestu obraćanja u programu, a zatim se odreduje da li se tom entitetu može pristupiti.

- Moguće je preklopiti (eng1. *overload)* funkcije članice, uključujući i konstruktore.

- Deklaracijom klase smatra se deo kojim se navodi ono što korisnici klase treba da vide. To su uvek javni članovi. Medutim, da bi prevodilac korektno zauzimao prostor za objekte klase, mora da zna njegovu veličinu, pa u deklaraciju klase ulaze i deklaracije privatnih podataka članova.

```
// deklaracija klase complex:
class complex
{
public:
void cAdd(complex);
void cSub(complex) ;
float cRe();
float cIm();
// ...
private:
float real,imag;
} ;
```

- Gore navedena deklaracija je, zapravo, definicija klase, ali se iz istorijskih razloga naziva deklaracijom.

- Pravu deklaraciju klase predstavlja samo deklaracija class S;.Pre potpune deklaracije (zapravo definicije) mogu samo da se defiu pokazivači i reference na tu klasu, ali ne i objekti te klase, jer se njihova veličina ne zna.

Znači, definicija klase predstavlja navođenje svih članova klase. Na osnovu te definicije mora da se zna veličina potrebanog memorijskog prostora za smeštanje pojednih objekata tipa te klase. Klasa se defie opisom *class* čiji je opšti oblik:

class Identifikator { član član -
public: član član -
private: član član - - };

Deklaracijom klase se samo naznači da neki identifikator predstavlja klasu, ali se ta ne kaže o sadržaju klase. Opšti oblik deklaracije klase je:

class Identifikator ;

Posle deklaracije klase mogu da se defiu pokazivači i upućivači na objekte tipa te klase, ali ne mogu da se defiu objekti tipa te klase. Za definisanje objekata neophodno je da prethodno bude navedena potpuna definicija klase. Pri definisanju objekata vrši se i dodela memorije, a za to je potrebno je da se zna veličina objekata!

Identifikator klase služi za identifikaciju klase koja se defie ili deklariše. Ima status identifikatora tipa, pa može samostalno da se koristi u naredbama za definisanje podataka i na drugim mestima gde se očekuje oznaka tipa.

Javni i privatni delovi klase razgraničavaju se oznakama *public* i *private*. Ista oznaka sme i više puta da se koristi. Početni deo klase, pre prve oznake public, je privatan. Članovi klase navedeni u privatnim delovima su privatni članovi, a članovi navedeni u javnim delovima javni članovi.

Član u definiciji klase može da bude:

- Definicija atributa u obliku naredbe za definisanje podataka. Odjednom mogu da se defiu više atributa zajedničkog osnovnog tipa (neki od njih mogu da budu pokazivači, upućivači ili nizovi). Ne mogu da se navedu inicijalizatori, niti atributi mogu da budu nepostojani. Atributi ne mogu da budu tipa klase koja se upravo defie, ali mogu da budu pokazivači ili upućivači na primerke te klase.

- Definicija metode koja se po formi poklapa sa definicijiom običnih (globalnih) funkcija: Za metode koje se defiu u definiciji klase, podrazumeva se modifikator *inline*. Oni se, dakle, ugrađuju neposredno u kod. Tumačenje tela metode se odlaže do kraja definicije klase. To omogućava da se u telu metode koriste i članovi klase čiji identifikatori budu uvedeni tek u nastavku definicije klase. Vrednost metode može da bude tipa klase koja se upravo defie. Takođe, i argumenti tih funkcija mogu da budu tipa tekuće klase. Naravno, mogu da budu i pokazivači ili upućivači na primerke te klase.

- Deklaracija metode koja se po formi poklapa sa prototipom običnih funkclja: Metode koje se u definiciji klase samo deklarišu, moraju da budu definisane na nekom drugom mestu, izvan definicije klase. Te metode, takođe, mogu da se neposredno ugrađuju u kod, ali to treba izričito tražiti modif1katorom *inline* -prilikom definisanja.
- Naredba *typedef* i *enum* kojom se uvodi identifikator tipa ili identifikatori simboličkih konstanti, ali koji ne stvaraju članove klase (ne utiču na veličinu objekata niti na funkcionalnost klase). Doseg svih identifikatora unutar klase je od mesta definisanja do kraja klase. Kaže se da članovi klase imaju klasni doseg.

Definicija klase mora da bude dostupna prevodiocu prilikom prevodenja svakog programskog modula koji stvara objekte date klase. Zato se definicije klasa obično stavljaju u zaglavlja (datoteke .h), čiji se sadržaji u prevodenje uključuju direktivama *include* pretprocesora.

Evo još jednog primera definicije klase:

```
class Nesto
{
int a, b                    // Privatni atributi.
int f (int) ;               // Privatna metoda.
Nesto n;                    // GRESKA: ne moze atribut  tipa Nesto!
Nesto *pn;                  // Pokazivac na Nesto moze.
Nesto &un;                  // Upucivac na Nesto moze.
public:
int c;                      // Javni atribut.
int g (int);                // Javna metoda.
Nesto h (Nesto);            // Metoda tipa Nesto moze,
                            // takodje i argument tipa Nesto.
};
```

Objekti klasnih tipova :

Posle definisanja neke klase, automatski stoje na raspolaganju sledeće radnje nad tom klasom:

- definisanje (stvaranje) objekata i nizova objekata (Š]),
- definisanje pokazivača (*) i upućivača (&) na objekte,
- dodeljivanje vrednosti (=) jednog objekta drugom,
- nalaženje adresa objekata (&) i pristup objektirna na osnovu adrese (*) ili indeksiranjem ([ ]),
- prisrup članovima objekata neposredno (.) ili posredno (->).

Objekti klasnih tipova (primerci klasa) defmišu se uobičajenim naredbama za definisanje podataka. Za oznaku tipa treba da bude odabran identifikator željene klase. Za svaki objekat date klase stvara se zaseban komplet svih atributa te klase. Mada se kaže da objekti sadrže i metode, to ne treba doslovce shvatiti. Prevod svake metode se, naravno, smešta u memoriju samo jednom. Taj jedini primerak se koristi kad god se metoda pozove za bilo koji objekat. Odvojenost se sastoji samo u nezavisnim kompletima lokalnih prolaznih podataka pri svakom pozivanju date metode.

Za pokazivače na objekte važe sva pravila adresne aritmetike:
Dodela vrednosti podrazumeva kopiranje vrednosti svih atributa izvorišnog objekta u odredišni objekat, atribut po atribut. Za slučaj pokazivačkih atributa prenosi se samo vrednost pokazivača, a ne i pokazivani podatak (podobjekat).

Pristup metodama operatorima . ili -> podrazumeva pozivanje te metode (funkcije).
Objekti mogu da budu argumenti funkcija kao i vrednosti funkcija. Objekti kao argumenti funkcija prenose se pomoću vrednosti, tj. objekat koji se navede kao stvarni argument kopira se u odgovarajući formalni argument na isti način kao i prilikom dodele vrednosti. Ako to ne odgovara, treba koristiti pokazivače ili upućivače.

Evo primera definisanja i korišćenja primeraka klase Nesto iz prethodnog odeljka:

```
Nesto n, *pn, &un=n;          // Objekat, pokazivac i upucivac.
n.c = 55;                     // Dodela vrednosti javnom atributu.
int i = n.g (12);             // Poziv javne metode.
n.a = 0;                      // GRESKA: dodela privatnom atributu.
int j = n.f (-2);             // GRESKA: poziv privatne metode.
pn = new Nesto;               // Stvaranje dinamickog objekta.
*pn = un;                     // *pn = un; (un je upucivac!)
un = n.h (*pn) ;              // Argument i rezultat su tipa Nesto.
delete pn;                    // Unistavanje dinamickog objekta.
```

Postoje slučajevi kada neki identifikatori klasnog dosega mogu da se koriste, bez pominjanja konkretnog objekta, i izvan njihove klase. Takvi su, na primer, identiflkatori uvedeni naredbama *typedef* ili *enum*. Pošto ti identitikatori izvan klase ne postoje, neophodno je naznačiti i klasu kojoj oni pripadaju. To je druga primena operatora za razrešenje dosega (::).

Kada se operator :: koristi kao binarni operator, prvi operand mora da bude identifikator klase, a drugi operand identifikator koji je uveden unutar te klase. Binarni operator :: je prioriteta 18 i grupiše se sleva udesno.

Evo primera za korišćenje nabrajanja izvan klase u kojoj je ono definisano:

```
class Prozor {
------------------
public:
enum Boja šCRNA, PLAVA, ZELENA, CRVENA, BELAć;
--------------------
}
Prozor :: Boja boja = Prozor :: BELA;
if (boja != Prozor :: CRNA)  {.....}
```

U klasi Prozor deinisano je nabrajanje Boja sa nekoliko simboličkih konstanti. Izvan klase je definisan podatak boja tipa tog nabrajanja koji je inicijalizovan jednom od simboličkih konstanti. U poslednjoj naredbi promenljiva boja se upoređuje sa jednom od mogućih vrednosti nabrajanja Boja. Za sve identiflkatore iz klase Prozor korišćen je operator : : .

Metode klasa

Metode klasa pored svojih formalnih argumenata imaju još jedan "skriveni" argument. Skriven, zato što se ne vidi u prototipu funkcije. Taj argument je objekat (primerak klase) za koju je metoda pozvana, tj. prvi operand operatora . ili ->. Naziva se i tekućim objektom. Članovima tog skrivenog argurnenta može da se pristupa navođenjem samo identifikatora člana, kao što se to radi sa običnim podacima ili funkcijarna. Adresa skrivenog argumenta, za vreme izvršavanja metode, nalazi se u pokazivaču sa imenom *this*. Tip tog pokazivača unutar svake klase je pokazivač na tu klasu. Kad god se unutar metode klase navodi identiflkator nekog člana bez navodenja konkretnog objekta, podrazumeva se this -> ispred tog identifikatora, tj. pristupa se odgovarajućem članu tekućeg (skrivenog) objekta. Pokazivač *this* eksplicitno se koristi relativno retko. Uglavnom se koristi samo ako tekući objekat (za koji je metoda pozvana) treba da bude vrednost funkcije metode koja se upravo izvršava. Drugi primer korišćenja je kada tekući objekat ili njegova adresa treba da bude argument neke funkcije ili, pak, ako tekući objekat treba uključiti u neku lančanu listu.
Kao i svi argumenti, skriveni argument funkcije člana može da bude nepromenljiv (const) ili nepostojan (volatile). To se u deklaraciji ili definiciji metode označava dodavanjem odgovarajućeg modifikatora iza zatvorene okrugle zagrade na kraju spiska formalnih argumenata:

tip funkcija (argumenti) modifikator ;
tip funkcija (argumenti) modifikator blok

Modifikator
može da bude const, volatile ili oba. Modifikator const označava da metoda neće promeniti vrednost objekta za koji je pozvana. Modifikator volatile označava da je metoda svesna činjenice da vrednost tog objekta može da se promeni bilo kad u toku izvršavanja metode, mimo njene kontrole. Ako se metoda defie unutar definicije klase, nema nikakvog prdblema u korišćenju identifikatora ostalih članova te klase. Svi se oni nalaze unutar istog dosega. Prilikom odvojenog definisanja metode, definicija se nalazi izvan dosega kome pripada metoda. Zato je neophodno operatorom za razrešenje dosega (::) iz prethodnog odeljka navesti ime klase kojoj pripada metoda. Drugi operand u posmatranom slučaju, treba da je identitikator metode koja se defie. Time se doseg navedene klase proširuje na celu definiciju metode, pa unutar tela metode članovi klase mogu da se koriste navođenjem samo njihovih identifikatora.

Evo primera kojim se defiu funkcije članovi klase Nesto :

```
int Nesto :: f (int x)
{
int z = x + a;              // z = + this->a
return z / b;               // z / this->b
}

int Nesto::g (int x)
{
this->a = x *this->b;       // Ovako se obicno ne  pise (a = x *b)
return x / a;
}

Nesto Nesto: :h (Nesto x)
{
Nesto y;
y . a=x . a+a;
y . b = x . b + a;
return y;
}
```

Primerci klase

• Za svaki objekat klase formira se poseban komplet svih podataka članova te klase.

• Za svaku funkciju članicu, postoji jedinstven skup lokalnih statičkih objekata. Ovi objekti žive od prvog nailaska programa na njihovu definiciju, do kraja programa, bez obzira na broj objekata te klase. Lokalni statički objekti funkcija članica imaju sva svojstva lokalnih statičkih objekata funkcija nečlanica, pa nemaju nikakve veze sa klasom i njenim objektima.

• Podrazumevano se sa objektima klase može raditi sledeće:

1. definisati primerci (objekti) te klase i nizovi objekata klase;
2. definisati pokazivači na objekte i reference na objekte;
3. dodeljivati vrednost (operator =)jednog objekta drugom;
4. uzimati adrese objekata (operator &) i posredno pristupati objektima preko pokazivača (operator *);
5. pristupati članovima i pozivati funkcije članice neposredno (operator .) ili posredno (operator ->);
6. prenositi objekti kao argumenti funkcija i to po vrednosti ili referenci, ili prenositi pokazivači na objekte;
7. vraćati objekti iz funkcija po vrednosti ili referenci, ili vraćati pokazivači na objekte.

• Neke od ovih operacija korisnik može redefinisati preklapanjem operatora. Ostale ovde nenavedene operacije korisnik mora definisati posebno ako su potrebne (ne podrazumevaju se).

Konstantne funkcije članice

• Dobra programerska praksa je da se korisnicima klase najavi da li neka funkcija članica menja unutrašnje stanje objekta ili ga samo ,,čita" i vraća informaciju korisniku klase.

• Funkcije članice koje ne menjaju unutrašnje stanje objekta nazivaju se *inspektori* ili *selektori* (engl. *inspector, selector).* Reč const iza zaglavlja funkcije ukazuje korisniku klase da je funkcija članica inspektor. Ovakve funkcije članice nazivaju se u jeziku C++ *konstantnim* funkcijama članicama (engl. *constant member functions).*

• Funkcija članica koja menja stanje objekta naziva se *mutator* ili *modifikator* (engl. *mutator, modifier)* i ne označava se posebno:

```
class X {
   public:
   int read ( ) const     { return i; }
   int write (int j=0)   { int temp=i; i=j; return temp; }
 private:
   int i;
};
```

• Deklarisanje funkcije članice kao inspektora samo je notaciona pogodnost i ,,stvar lepog ponašanja prema korisniku". To je ,,obećanje" projektanta klase da funkcija ne menja stanje objekta koje je projektant klase definisao. Prevodilac nema načina da u potpunosti proveri da li inspektor posredno menja neke podatke članove klase.

• Inspektor može da menja podatke članove pomoću eksplicitne konverzije, koja ,,probija" kontrolu konstantnosti. To je ponekad slučaj kada inspektor treba da izračuna podatak koji vraća, pa ga onda sačuva u nekom članu da bi sledeći put brže vratio odgovor.

• U konstantnoj funkciji članici tip pokazivača this je const X*const, tako da pokazuje na konstantni objekat, pa nije moguće menjati objekat preko ovog pokazivača (svaki neposredni pristup članu je implicitni pristup preko ovog pokazivača). Takođe, za konstantne objekte klase nije dozvoljeno pozivati nekonstantnu funkciju članicu (korektnost konstantnosti). Za prethodni primer:

```
x x;
const X cx;
x.read( );      // u redu: konstantna funkcija nekonstantnog objekta;
x.write( );    // u redu: nekonstantna funkcija nekonstantnog objekta;
cx.read( );   // u redu: konstantna funkcija konstantnog objekta;
cx.write( );  // greska: nekonstantna funkcija konstantnog objekta;
```

Prijateljske funkcije klasa

Prijateljske funkcije neke klase su funkcije koje nisu članovi te klase ali imaju pravo pristupa do privatnih članova te klase. Prijateljske funkcije mogu da budu obične (globalne) funkcije ili da budu metode drugih klasa.

Da bi funkcija postala prijateljska funkcija neke klase, potrebno je u definiciji te klase da se navede njen prototip ili definicija sa modifikatorom *friend* na početku:

```
friend tip funkcija ( argumenti );
friend tip funkcija ( argumenti ) blok
```

Nije bitno da li se funkcija proglašava prijateljskom funkcijom u privatnom ili javnom delu klase, pošto ona nije član posmatrane klase. Ako se navede definicija funkcije, modifikator *inline* se podrazumeva, kao i za članove klase. Uprkos tome, identifikator funkcije neće imati klasni doseg, već pripašće dosegu identifikatora cele klase. Najčešće je to datotečki doseg.

Dešava se da sve metode neke klase treba da budu prijateljske funkcije date klase. Umesto da budu navedeni prototipovi svih tih metoda, sve one mogu da se učine prijateljskim funkcijama naredbom oblika:

friend identifikator_klase ;
friend class identifikator_klase ;

Druga varijanta je potrebna ako prethodno još nije navedena ni definicija niti deklaracija klase čije metode treba da budu prijateljske funkcije. Naredba se, naravno, stavlja u definiciju klase kojoj te metode treba da budu prijateljske funkcije. Prijateljske funkcije se defiu kao i bilo koje druge funkcije. Ne poseduju pokazivač *this* za klasu čije su prijateljske funkcije. Zbog toga mogu da obrađuju samo konkretne objekte te klase. Ti objekti mogu da budu, na primer, formalni argumenti, lokalni objekti, itd. Naravno ako je prijateljska funkcija metoda neke druge klase, poseduje pokazivač *this* za svoju klasu. Data funkcija može da bude prijateljska funkcija većeg broja klasa istovremeno. Nema nikakvih formalnih razloga da se da prednost ostvarivanju funkcija u obliku metoda ispred ostvarivanja u obliku prijateljskih funkcija. Međutim metoda može da bude pozvana samo za "stvarni objekat", dok prijateljska funkcija može i za podatke koji su stvoreni u toku implicitne konverzije tipa (konverzija tipa za slučaj klasnih tipova).

Druga situacija kada se koriste prijateljske funkcije je kada neka funkcija treba neposredno da pristupa člačovima više klasa. Na primer ako uz klasa Matrica i Vektor želi da se realizuje množenje matrice sa vektorom. Ako članovi klase Vektor nisu prijateljske funkcije klasi Matrica, do elemenata matrice mogu da dođu samo pozivanjem odgovarajućih javnih metoda klase Matrica. To, međutim, može biti vrlo neefikasno. Na kraju, neki programeri više vole klasičnu notaciju pozivanja funkcija kao što je f (x) , umesto x .g ( ). Prvi način je pozivanje prijateljske funkcije za objekat x neke klase, a drugi pozivanje metode iste klase.

Evo primera kojim se povlači paralela između korišćenja metoda i prijateljskjh funkcija:

```
class Alfa
{
int x;
public:
void p(int n){ x = n; }                          // Smestanje vrednosti.
int q() { return x; }                            // Uzimanje vrednosti.
friend void r(int n, Alfa &a) { a . x = n; }     // Smestanje vrednosti.
friend int s(Alfa a) { return a . x; }           // Uzimanje vrednosti.
} ;

#include <iostream. h>
void main ()
{
Alfa a;
a . p (55) ;              //Smestanje u objekat metodom.
int i = a.q ();           //Uzimanje iz objekta metodom.
r(55, a) ;                //Smestanje u objekat prijateljskam funkcijam.
int j = s (a);            //Uzimanje iz objekta prijateljskam funkcijom.
cout << i << ' ' << j << endl; }
```

Klasa Alfa ima jedan privatni atribut x. Smeštanje neke vrednosti u taj atribut ili uzimanje vrednosti tog atributa moguće je samo posredstvom funkcija koje su ovlašćene za to. Funkcije p( ) i q( ) te radnje izvode kao metode, a funkcije r( ) i s( ) kao prijateljske funkcije.

Pošto se radi o vrlo jednostavnim funkcijama, predviđeno je njihovo neposredno ugrađivanje u kod. To se podrazumeva, pošto su definisane unutar definicije klase. Treba naglasiti, da će prijateljske funkcije, ipak, biti globalne funkcije sa datotečkim dosegom identifikatora.

Treba uočiti da funkcije za istu operaciju imaju po jedan argument više kod prijateljskih funkcija nego kod metoda. To je objekat kome se pristupa. Kod metoda objekat se podrazumeva prilikom definicije, a pojavljuje se prilikom pozivanja metode kao prvi operand operatora . ili ->.

Funkcija p( ) mora da bude tipa void, jer nema šta da bude njena vrednost funkcije. Funcija r( ) mogla bi, a da se postigne isti konačni efekat, da se detie i kao funkcija tipa Alfa sa jednim argumentom tipa int. Dva reda, u kojima se u gornjem primeru koristi ta funkcija, izgledala bi onda ovako:

friend Alfa r (int) { Alfa w; w . x = n; return w; }
a = r (55);

Treba naglasiti, da se u ovom slučaju isti konačni efekat dobija na osetno drugačiji način. Funkcija r( ) je sada neka vrsta funkcije za konverziju tipa koja podatak tipa int pretvara u objekat tipa Alfa. Pošto su objekat a i funkcija r( ) istih tipova, dodela vrednosti u poslednjem redu je dozvoljena.

Ovo drugo rešenje za funkciju r( ) ima još i tu prednost da može da se koristi i u složenijim izrazima, pod uslovom da su operatori jezika *C++* osposobljeni za prihvatanje operanada tipa Alfa.

Zajednički članovi klasa

Stavljanjem modifikatora *static* na početak definicije ili deklaracije nekog člana date klase, taj član postaje zajednički za sve objekte te klase koji budu stvoreni u toku izvršavanja programa. Članovi koji nisu zajednički, nazivaju se pojedinačnim članovima klase. Očigledno, bez izričitog zahteva članovi klasa su pojedinačni članovi. U slučaju atributa to znači da će postojati samo jedan primerak tog člana, bez obzira na broj objekata tipa te klase. Pristup tom atributu u sastavu bilo kog od tih objekata označava i pristup fizički istoj memorijskoj lokaciji. Menjanjem vrednosti zajedničkih atributa deluje se na stanje svih objekata te klase, a ne samo na stanje jednog od njih. Zajednički atributi su slični globalnim podacima, s tom razlikom da njihove vrednosti mogu da se menjaju samo na način koji odgovara opisu klase.

Dok definisanje primerka date klase podrazumeva definisanje svih pojedinačnih atributa (dodelu memorijskog prostora i eventualnu inicijalizaciju), to nije slučaj za zajedničke atribute. Oni se smatraju zasebnim trajnim podacima sa spoljašnjim povezivanjem. Njihov opis unutar klase se smatra samo deklaracijom. Zajednički atributi se defiu zasebnim naredbama za defnisanje podataka koje se pišu izvan definicija Idasa. Pošto se nalaze izvan dosega svojih klasa, mora da se koristi operator za razrešenje dosega (: :) da bi bilo označeno kojoj klasi pripadaju identifikatori koji se defiu. Prilikom definisanja zajedničkih atributa može da se izvrši i njihova inicijalizacija početnim vrednostima koje mora da budu konstantni izrazi. Dodela početnih vrednosti vrši se pre stvaranja prvog objekta tipa njihove klase, obično na samom početku izvršavanja programa, pre pozivanja funkcije main ( ) .

Zajedničke metode ne poseduju pokazivač *this*. Zbog toga neposredno, navodenjem samo identifikatora člana, mogu da pristupaju samo zajedničkim članovima svojih klasa. Pojedinačnim članovima mogu da pristupaju samo za konkretne objekte. Ti objekti mogu da budu argumenti zaječničkih metoda, lokalni podaci u njima ili globalni podaci. Pristupanje zajedničkim članovima (atributima i metodama) iz svhih vrsta funkcija (metoda i globalnih funkcija) moguće je navođenjem konkretnog objekta (*objekat* . *član* ili *pokazivač -> član)* ili bez navođenja objekta *(Klasa* : : *član).* U metodama klase može da se koristi i samo identifikator člana *(član).* Naravno, *objekat* i *pokazivač* mogu da budu izrazi čije su vrednosti objekat odnosno pokazivač na objekat posmatrane klase. *Klasa* može da bude samo identifikator klase.

Bez obzira na navedene mogućnosti pristupanja zajedničkim članovima izričito se pre- poručuje korišćenje oblika *Klasa* : : *član.* Time se naglašava da se pristupa klasi u celini, a ne samo navedenom objektu. Ako dođe do bilo kakve promene, to će da utiče na stanje cele klase, tj. svih objekata te klase.

Zajednički članovi postoje i pre stvaranja prvog objekta date klase. Razume se, tada im je moguće pristupiti samo izrazom oblika *Klasa* : : *član.* Zajednički članovi koriste se u slučajevima kada svi primerci neke klase čine neku logičku celinu (zbirku podataka) uobličenu u neku složenu strukturu podataka. Kao primer može da se navede lista kod koje primerci klase čine elemente jedne liste. Zajednički član može da bude pokazivač na prvi element liste, a možda i broj elemenata u listi. Pojedinačni članovi te klase mogu da budu sadržaj elementa i pokazivač na naredni element liste. Korišćenjem zajedničkih članova klasa može znatno da se smanji broj globalnih podataka u složenom programskom sistemu. Mana globalnih podataka je što su bez ograničenja pristupačni iz svake funkcije sistema!
Evo primera klase sa zajedničlcim članovima:

```
class Abc{
static int a;                          // Zadjednicki atribut:.
int b;                                 // Pojedinacni atribut:.
public:
static int f() ;                       // Zajednicke metode.
static void g(Abc, Abc *, Abc &) ;
int h(int) ;                           // Pojedinacna metoda.
}
int Abc :: a = 55;                     // Definicija zajednickog atributa.
int Abc :: f () {
int i = a;                             // Dohvatanje zajednickog atributa .
int j = b; ,                           // GREŠKA: Ne moze pojedinacni atribut
                                       // (this->b, a this ne postoji).
return i + j ;
}

void Abc :: g(Abc x, Abc *y, Abc &z) {
int i = x . b;                         // Pristup pojedinacnim atributima
int j = y -> b;                        // objekata koji su argumenti funkcije
z . b = i + j;                         //(vrednost, pokazivac, upucivac) .
}
int Abc :: h(int x) {                  //pojedinacna metoda moze ravnopravno
return (a + b) * x;                    // da koristi i pojedinaxne i zajednixke
}                                      // atribute
```

```
void main(){
int p = Abc :: f () ;          // Moze mada jos ne postoji nijedan objekat.
int q = Abc :: h (5) ;         // GRESKA: mora konkretan objekat .
Abc k;                         // Stavaranje prvog objekta.
int r = k . f ();              // Sada vec moze i ovako (izbegavati).
Abc :: g(k, &k, k);
int s = k . h (7) ;            // Konkretan objekat je k.
}
```

Klasa Abc ima jedan zajednički i jedan pojedinačni atribut. Zajednički atribut a je definisan zasebnom naredbom odmah iza defmicije klase. Tom prilikom je i inicijalizovan vrednošću 55. Zajednička metoda f( ) sme da koristi zajednički atribut a, a ne i pojedinačni atribut b. Za pojedinačne članove, naime, podrazumeva se pristup pomoću pokazivača *this* (tj. this->b), a za zajedničke metode taj pokazivač nije definisan. Metoda g( ) je primer za mogućnosti korišćenja objekata tipa Abc unutar zajedničkih metoda. Konkretni objekti tog tipa mogu da budu argulmenti funkcije, bilo u obliku vrednosti (x), pokazivača na objekte (y) ili upućivača na objekte (z). Metoda h() je pojedinačna metoda. Unutar takvih metoda, formalno, na ravnopravan način mogu da se koriste i pojedinačni i zajednički članovi matične klase. Pokazivač *this* koristiće se za pristup do pojedinačnih članova. Na kraju, glavni program prikazuje nekoliko primera korišćenja objekata klase Abc.

Zajednička metoda f ( ) može da se poziva pre stvaranja bilo kog objekta klare Abc. Ona koristi samo zajedničke atribute klase, a oni postoje od samog početka programa (naravno pod pretpostavkom da se prethodno otkloni greška koja je u gornjem tekstu namerno učinjena). Pojedinačna metoda h( ) ne može da se poziva u drugoj naredbi glavnog programa. Za nju mora da se navede konkretan objekat tipa Abc, a takav objekat još ne postoji. Iz istih razloga na tom mestu ne bi smela da se pozove ni zajednička metoda g( ). Njoj su konkretni objekti potrebni kao argumenti. Posle definisanja objekta k tipa Abc u trećoj naredbi, sve tri metode klase Abc mogu da se pozivaju za konkretan objekat. Ne preporučuje se, medutim, pozivanje zajedničke metode f( ) na način kako se pojedinačne metode moraju pozivati (k .f( )), već se preporučuje oblik pozivanja bez pominjanja konkretnog objekta (Abc : : f( )). Funkcija f( ), bez obzira na način pozivanja, ne može da koristi sve članove objekta k, već samo njegove zajedničke članove. Međutim, ti zajednički članovi ne pripadaju samo objektu k!

Pokazivači na atribute klasa

Pokazivači na atribute klasa pokazuju na određeni atribut primeraka klasa. Za razliku od običnih pokazivača, koji pokazuju na neki konkretan podatak, dodelom vrednosti pokazivaču na članove klasa samo se označi neki član te klase. Objekat, čijem obeleženom članu će da se pristupi, određuje se kao operand odgovarajućeg operatora u momentu pristupanja.

Pokazivač na atribute klasa se defie uobičajenim naredbama za definisanje podataka, s tim da se ispred identifikatora dodaje modifikator *klasa:* : *, gde je *klasa* identifikator klase na čije članove će da pokazuje definisani pokazivač. Oznaka tipa na početku naredbe određuje tip atributa klase na koje može definisani pokazivač da pokazuje. Za nalaženje adrese atributa klase koristi se uobičajeni (unarni) operator & čiji operand treba da bude identifikator člana klase. Pošto se ta adresa određuje izvan dosega date klase, uz identifikator, naravno, mora da se koristi i operator za razrešenje dosega (::). Pristupanje atributima pomoću pokazivača na atribute klasa vrši se pomoću binarnih operatora .* ili ->*. Prvi operand operatora .* treba da je objekat, a operatora ->* pokazivač na objekat date klase. Drugi operand oba operatora je identifikator pokazivača na članove klasa.

Evo primera za definisanje i korišćenje pokazivača na članove klasa:

```
class Alfa { -public: int a, b; -}
int Alfa :: *pc;          // pc je pokazivac na int atribute klase Alfa.
Alfa alfa, *beta;         // Objekat i pokazivac na objekte klase Alfa.
beta = &alfa;             // beta pokazuje na objekat alfa.
pc = &Alfa :: a;          // pc pokazuje na atribute  a objekdta klase Alfa.
alfa .* pc = 1;           // alfa . a = 1;
beta ->* pc = 1;          // beta -> a = 1;
pc = &Alfa :: b;          // pc pokazuje na atribute b objekata klase Alfa.
alfa .* pc = 2;           // alfa . b = 2;
beta ->* pc = 2;          // beta -> b = 2;
```

Pokazivač na atribute klasa pc je definisan da može da pokazuje na bilo koji atribut tipa int objekata klase Alfa. Posle toga, definisan je jedan objekat (alfa) i jedan pokazivač na objekte (beta) klase Alfa. Posle prve dodele vrednosti pokazivaču na atribute pc, alfa .* pc označava član alfa .a, a posle druge dodele vrednosti, doslovce isti izraz označva član alfa .b. Slično je i kada se za pristup koristi pokazivač na objekte beta. Pokazivač na atribute klasa deluje slično kao indeks kod rada sa nizovima. Kao što izraz niz Š i] označva različite komponente niza zavisno od vrednosti indeksa i, tako izraz objekat .*pc označava različite atribute objekta, zavisno od vrednosti pokazivača na članove pc. Treba naglasiti, dok indeks može da bude proizvoljan izraz, pokazivač na članove može da bude samo identifikator!

Lokalne klase

Klase mogu da se defiu unutar funkcija. Takve klase se nazivaju lokalnim klasama. ldentifikator lokalne klase ima blokovski doseg. Unutar lokalne klase dozvoljeno je korišćenje samo identifikatora tipova, trajnih podataka, spoljašnjih podataka i funkcija kao i nabrojanih konstanti iz okružujućeg dosega. Pristupanje članovima lokalne klase iz funkcije unutar koje je definisana podleže svim uobičajenim pravilima i ograničenjima. Metode lokalne klase moraju da se defiu unutar definicije klase. Lokalna klasa ne može da ima zajedničke članove (atribute i metode). Ova ograničenja treba da obeshrabre pokušaje sastavljanja složenih lokalnih klasa.

Evo primera za definisanje lokalne klase:

```
int x;
void f ()
{ static int s;
int x;
extern int g () ;
class Lok {                    // Definicija lokalne klase.
------------------
public:
int h () { return x; }         // GRESKA: x nije trajni podatak.
int j () { return s; }         // trajni podatak s.
int k () { return ::x; }       // Globalni podatak.
int l () { return g(); }       // Spoljasnja funkcija g.
{;
------------
{
Lok *p = 0,                    // GRESKA: Lok nije u dosegu.
```

5.3.3. Prijatelji klasa

• Često je dobro da se klasa projektuje tako da ima i povlaščene korisnike, odnosno funkcije ili druge klase koje imaju pravo pristupa njenim privatnim članovima. Takve funkcije i klase nazivaju se prijateljima *(engl.friends).*

Prijateljske funkcije

• Prijateljske funkcije *(eng.-friendfunctions)* neke klase nisu članice te klase, ali imaju pristup do privatnih članova te klase. Te funkcije mogu da budu globalne funkcije ili članice drugih klasa.

• Da bi se neka funkcija proglasila prijateljem klase, potrebno je bilo gde u deklaraciji te klase navesti deklaraciju te funkcije sa ključnom reči friend ispred. Prijateljska funkcija se defie na uobičajen način:

```
class X {
 public:
     void f(int ip) {i=ip;}
 private:
friend void g (int,X&)       // prijateljska globalna funkcija
friend void Y: : h ( ) ;     // prijateljska clanica druge klase
int i;}
void g (int k, X &x) {
x.i=k;                       // prijateljska funkcija moze da pristupa
                             // privatnim clanovima klase
void main ( ) { X x;
x.f(5);                      // postavljanje preko clanice
g(6,x);                      // postavljanje preko prijatelja
 }
```

• Globalne funkcije koje predstavljaju usluge neke klase ili operacije nad tom klasom (najčešće su prijatelji te klase) nazivaju se *klasnim uslugama* (engl. *class utilities).*

• Nema formalnih razloga da se koristi globalna (najčešće prijateljska) funkcija umesto funkcije članice. Globalne funkcije su ponekad pogodnije:

1. globalnoj funkciji može da se dostavi kao argument i objekat drugog tipa, koji će se konvertovati u potreban tip, dok funkcija članica mora da se pozove za objekat date klase;

2. kada funkcija treba da pristupa članovima više klasa;

3. kada je notaciono pogodnije da se koriste globalne funkcije (poziv je f (x)) umesto članica (poziv je x. f ( )); npr., max (a,b) je čitljivije od a. max (b) ;

• "Prijateljstvo" se ne nasleđuje: ako je funkcija f prijatelj klasi X, a klasa Y izvedena (naslednik) iz klase X, funkcija f nije prijatelj klasi Y.

Prijateljske klase

• Ako je potrebno da sve funkcije članice klase Y budu prijateljske funkcije klasi X, onda se klasa Y deklariše kao prijateljska klasa *(eng.friend class)* klasi X. Tada sve funkcije članice klase Y mogu da pristupaju privatnim članovima klase X, ali obratno ne važi („prijateljstvo" nije simetrična relacija):

```
class X {
      friend class Y;
//...
};
```

• „Prijateljstvo" nije ni tranzitivna relacija: ako je klasa Y prijatelj klasi X, a klasa Z prijatelj klasi Y, klasa z nije automatski prijatelj klasi X, već to mora eksplicitno da se naglasi (ako je potrebno).

• Prijateljske klase se obično koriste kada dve klase imaju tešnje međusobne veze. Pri tome je nepotrebno (i loše) „otkrivati" delove neke klase da bi oni bili dostupni drugoj klasi jer će onda biti dostupni i ostalima (ruši se enkapsulacija). U tom slučaju se ove dve klase proglašavaju prijateljskim. Na primer, na sledeći način može se obezbediti da samo klasa Creator može da stvara objekte klase X:

```
class x {
public:
....
....
private:
    friend class Creator;
    X( );        // konstruktor je dostupan samo klasi Creator
....
....

} ;
```

```
// Primer za prijateljsku funkciju duplicate cijim posredstvom
//mozemo da pristupimo clanovima width i height razlicitih
//objekata tipa CRectangle. Novi clanovi klase  su duzina i sirina
//pravougaonika pomnozene sa dva.

#include <iostream.h>
class CRectangle
{
int width, height;
public:
void set_values (int, int);
int area (void) {return (width * height);}
friend CRectangle duplicate (CRectangle);
};
```

```
void CRectangle::set_values (int a, int b) { width = a;
height = b; }

CRectangle duplicate (CRectangle rectparam)
{
CRectangle rectres;
rectres.width=rectparam.width*2;
rectres.height = rectparam.height*2;
return (rectres);
}
main ( )
{
CRectangle rect, rectb;
rect.set_values (2,3);
rectb = duplicate (rect);
cout <<"\nSada je povrsina: "<< rectb.area();
}


// Primer deklaracije klase CRectangle kao prijatelja klase
//CSquare koja moze da pristupi protected i private clanovima
//klase CSquare, a konkretno do promenljive CSquare::side koja
//definise duzinu strane kvadrata. Obrnuto naravno ne vazi!

#include <iostream.h>
class CSquare; //prazan prototip klase CSquare
class CRectangle
{
int width, height;
public:
int area (void)
{return (width * height);}
void convert (CSquare a);
} ;

class CSquare
{ private:
int side;
public:
void set_side (int a) {side=a;}
friend class CRectangle;
{ ;

void CRectangle::convert (CSquare a) { width = a.side;
height = a.side; }

main ( )
{ CSquare sqr;
CRectangle rect;
sqr.set_side(4);
rect.convert(sqr);
cout << "Povrsina kvadrata je:"<<rect.area();
return 0; }
```

Konstruktori i **destruktori**

Novi objekti (primerci klasa) mogu da se stvaraju:

- izvršavanjem naredbi za definisanje podataka (stalni objekti koji se stvaraju prilikom prvog nailaska na naredbu za definisanje i prolazni objekti koji se stvaraju svaki put kada se dode do naredbe za definisanje),
- stvaranjem lokalnih podataka za formalne argumente prilikom pozivanja funkcija -- (prolazni objekti),
- u toku izvršavanja složenih izraza za odlaganje međurezultata (privremeni objekti), ili
- izvršavanjem operatora new (dinamički objekti). Stvaranje objekata podrazumeva dodelu memorijskog prostora i, eventualno, inicijaliza-ciju dodeljivanjem nekih početnih vrednosti.

Memorijski prostor se dodeljuje automatski za sve atribute klasa prema definiciji klasa. Inicijalizaciju, tj. dodelu početnih vrednosti, atributima stvaranih objekata treba da predvidi programer. Izuzetno, za stalne (static) objekte podrazumevane početne vrednosti su nule u svim bajtovima svih atributa klase. Naročito je važna ispravna inicijalizacija objekata klasa koje imaju i pokazivačke atribute. To može da zahteva i dodatnu dodelu memorijskog prostora u dinamičkoj zoni, što ne može nikako da se uradi automatski.

Inicijalizaciju stalnih, prolaznih i dinamičkih objekata može da predvidi programer. Postoji međutim opasnost da on to propusti da učini. Što se privremenih objekata tiče, njih čak i da hoće, programer ne može da inicijalizuje. Da li će se i koliko privremenih objekata koristiti, zavisi od prevodioca. Privremeni objekti nemaju ni imena, niti pokazivače na njih pomoću kojih bi programer mogao da im pristupi.

Mehanizam koji jezik *C++* nudi za zagarantovanu inicijalizaciju svih kategorija objekata zasniva se na specijalnim metodama klasa koji se nazivaju konstruktorima. Ako su ispunjeni određeni uslovi, konstruktori se pozivaju automatski kad god se stvara neki objekat date klase, bez obzira da li se radi o stalnim, prolaznim, dinamičkim ili pak privremenim objektima. Neinicijalizovani objekat, u stvari i nije objekat, već samo jedno parče memorije čiji sadržaj nema nikakvog smisla. Pravi objekatje nešto drugo, ima " inteligentan " sadržaj koji zadovoljava osobine svoje klase. Zadatak konstruktora je da parče dodeljene memorije pretvori u objekat sa svim obeležjima svoje kJase. Objekte koji više nisu potrebni, treba ispravno utiti. Pod time se, prvenstveno, podrazumeva oslobađanje memorijskog prostora koji je bio dodeljen prilikom stvaranja objekata.

Utavanje objekata se vrši:

- za stalne objekte automatski na kraju programa,
- za prolazne objekte automatski u momentima napuštanja dosega njihovih identifikatora,
- za privremene objekte automatski čim je to moguće (obično po završetku izračunavanja izraza u toku kojeg su stvoreni) i
- za dinamičke objekte na zahtev programera pomoću operatora delete ili automatski na kraju programa.

Memorijski prostor koji je objektima dodeljen za atribute klasa, na osnovu definicije klasa, oslobađa se automatski, bez intervencije programera. Oslobađanje eventualnog dodatnog memorijskog prostora u dinamičkoj zoni mora da predvidi programer. Opet, to može da se predvidi samo za stalne, prolazne i dinamičke objekte, a ne i za privremene objekte. Pored toga, programer može i da propusti da izda zahtev za utavanje nekih objekata.

Mehanizam koji jezik *C++* nudi za zagarantovano utavanje svih kategorija objekata zasniva se na specijalnim metodama klasa koji se nazivaju destruktorima. Ako u nekoj klasi postoji destruktor, biće sigumo automatski pozvan kad god se vrši utavanje nekog objekta te klase. Posle primene destruktora na neki objekat, isti izgubi obeležja svoje klase. Objekat se time opet pretvori u parče "neinteligentne" memorije, kakav je bio pre primene konstruktora.

Konstruktor

- Funkcija članica koja nosi isto ime kao i klasa naziva se konstruktor (engl. *constructor).* Ova funkcija se uvek poziva prilikom nastanka objekta te klase.
- Konstruktor nema tip koji vraća. Konstruktor može da ima argumente proizvoljnog tipa. Unutar konstruktora, članovima objekta pristupa se kao i u bilo kojoj drugoj funkciji članici.
- Konstruktor se uvek implicitno poziva pri nastanku objekta klase, odnosno na početku životnog veka svakog objekta date klase.
- Konstruktor, kao i svaka funkcija članica, može biti preklopljen (engl. *overloaded).* Kon- struktor koji se može pozvati bez stvarnih argumenata (nema formalne argumente ili ima sve argumente sa podrazumevanim vrednostima) naziva se podrazumevanim konstruktorom.
- Ukoliko u klasi nije eksplicitno deklarisan nijedan konstruktor, prevodilac implicitno generiše podrazumevani konstruktor koji je javni i koji vrši podrazumevanu inicijalizaciju podobjekta osnovne klase i objekata - članova pozivima njihovih podrazumevanih konstniktora, ako ih ima (bilo kao eksplicitno deklarisan ili implicitno generisan). Ako takvih konstruktora u odgovarajućim klasama nema, javlja se greška.

Kada se poziva konstruktor?

Konstruktor je funkcija koja pretvara "presne" memorijske lokacije koje je sistem odvojio zanovi objekat (i sve njegove podatke članove) u "pravi" objekat koji ima svoje članove i koji može da prima poruke, odnosno ima sva svojstva svoje klase i konzistentno početno stanje. Pre nego što se pozove konstruktor, objekat je u trenutku definisanja samo "gomila praznih bitova" u memoriji računara. Konstruktor ima zadatak da od ovih bitova napravi objekat tako što će inicijalizovati podobjekat osnovne klase i članove. Konstruktor se poziva uvek kada objekat klase nastaje, tj. kada se:

- izvršava definicija statičkog objekta;
- izvršava definicija automatskog (lokalnog nestatičkog) objekta unutar bloka; formalni argumenti, pri pozivu funkcije, nastaju kao lokalni automatski objekti;
- inicijalizuje objekat, pozivaju se konstruktori njegovih podataka članova;

- stvara dinamički objekat operatorom new;
- stvara privremeni objekat, pri povratku iz funkcije, koji se inicijalizuje vraćenom vrednošću funkcije.

Načini pozivanja konstruktora

Konstruktor se poziva kada nastaje objekat klase. Na tom mestu je moguće navesti inicijali- zatore, tj. stvarne argumente poziva konstruktora. Poziva se onaj konstruktor koji se najbolje slaže po broju i tipovima argumenata (pravila su ista kao i pri preklapanju funkcija):

Klasa (argumenti) : inicijalizator,-----,inicijazizator blok

```
T::T  ( )  :  a( 0 ), b( 1 )  {}
T::T  ( )  : {a( 0 ), b( 1 );}
T::T  (int i )  :  a( i ), b( 2*i )  {}
T::T  (int i )   { a=i, b=2*i;}
T::T  (int i )  :     b(2*i);{ a=i }
```

U slučaju da je definicija konstruktora izvan definicije klase, neophodno je pomoću operatora za razrešenje dosega :: proširiti opseg klase na tu definiciju.

Destruktor

Funkcija članica koja ima isto ime kao klasa, uz znak ispred imena, naziva se *destruktor* (engl. *destructor).* Ova funkcija poziva se automatski, pri prestanku života objekta klase, za sve navedene slučajeve (statičkih, automatskih, klasnih članova, dinamičkih i privre- menih objekata):

```
class x {
public:
~X () { cout<<"Poziv destruktora klase X!\n"; }
}
void main ()
X x;
//...

} // ovde se poziva destruktor objekta x
```

- Destruktor nema tip koji vraća i ne može imati argumente. Unutar destruktora, članovima se pristupa kao i u bilo kojoj drugoj funkciji članici. Svaka klasa može da ima najviše jedan destru tor.
- Destruktor se implicitno poziva i pri utavanju dinamičkog objekta pomoću operatora delete. Za niz, destruktor se poziva za svaki element ponaosob. Redosled poziva destruktora je, u svakom slučaju, obratan od redosleda poziva konstruktora. Ako klasa nema eksplicitno deklarisan destruktor, prevodilac implicitno generiše podrazumevani destruktor koji je javni i koji vrši destrukciju podataka članova i podobjekta osnovne klase pozivom njihovih destruktora.
- Destruktori se uglavnom koriste kada objekat treba da dealocira memoriju ili neke sistemske resurse koje je konstruktor alocirao; to je najčešće potrebno kada klasa sadrži

članove koji su pokazivači na pridružene dinarničke objekte koji ekskluzivno pripadaju datom objektu, pa ih je potrebno utiti prilikom utavanja tog objekta.

// Primer za konstruktor klase

```
#include <iostream.h>
class CRectangle
{
int width, height;
public:
CRectangle (int,int);
int area (void) {return (width*height);}} ;
CRectangle::CRectangle (int a, int b) { width = a;
height = b; }
main ( )
{
CRectangle rect (3,4);
CRectangle rectb (5,6);
cout << "rect area : " << rect .area ( ) << endl;
cout << "rectb area: " << rectb.area() << endl;
}
```

Izlaz:
rect area: 12
rectb area: 30

```
// Primer za konstruktor i destruktor
#include <iostream.h>
class CRectangle
{
int *width, *height;
public:
CRectangle (int,int);
~CRectangle ();
int area (void) {return (*width * *height);}
} ;
CRectangle::CRectangle (int a, int b)
{ width = new int;
height = new int;
*width = a;
*height = b; }
CRectangle::čCRectangle ()
{ delete width;
delete height; }


main ()
{
CRectangle rect (3,4), rectb (5,6);
```

```
cout << "rect area : " << rect .area()<<endl;
cout << "rectb area: " << rectb.area()<< endl;
return 0;
}
```

Izlaz:
rect area: 12
rectb area: 30
Ukazatelji klasa:

```
// Primer ukazatelja kod klasa
#include <iostream.h>
class CRectangle
{
int width, height;
public:
void set_values (int, int);
int area (void) {return (width * height);}
} ;
void CRectangle::set_values (int a, int b)
{ width = a; height = b; }
main ( )
{
CRectangle a, *b, *c;
CRectangle * d = new Crectangle[2];
b= new CRectangle;
c= &a;
a.set_values (1,2);
b->set_values (3,4);
d->set_values (5,6);
d[1].set_values (7,8);
cout << "a area: " << a.area() << endl;
cout << "*b area: " << b->area() << endl;
cout << "*c area: " << c->area() << endl;
cout << "d[0] area: " << d[0].area() << endl;
cout << "d[1] area: " << d[1].area() << endl;
return 0;
}
```

Izlaz:
a area: 2
*b area 12
*c area 2
d[0] area30
d[1] area56

Operatorske funkcije

Osnovna pravila

• U jeziku C++, pored ,,običnih" funkcija koje se eksplicitno pozivaju navođenjem identifikatora sa zagradama, postoje i operatorske funkcije.

• Operatorske funkcije su posebna vrsta funkcija koje imaju posebna imena i način pozivanja. Kao i obične funkcije, i one se mogu preklopiti za operande koji pripadaju korisničkim tipovima. Ovaj princip se naziva preklapanje operatora (engl. *operator overloading).*

• Ovaj princip omogućava da se defiu značenja operatora za korisničke tipove i formiraju izrazi sa objektima ovih tipova, na primer operacije nad kompleksnim brojevima

(ca*cb+cc-cd), matricama (ma*mb+mc-md) itd.

• Ipak, postoje neka ograničenja u preklapanju operatora. Ne mogu se:

1. preklapati operatori .,.*,::,?:i sizeof, dok svi ostali mogu;
2. redefinisati značenja operatora za ugrađene tipove podataka;
3. uvoditi novi simboli za operatore;
4. menjati osobine operatora koje su ugrađene u jezik: broj operanada, prioriteti i asocijativnost (smer grupisanja).

• Operatorske funkcije imaju imena operator @, gde je @ znak operatora. Operatorske funkcije mogu biti članice ili globalne funkcije (uglavnom prijatelji klasa) kod kojih je bar jedan argument tipa korisničke klase:

```
complex operator+ (complex c, double d) {
   return complex(c.real+d,c.imag);
}  // ovo je globalna funkcija prijatelj
complex operator** (complex c, double d) { // ovo ne moze
   // hteli smo stepenovanje
}
```

• Za korisničke tipove su implicitno definisana dva operatora: = (dodela vrednosti).i & (uzimanje adrese). Sve dok ih korisnik ne redefie, oni imaju podrazumevano značenje.

• Ukoliko klasa nema eksplicitno deklarisan operator dodele, prevodilac implicitno generiše podrazumevani operator dodele koji je javni i koji vrši operaciju dodele podobjekta osnovne klase i objekata-članova, pozivom njihovih operatora dodele (eksplicitno deklarisanih ili implicitno generisanih). Za ugrađene tipove, dodela znači prosto kopiranje vrednosti. Ako objekat sadrži člana koji je pokazivač, kopiraće se, naravno, samo taj pokazivač, a ne i pokazivana vrednost. Ovo nekad nije odgovarajuće i korisnik treba da redefie operator =.

• Vrednosti operatorskih funkcija mogu da budu bilo kog tipa, pa i void.

Bočni efekti i veze između operatora

• Bočni efekti koji postoje kod operatora za ugrađene tipove nikad se ne podrazumevaju za redefinisane operatore: ++ ne mora da menja stanje objekta, niti da znači sabiranje sa 1. Isto važi i za - - i sve operatore dodele (=, +=, - =, *= itd.).

• Operator = (i ostali operatori dodele) ne mora da menja stanje objekta. Ipak, ovakve upotrebe treba strogo izbegavati: redefinisani operator treba da ima isto ponašanje kao i za ugrađene tipove.

• Veze koje postoje između operatora za ugrađene tipove ne podrazumevaju se za redefinisane operatore. Na primer, a+=b ne mora automatski da znači a=a+b, ako je definisan operator +, već operator + = mora posebno da se defie.

• Preporučuje se da operatori koje defie korisnik imaju očekivano značenje, radi čitljivosti programa. Na primer, ako su definisani i operator + = i operator +, dobro je da a+ =b ima isti efekat kao i a=a+b. Treba izbegavati neočekivana značenja, na primer da operator- realizuje sabiranje matrica.

• Kada se defiu operatori za klasu, treba težiti da njihov skup bude kompletan. Na primer, ako su definisani operatori = i +, treba definisati i operator + =; ili, uvek treba definisati oba operatora = = i ! =, a ne samo jedan.

Operatorske funkcije kao članice i globalne funkcije

• Operatorske funkcije mogu da budu članice klasa ili (najčešće prijateljske) globalne funkcije. Ako je @ neki binarni operator (na primer +), on može da se realizuje kao funkcija članica klase X na sledeći način (argumenti se mogu prenositi i po referenci):

tip operator@ (X)
ili kao prijateljska globalna funkcija na sledeci nacin:
tip operator@ (X,X)

Nije dozvoljeno da se u programu nalaze obe ove funkcije.

• Poziv a@b se sada tumaci kao:

a. operator@ (b), za funkciju clanicu, ili operator@ (a, b), za globalnu funkciju.

• Primer:

```
class complex { public:
    complex (double re=0, double im=0) : real(r), imag(i) {}
    complex operator+(coplex c)
{ return complex(real+c.real,imag+c.imag; }
private:
  double real,imag;};
// ili, alternativno:
class complex {
 public:
complex (double re=0, double im=0) : real(r), imag(i) {}
friend complex operator+(complex,coplex);
private:
double real,imag;
} ;
```

```
complex operator+ (complex cl, complex c2) {
return complex(cl.real+c2.real,cl.imag+c2.imag);
}

void main ( ) {
complex cl (2 , 3) , c2 (3 . 4) ;
complex c3=cl+c2;  // poziva se cl. operator+(c2) ili // operator+(cl,c2)
//. . . }
```

• Razlozi za izbor jednog ili drugog načina (članica ili globalna prijateljska funkcija) isti su kao i za druge funkcije. Ako se želi da se u prethodnom primeru mogu sabrati realni i kompleksni broj, treba definisati globalnu funkciju. Ako se želi da se može izvršiti d+c, gde je d tipa double, ne može se definisati nova operatorska ,,članica klase double, jer ugrađeni tipovi nisu klase (C++ nije čisti OOjezik). Operatorska funkcija članica ,,ne dozvoljava promociju levog operanda", što znači da se neće konvertovati operand d u tip complex. Treba izabrati drugi navedeni postupak (sa prijateljskom operatorskom funkcijom).

Konstruktor kopije i operator dodele

• Inicijalizacija objekta pri njegovom nastanku i dodela vrednosti predstavljaju dve suštinski različite operacije.

• Inicijalizacija se vrši u svim slučajevima kada objekat nastaje (statički, automatski, klasni član, privremeni i dinamički). Tada se poziva konstruktor, iako se inicijalizacija obavlja preko znaka =. Ako je izraz sa desne strane znaka = istog tipa kao i objekat koji se inicijalizuje, poziva se konstruktor kopije.

• Dodelom se izvršava operatorska funkcija operator =. To se dešava kada se eksplicitno u nekom izrazu poziva ovaj operator. Ovaj operator najčešće prvo utava prethodno formirane delove objekta, pa onda formira nove, uz kopiranje delova objekta sa desne strane znaka dodele. Ova operatorska funkcija mora biti nestatička funkcija članica.

• Inicijalizacija podrazumeva da objekat još ne postoji. Dodela podrazumeva da objekat sa leve strane operatora postoji.

• Ako neka klasa sadrži pokazivač na dinamički objekat koji je pridružen svakom objektu te klase i koji je u njegovoj isključivoj nadležnosti, onda treba razmotriti potrebu da ona ima definisan i destruktor, i konstruktor kopije i operator dodele.

• Primer - klasa koja realizuje niz znakova:

```
class String { public:
String(const char*);
String(const String&);                  // konstruktor kopije
-String( );                             // destruktor
String& operator= (const String&);      // operator dodele
// ...
 private:
char *niz;};
String::String (const String &s) {
if (niz=new char [strlen(s.niz)+1]) strcpy(niz,s.niz);}
String::~String ( ) { delete [] niz; } ,
String& String:operator= (const String &s) {
```

```
if (&s!=this) {                                    // provera na s=s
    delete [] niz;                                 // prvo oslobodi staro,
    .if (niz=new char [strlen(s.niz)+1]) strcpy(niz,s.niz) ;
}                                                  // pa onda zauzmi novo
 return *this;
}
void main ( ) {
String a("Hello world!"), b=a;                     // String(const String&) ;
a=b;                                               // operator=
//. . .
}
```

Nasleđivanje

Izvedene klase

Šta je nasleđivanje i šta su izvedene klase?

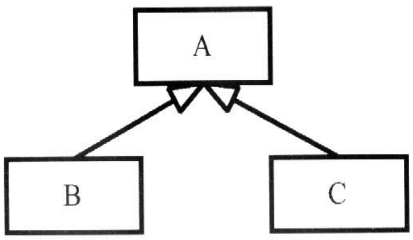
• U praksi se često sreće slučaj da je jedna klasa objekata (klasa B) podvrsta neke druge klase (klasa A). To znači da su objekti klase B ,,(specijalna) vrsta" (engl. *"a-kind-of")* objekata klase A, ili da objekti klase B imaju sve osobine klase A, i još neke, sebi svojstvene. Ovakva relacija između klasa naziva se *nasleđivanje* (engl. *inheritance):* klasa B nasleđuje klasu A.

Primeri:

1. Sisari su klasa kojoj je svojstven način razmnožavanja. Mesožderi su sisari koji se hrane mesom. Biljojedi su sisari koji se hrane biljkama. Uopšte, u živom svetu odnosi podvrsta mogu da predstavljaju relaciju nasleđivanja klasa.
2. Geometrijske figure u ravni su klasa objekata koji imaju zajedničko svojstvo - koordinate težišta. Krug je figura koja ima i dužinu poluprečnika. Kvadrat je figura koja ima i dužinu ivice.
3. Izlazni uređaji računara su klasa koja ima operacije pisanja jednog znaka. Ekran je izlazni uređaj koji ima mogućnost i crtanja, brisanja, pomeranja kursora itd.

• Relacija nasleđivanja se u programskom modelu defie u odnosu na to šta želimo da klase rade, odnosno koja svojstva i servise da imaju. Primer: da li je krug vrsta elipse, ili je elipsa vrsta kruga, ili su i krug i elipsa podvrste ovalnih figura?

• Ako je klasa B nasledila klasu A, kaže se još da je klasa A *osnovna klasa* (engl. *base class),* a klasa B *izvedena klasa* (engl. *derived class).* Može se reći i da je klasa A nadklasa (engl. *superclass),* a klasa B podklasa (engl. *subclass),* ili da je klasa A roditelj (engl. *parent),* a klasa B dete (engl. *child).* Relacija nasleđivanja se najčešće prikazuje usmerenim acikličnim grafom:



• Jezici koji podržavaju nasleđivanje nazivaju se *objektno orijentisanim (eng.0bject-0riented Programming Languages, OOPL).*
Kako se definisu izvedene klase u jeziku C++?

• Da bi se klasa izvela iz postojeće klase, nije potrebno menjati postojeću klasu, niti je ponovo prevoditi. Izvedena klasa se deklariše navođenjem reči public i naziva osnovne klase, iza znaka : (dvotačka):

```
class Base {
  int l ;
public:
  void f( ) ;
} ;
class Derived : public Base {
  int j ;
public:
  void g( ) ;
} ;
```

• Objekti izvedene klase imaju sve članove osnovne klase, i svoje posebne članove koji su navedeni u deklaraciji izvedene klase.

• Objekti izvedene klase defiu se i koriste na uobičajen način:

```
void main ( ) {
  Base b;
  Derived d;
  b . f ( ) ;
  b.g( );   // ovo, naravno, ne moze
 d.f( );    // d ima i funkciju f,
 d.g( );    // i funkciju g
}
```

• Izvedena klasa ne nasleđuje funkciju članicu operator =.

Prava pristupa

• Ključna reč public u zaglavlju deklaracije izvedene klase znači da su svi javni članovi osnovne klase ujedno i javni članovi izvedene klase.

• Privatni članovi osnovne klase uvek to i ostaju. Funkcije članice izvedene klase ne mogu da pristupaju privatnim članovima osnovne klase. Nema načina da se ,,povredi privatnost" osnovne klase (ukoliko neko nije prijatelj te klase, što je zapisano u njenoj deklaraciji), jer bi to značilo da postoji mogućnost da se probije enkapsulacija koju je zamislio projektant osnovne klase.

• Javnim članovima osnovne klase se iz funkcija članica izvedene klase pristupa neposredno, kao i sopstvenim članovima:

```
class Base {
   int pb;
public:
  int j b ;
  void put(int x) {pb=x;}
 } ;
```

```
class Derived : public Base {
   int pd;
public:
  void write(int a, int b, int c) {
     pd=a;
     jb=b;
     pb=c;     // ovo ne moze,
     put(c);   // vec mora ovako
 } };
```

• Deklaracija člana izvedene klase sakriva istoimeni član osnovne klase. Sakrivenom članu osnovne klase može da se pristupi pomoću operatora : :. Na primer, Base : : j b.

• Često postoji potreba da nekim članovima osnovne klase pristupaju funkcije članice izvedenih klasa, ali ne i korisnici klasa. To su najčešće funkcije članice koje direktno pristupaju privatnim podacima članovima. Članovi koji su dostupni samo izvedenim klasama, ali ne i korisnicima spolja, navode se iza ključne reči protected: i nazivaju se *zaštićeni članovi* (engl. *protected members).*

• Zaštićeni članovi ostaju zaštičeni i za sledeće izvedene klase pri sukcesivnom nasleđivanju. Uopšte, ne može se povećati pravo pristupa nekom članu koji je privatan, zaštićen ili javni.

```
class Base {
   int pb;
protected:
    int zb;
public:
   int jb;
//. . .
};
class Derived : public Base {
//...
 public:
void write(int x) š
jb=zb=x;        // moze da pristupi javnom i zasticenom clanu,
 pb=x;          // ali ne i privatnom: greska!
}
 {;
void f ( ) }
Base b;
b.zb=5; // odavde ne moze da se pristupa zasticenom clanu
{
```

Konstruktori i destruktori izvedenih klasa

• Prilikom nastanka objekta izvedene klase, poziva se konstruktor te klase, ali se pre izvršavanja njegovog tela poziva konstruktor osnovne klase. U zaglavlju definicije konstruktora izvedene klase, u listi inicijalizatora, moguće je navesti i inicijalizator osnovne klase (argumente poziva konstruktora osnovne klase). To se radi navođenjem imena osnovne klase i argumenata poziva konstruktora osnovne klase:

```
class Base {
   int bi;
   //...
 public:
   Base(int);    // konstruktor osnovne klase
   //...
} ;
Base::Base (int i) : bi(i) {/*...*/}
class Derived : public Base {
  int di ;
  / / . . .
public:
Derived(int) ;
//. . .
} ;

Derived::derived (int i) : Base (i),di(i+1) {/*...*/}
```

• Pri inicijalizaciji objekta izvedene klase redosled poziva konstruktora je sledeći:

1. Inicijalizuje se podobjekat osnovne klase, pozivom konstruktora osnovne klase.
2. Inicijalizuju se podaci članovi, eventualno pozivom njihovih konstruktora, po redosledu njihovog deklarisanja.
3. Izvršava se telo konstruktora izvedene klase.

• Pri utavanju objekta, redosled poziva destruktora uvekje obratan.

```
class XX {
  //. . .
public:
  XX( ) {cout<<"Konstruktor klase XX. \n" ; }
 čXX( ) {cout<<"Destruktor klase XX. \n";}
} ;
class Base {
   //...
 public:
    Base( ) {cout<<"Konstruktor osnovne klase.\n";}
   ~Base( ) {cout<<"Destruktor osnovne klase.\n";}
//. ..
};
class Derived : public Base {
   XX xx;
   //.. .
 public:
    Derived( ) {cout<<"Konstruktor izvedene klase.\n";}
    čDerived( ) {cout<< "Destruktor izvedene klase.\n";}
//.. .
};
void main ( ) {
Derived d;
}
```

```
/* Izlaz će biti;
Konstruktor osnovne klase.
 Konstruktor klase XX.
Konstruktor izvedene klase.
 Destruktor izvedene klase.
Destruktor klase XX.
 Destruktor osnovne klase.
*/
```

Polimorfizam

Sta je polimorfizam ?

• Pretpostavimo da smo projektovali klasu geometrijskih figura sa namerom da sve figure imaju funkciju crtaj ( ) kao članicu. Iz ove klase izveli smo klase kruga, kvadrata, trougla itd. Naravno, svaka izvedena klasa treba da realizuje funkciju crtanja na sebi svojstven način (krug se sasvim drugačije crta od trougla). Sada nam je potrebno da u nekom delu programa iscrtamo sve figure koje se nalaze na našem crtežu. Ovim figurama pristupamo preko niza pokazivača tipa Figura*. C++ omogućava da figure iscrtamo prostim navodenjem:

```
void crtanje ( ) {
   for (int i=0; i <brojFigura; i++)
      nizFigura [ i]->crtaj ( );
}
```

• Iako se u ovom nizu mogu naći različite figure (krugovi, trouglovi itd.), mi im pristupamo kao figurama jer sve vrste figura imaju zajedničku osobinu ,,da mogu da se nacrtaju". Ipak, svaka od figura svoj zadatak ispuniće onako kako joj to i priliči, odnosno svaki objekat će ,,prepoznati" kojoj izvedenoj klasi pripada, bez obzira na to što mu se obraćamo ,,uopšteno", kao objektu osnovne klase. To je posledica naše pretpostavke da su i krug i kvadrat i trougao vrste figura.

• Svojstvo da svaki objekat izvedene klase, čak i kada mu se pristupa kao objektu osnovne klase, izvršava metod tačno onako kako je to definisano u njegovoj izvedenoj klasi, naziva se *polimorfizam* (engl. *polymorphism).*

Virtuelne funkcije

• Funkcije članice osnovne klase koje se u izvedenim klasama mogu realizovati specifično za svaku izvedenu klasu, nazivaju se *virtuelne funkcije* (engl. *virtualfunctions).*

• Virtuelna funkcija se u osnovnoj klasi deklariše pomoću ključne reči virtual na početku deklaracije. Prilikom definisanja virtuelnih funkcija u izvedenim klasama ne mora se stavljati reč virtual, ali se to preporučuje radi povećanja čitljivosti i razumljivosti programa.

• Prilikom poziva odaziva se ona funkcija koja pripada klasi kojoj i objekat koji prima poziv.

```
class ClanBiblioteke {
public:
virtual void platiClanarinu ( )     // virtuelna funkcija
   { r--clanarina; }
//...
 private:
```

```
Racun r ;
//...
 } ;
class PocasniClan : public ClanBiblioteke{
  public:
virtual void platiClanarinu ( ) {}
//. . .
 } ;


void main ( ) {
ClanBiblioteke *clanovi[100] ;
//.. .
for (int i=0; i<brojClanova; i++)
clanovi[i] ->platiClanarinu( ) ;
//...
}
```

• Virtuelna funkcija osnovne klase ne mora da se redefie u svakoj izvedenoj klasi. U izvedenoj klasi u kojoj virtuelna funkcija nije definisana, važi značenje te virtuelne funkcije iz osnovne klase.

• Deklaracija neke virtuelne funkcije u svakoj izvedenoj klasi mora da se u potpunosti slaže sa deklaracijom te funkcije u osnovnoj klasi (broj i tipovi argumenata, kao i tip rezultata).

• Ako se u izvedenoj klasi deklariše neka funkcija koja ima isto ime kao i virtuelna funkcija iz osnovne klase, ali različit broj i/ili tipove argumenata, onda ona sakriva (a ne redefie) *sve* ostale istoimene funkcije iz osnovne klase.

Nizovi i izvedene klase

• Objekat izvedene klase je vrsta objekta osnovne klase. Međutim, niz objekata izvedene klase nije vrsta niza objekata osnovne klase. Uopšte, neka kolekcija objekata izvedene klase nije vrsta kolekcije objekata osnovne klase.

• Na primer, iako je automobil vrsta vozila, parking za automobile nije i parking za sve vrste vozila, jer na parking za automobile ne mogu da stanu i kamioni (koji su takođe vozila). Ili, ako korisnik neke funkcije prosledi toj funkciji korpu banana (banana je vrsta voća), ne bi valjalo da mu ta funkcija vrati korpu u kojoj je jedna šljiva (koja je takođe vrsta voća), smatrajući da je korpa banana isto što i korpa bilo kakvog voća.

• Ako se računa sa nasleđivanjem, u programu ne treba koristiti nizove objekata, već nizove pokazivača na objekte. Ako se formira niz objekata izvedene klase i on prenese kao niz objekata osnovne klase (što, po prethodno rečenom, semantički nije ispravno, ali je moguće), može doći do greške:

```
class Base {
 public: int bi;
} ;
class Derived : public Base {
public; int di;
};
```

```
void f (Base *b) { cout«b [2] . bi ; }
void main ( ) {
Derived d [5] ;
 D[2].bi=77;
 f (d) ;          // nece se ispisati 77
}
```

• U prethodnom primeru, funkcija f smatra da je dobila niz objekata osnovne klase koji su kraći (nemaju sve članove) od objekata izvedene klase. Kada joj se prosledi niz objekata izvedene klase (koji su duži), funkcija nema načina da odredi da se niz sastoji samo od objekata izvedene klase. Rezultat je, u opštem slučaju, neodređen.

• Pored navedene greške, fizički nije moguće direktno smeštati objekte izvedene klase u niz objekata osnovne klase. Objekti izvedene klase su duži, a za svaki element niza je odvojen samo prostor koji je dovoljan za smeštanje objekta osnovne klase.

• Zbog svega što je rečeno, kolekcije (nizove) objekata treba realizovati kao nizove pokazivača na objekte:

```
void f (Base **b, int i) { cout<<b [i] - >bi; }
void main ( ) {
  Base bl,b2;
  Derived dl,d2,d3 ;
  Base *b[5];                    // b se moze konvertovati u tip
                                 // Base**
  b[0]=&dl; b[1]=&b1; b[2]=&d2;  // konverzije Derived* u Base*
  b[3]=&d3; b[4]=&b2;
 d2.bi=77 ;
 f(b,2);                         // ispisace se 77
}
```

• Kako je objekat izvedene klase vrsta objekta osnovne klase. C++ dozvoljava implicitnu konverziju pokazivača Derived* u Base* (prethodni primer). Zbog logičkog pravila da niz objekata izvedene klase nije vrsta niza objekata osnovne klase, a kako se nizovi ispravno realizuju pomoću nizova pokazivača, C++ ne dozvoljava implicitnu konverziju pokazivača Derived* * (u koji se može konvertovati tip niza pokazivača na objekte izvedene klase) u Base* * (u koji se može konvertovati tip niza pokazivača na objekte osnovne klase). Za prethodni primer nije dozvoljeno:

```
void main ( ) {
   Derived *d[5];  // d je tipa Derived**
   //...
  f(d,2);     // nije dozvoljena konverzija Derived** u Base**
}
```

Apstraktne klase

• Čest je slučaj da neka osnovna klasa nema nijedan konkretan primerak (objekat), već samo predstavlja generalizaciju izvedenih klasa.

• Na primer, svi izlazni, znakovno orijentisani uređaji računara imaju funkciju za ispis jednog znaka, ali se u osnovnoj klasi izlaznog uređaja ne može definisati način ispisa tog znaka, već je to specifično za svaki uređaj posebno. Ili, ako iz osnovne klase osoba izvedemo dve klase muškaraca i žena, onda klasa osoba ne može imati primerke jer ne postoji osoba koja nije ni muškog ni ženskog pola.

• Klasa koja nema instance (objekte), već su iz nje samo izvedene druge klase, naziva se *apstraktna klasa* (engl. *abstract class).*

• U jeziku C++, apstraktna klasa sadrži bar jednu virtuelnu funkciju članicu koja je u njoj samo deklarisana, ali ne i definisana. Definicije te funkcije daće izvedene klase. Ovakva virtuelna funkcija naziva se čistom virtuelnom funkcijom. Njena deklaracija u osnovnoj klasi završava se sa = 0:

```
class OCharDevice {

 public:
virtual int put (char) =0;    // cista virtuelna funkcija
//...
} ;
```

• U jeziku C++ apstraktna klasa je klasa koja sadrži bar jednu čistu virtuelnu funkciju. Ovakva klasa ne može imati instance, već se iz nje izvode druge klase. Ako se u izvedenoj klasi ne navede definicija neke čiste virtuelne funkcije iz osnovne klase, i ova izvedena klasa je takođe apstraktna.

• Pokazivači i reference na apstraktnu klasu mogu da se defiu, ali oni ukazuju na objekte izvedenih konkretnih (neapstraktnih) klasa.

Višestruko nasleđivanje

Šta je višestruko nasleđivanje?

• Nekad postoji potreba da izvedena klasa ima osobine više osnovnih klasa istovremeno. Tada se radi o *višestrukom nasleđivanju* (engl. *multiple inheritance).*

• Na primer, motocikl sa prikolicom je vrsta motocikla, ali i vrsta vozila sa tri točka. Pri tom, motocikl nije vrsta vozila sa tri točka, niti je vozilo sa tri točka vrsta motocikla, već su ovo dve različite klase. Klasa motocikala sa prikolicom nasleđuje obe ove klase.

• Klasa se deklariše kao naslednik više klasa tako što se u zaglavlju deklaracije, iza znaka :, navode osnovne klase razdvojene zarezima. Ispred svake osnovne klase treba da stoji reč public. Na primer:

```
class Derived : public Basel, public Base2, public Base3 {
 //...
};
```

• Sva navedena pravila o nasleđenim članovima važe i ovde. Konstruktori svih osnovnih klasa pozivaju se pre poziva konstruktora članova izvedene klase i pre izvršavanja tela konstruktora izvedene klase. Konstruktori osnovnih klasa pozivaju se po redosledu deklarisanja tih klasa u zaglavlju izvedene klase. Destruktori osnovnih klasa izvršavaju se na kraju, posle izvršavanja tela destruktora izvedene klase i destruktora članova.

Virtuelne osnovne klase

• Posmatrajmo sledeći primer:

```
class B {/*...*/};
class X : public B {/*...*/};
class Y : public B {/*...*/};
class Z : public X, public Y {/*...*/};
```

• U ovom primeru klase X i Y nasleđuju klasu B, a klasa Z klase X i Y. Klasa Z ima sve što imaju X i Y. Kako svaka od klasa X i Y ima po jedan primerak članova klase B, to će klasa Z imati dva skupa članova klase B. Njih je moguće razlikovati pomoću operatora : ; (npr.z .X: : i ili z .Y: : i).

• Ako ovo nije potrebno, klasu B treba deklarisati kao virtuelnu osnovnu klasu:

```
class B {/*...*/};
class X : virtual public B {/*...*/};
class Y : virtual public B {/*...*/};
class Z : public X, public Y {/*...*/};
```

• Sada klasa z ima samo jedan skup članova klase B.

• Ako neka izvedena klasa ima virtuelne i nevirtuelne osnovne klase, onda se konstruktori virtuelnih osnovnih klasa pozivaju pre konstruktora nevirtuelnih osnovnih klasa, po redosledu deklarisanja. Svi konstruktori osnovnih klasa se, naravno, pozivaju pre konstruktora članova i konstruktora izvedene klase.

Uvod u objektno modelovanje

Apstraktni tipovi podataka

• Apstraktni tipovi podataka predstavljaju realizacije struktura podataka sa pridruženim protokolima (operacijama i definisanim načinom i redosledom pozivanja tih operacija). Na primer, *red* (engl. *queue)* je struktura elemenata koja ima operacije stavljanja i uzimanja elemenata u strukturu, pri čemu se elementi uzimaju po istom redosledu po kom su stavljeni.

• Kada se realizuju strukture podataka (apstraktni tipovi podataka), najčešće nije bitno koji je tip elementa strukture, važan je samo skup operacija. Načini realizacija tih operacija ne zavise od tipa elementa, već samo od tipa strukture.

• Za realizaciju apstraktnih tipova podataka kod kojih tip nije bitan, u jeziku C++ postoje *šabloni* (engl. *templates).* Šablon klase predstavlja definiciju čitavog skupa klasa koje se razlikuju samo po tipu elementa i, eventualno, po dimenzijama. Šabloni klasa se ponekad nazivaju i generičkim klasama.

• Konkretna klasa generisana iz šablona dobija se navođenjem stvarnog tipa elementa.

• Formalni argumenti šablona zadaju se u zaglavlju šablona:

```
template <class T>
class Queue {
 public:
Queue ( );
~Queue ( ) ;
void put (const T&) ;
T    get ();
//...
} ;
```

• Konkretna generisana klasa dobija se samo navođenjem imena šablona, uz definisanje stvarnih argumenata šablona. Stvarni argumenti šablona su tipovi i, eventualno, celobrojne dimenzije. Konkretna klasa se generiše na mestu navođenja, u fazi prevođenja. Na primer, red događaja može se kreirati na sledeći način:

```
class Event;
Queue<Event*> que ;
que . put (e) ;
if (que.get( )->isUrgent( ) )...
```

• Generisanje je samo stvar automatskog generisanja parametrizovanog koda istog oblika, a nema nikakve veze sa izvršavanjem. Generisane klase nemaju nikakve posebne međusobne veze.

Projektovanje apstraktnih tipova podataka

• Apstraktni tipovi podataka (strukture podataka) su veoma često korišćeni elementi svakog programa. Projektovanje biblioteke klasa koje realizuju standardne strukture podataka veoma je delikatan posao. Ovde će biti prikazana konstrukcija dve česte linearne strukture podataka:

1. Kolekcija (engl. *collection)* je linearna, neuređena struktura elemenata koja ima samo operacije stavljanja elementa i izbacivanja datog elementa iz strukture. Redosled elemenata nije bitan, a elementi se mogu i ponavljati.

2. Red (engl. *queue)* je linearna, uređena struktura elemenata. Elementi su uređeni po redosledu stavljanja. Operacija uzimanja vraća element koji je prvi stavljen u strukturu.

• Važan koncept pridružen strukturama je pojam *iteratora* (engl. *iterator).* Iterator je objekat pridružen linearnoj strukturi koji služi za pristup redom elementima strukture. Iterator ima operacije za postavljanje na početak strukture, za pomeranje na sledeći element strukture, za pristup do tekućeg elementa na koji ukazuje i operaciju za ispitivanje da li je došao do kraja strukture. Za svaku strukturu može se napraviti proizvoljno mnogo objekata iteratora i svaki od njih pamti svoju poziciju.

• Suština koncepta iteratora je da se obezbedi sekvencijalni pristup do elemenata agregatne strukture, bez izlaganja njene inteme realizacije. S druge strane, loše je opterećivati sam interfejs date strukture operacijama koje obezbeđuju takav pristup. Zato se obezbeđuje posebna klasa iteratora pridružena datoj strukturi, koja je odgovorna za obilazak strukture, poznaje njenu intemu predstavu, i za koju se onda može napraviti proizvoljan broj instanci koje nezavisno iteriraju. Ovakvo rešenje predstavlja projektni obrazac (engl. *design pattern) Iterator* (ili *Cursor).*

• Pri realizaciji biblioteke klasa za strukture podataka, bitno je razlikovati *protokol* strukture koji defie njenu semantiku, od njene *implementacije.*

• Protokol strukture određuje značenje njenih operacija, potreban način ili redosled pozivanja itd.

• Implementacija se odnosi na način smeštanja elemenata u memoriju, organizaciju njihove veze itd. Važan aspekt implementacije je da li je ona ograničena ili nije. Ograničena realizacija se oslanja na fiksno dimenzionisani niz elemenata, dok se neograničena realizacija odnosi na dinamičku strukturu (najčešće listu).

• Na primer, protokol reda izgleda otprilike ovako:

```
template <class T>
class Queue {
 public:
virtual IteratorQueue<T>* createIterator() const =0;
virtual void put     (const T&) =0;
virtual T    get       ( ) =0;
virtual void clear    ( ) =0;
virtual const T& first    ( ) const=0;
virtual int      isEmpty   ( ) const=0;
virtual int      isFull      ( ) const=0;
virtual int      length      ( ) const=0;
virtual int      location (const T& ) const=0;
```

• Da bi se korisniku obezbedile obe realizacije (ograničena i neograničena), postoje dve klase izvedene iz navedene apstraktne klase koja defie interfejs reda. Jedna od njih podrazumeva ograničenu (engl. *bounded)* realizaciju, a druga neograničenu (engl. *unbounded).* Na primer:

```
template <class T, int N>
class OueueB : public Queue<T> {
public:
QueueB ( ) {}
 QueueB (const Queue<T>&) ;
virtual ~QueueB ( ) {}
Queue<T>& operator= (const Queue<T>&) ;
virtual IteratorQueue<T>* createIterator( ) const;
virtual void put    (const T& t) ;
virtual T    get    ( ) ;
virtual void clear  ( )        ;
virtual const T& first    ( ) const;
virtual int      isEmpty   ( ) const;
virtual int      isFull      ( ) const;
virtual int      length      ( ) const;
virtual int      location (const T& t) const;
};
```

• Treba obratiti pažnju na način kreiranja iteratora. Korisniku je dovoljan samo opšti, zajednički interfejs iteratora. Korisnik ne treba ta da zna o specifičnostima realizacije iteratora i njegovoj vezi sa konkretnom izvedenom klasom reda. Zato je definisana osnovna apstraktna klasa iteratora, iz koje su izvedene klase za iteratore vezane za dve posebne realizacije reda:

```
template <class T>
class IteratorQueue {
public:
virtual čIteratorQueue ( ) {}
virtual void reset( ) =0;
virtual int next ( ) =0;
vlrtual int isDone( ) const =0;
virtual const T* currentltem( ) const =0;
};
```

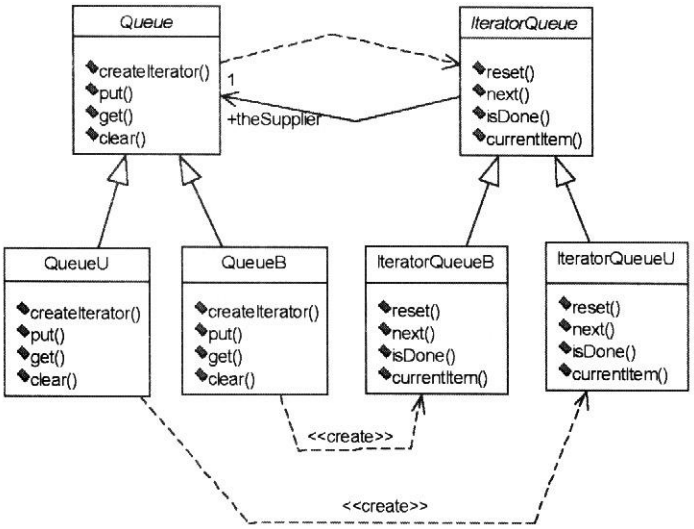
• Izvedene klase reda praviće posebne, njima specifične iteratore koji se uklapaju u zajednički interfejs iteratora:

```
template<class T, int N>
IteratorQueue<T>* QueueB<T,N>::createIterator( ) const {
 return new IteratorQueue B<T, N> (this) ;
}
```

• Suština ovog rešenja je da se korisnici ovog podsistema oslanjaju samo na apstraktne interfejse Queue i IteratorQueue, odnosno da ne znaju za specifičnosti konkretnih implementacija. čak i samo pravljenje iteratora obavlja se preko apstraktnog interfejsa, tj. operasije *createiterator* , koju defiu konkretne izvedene klase QueueB i QueueU, tako što prave konkretne specifične iteratore. Ovakav pristup znatno povećava fleksibilnost softvera.

• Ovo rešenje, tj. operacija createIterator ( ) predstavlja projektni obrazac (engl. *designpattern) Factory Method* (ili *Virtual Constructor),* koji se sastoji u obezbeđivanju interfejsa za pravljenje objekta, pri čemu konkretne izvedene klase odlučuju koji konkretni objekat da naprave.

• Relacije između opisanih klasa prikazane su na sledećem dijagramu:



• Sama realizacija ograničene i neograničene strukture oslanja se na dve klase koje imaju sledeće interfejse:

```
template <class T>
class Unbounded {
public:
   Unbounded ( ) ;
   Unbounded (const Unbounded<T>&) ;
  -Unbounded  ( ) ;
Unbounded<T>& operator= (const Unbounded<T>&);
void append (const T&) ;
void insert (const T&, int at=0);
void remove (const T&) ;
void remove (int at=0);
```

```
void clear  ( ) ;
Int       isEmpty  ( ) const;
int       isFull       ( ) const;
int       length       ( ) const;
const T& first    ( ) const;
const T& last     ( ) const;
const T& itemAt   (Int at) const;
T&       itemAt   (int at) ;
int      location (const T&) const;
} ;

template <class T, int N>
 class Bounded {
 public:
Bounded ( ) ;
Bounded (const Bounded<T,N>&) ;
~Bounded ( ) ;
Bounded<T,N>& operator = (const Bounded<T, N>&) ;
void append (const T&) ;
vold insert (const T&, int at=0);
void remove (const T&) ;
void remove (int at=0) ;

void clear  ( );
int;    isEmpty     ( )   const.;
int      isFull         ( ) const;
int      length         ( ) const;
const T& first          ( ) const;
const T& last           ( ) const;
const T& iternAt    (int at) const;
T&     itemAt         (int at) ;
int      location       (const T&) const;
};
```

• Definisana struktura može se koristiti kao u sledećem primeru:

```
class Event {
//...
 };
typedef OueueB<Event*,MAXEV> EventOueue;
typedef IteratorQueue<Event*> Iterator;
//...
 EventQueue que;
que.put(e) ;
Iterator* it=que.createlterator( );
for (; !it->isDone ( ); it->next( ))
     (*it- >current Item ( ) ) ->handle ( ) ;
delete it;
```

• Promena orijentacije na ograničeni red veoma je jednostavna. Ako se želi neograničeni red, dovoljno je promeniti samo:

typedef QueueU<Event*> EventQueue;

Modelovanje strukture - klase i osnovne relacije između klasa

• Klasa je osnovna jedinica strukturnog modela sistema.

• Klasa je veoma retko izolovana. Ona dobija smisao samo uz druge klase sa kojima je u relaciji. Osnovne relacije izmedu klasa su: asocijacija, zavisnost i nasleđivanje.

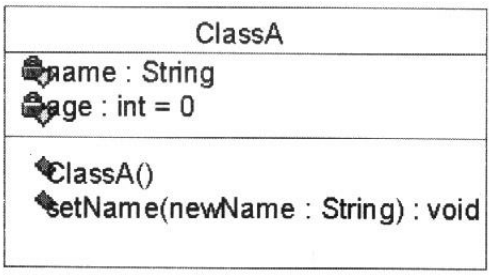
Klasa, atributi i operacije

• Klasom se modeluje apstrakcija. Klasa je opis skupa objekata koji dele iste atribute, operacije, relacije i semantiku.

• Atribut je imenovano svojstvo entiteta. Njime se opisuje opseg vrednosti koje instance tog svojstva mogu da imaju.

• Operacija je implementacija usluge koja se može zatražiti od bilo kog objekta klase da bi se uticalo na ponašanje.

• Klasa se prikazuje pravougaonim simbolom u kome mogu postojati posebni odeljci za ime klase, atribute i operacije:



Asocijacija

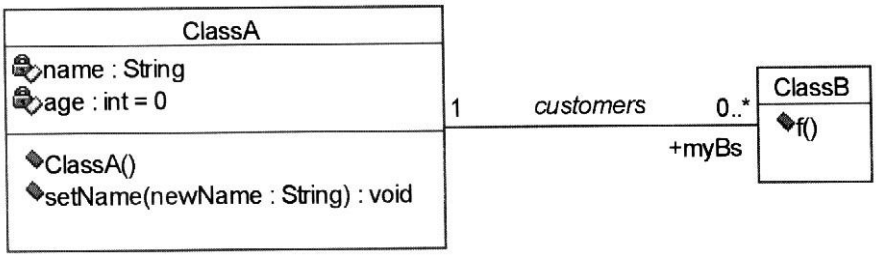
• Asocijacija (pridruživanje, engl. *association)* je relacija između klasa čiji su objekti na neki način strukturno povezani. Ta veza između objekata klasa postoji određeno duže vreme, a ne samo tokom trajanja izvršavanja operacije jednog objekta koju poziva drugi objekat. Instanca asocijacije naziva se *vezom* (engl. *link)* - postoji između objekata datih klasa.

• Asocijacija se predstavlja punom linijom koja povezuje dve klase. Asocijacija može imati ime koje opisuje njeno značenje. Svaka strana u asocijaciji ima svoju *ulogu* (engl. *role)* koja se može naznačiti na strani date klase.

• Na svakoj strani asocijacije može se definisati kardinalnost (multiplikativnost, engl. *multiplicity)* pomoću sledećih oznaka:

1 tačno jedan
* proizvoljno mnogo
1..* jedan i više
0..1 nula ili jedan
3..7 zadati opseg
i slično.

• Primer:



• Druga posebna karakteristika svake strane asocijacije je *navigabilnost* (engl. *navigability):* sposobnost da se sa te strane (od objekta sa jedne strane veze) dospe do druge strane (do objekta sa druge strane veze). Prema ovom svojstvu, asocijacija može biti simetrična ili asimetrična.

• Primer: prikazana asocijacija se na jeziku C++ na strani klase A realizuje na sledeći način, ukoliko postoji mogućnost navigacije prema klasi B:

```
class B;
class A {
 public:
  //. . .
  // Funkcije za uspostavljanje veza asocijacije:
  void insert (B* b) š myBs.add(b); }
  void remove (B* b) { myBs.remove(b); }
private;
CollectionU<B*> myBs;
};
```

• Na strani klase **B,** ukoliko postoji mogućnost navigacije prema klasi A, ova veza se uspostavlja pomoću konstruktora klase B (jer je kardinalnost tačno 1):

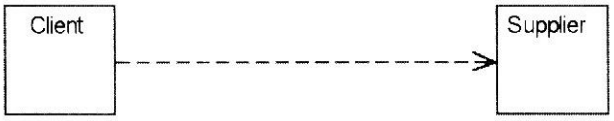
```
class A;
class B {
public :
  B (A* a) š myA=a ; }
//...
private:
 A* myA;
} ;
```

• Kod navedene realizacije na jeziku C++, potrebno je obratiti pažnju na sledeće. U deklaraciji interfejsa klase B nije potrebna potpuna definicija klase A, već samo prosta deklaracija class A;, jer je objekat B vezan za objekat klase A preko pokazivača. Samo u implementaciji neke složenije operacije potrebna je potpuna definicija klase A. Kako se implementacije ovih funkcija nalaze u modulu B . cpp, samo ovaj modul zavisi od modula sa interfejsom klase A. dok modul B. h ne zavisi. Na ovaj način se drastično smanjuju zavisnosti između modula i vreme prevođenja.

Zavisnost

• Relacija zavisnosti (engl. *dependency)* postoji ako klasa A na neki način koristi usluge klase B. To može biti npr. odnos klijent-server (klasa A poziva funkcije klase B) ili odnos instancijalizacije (klasa A pravi objekte klase B).

• Za realizaciju ove relacije između klase A i B potrebno je da interfejs ili implementacija klase A ,,zna" za definiciju klase B. Tako je klasa A zavisna od klase B.
• Oznaka:



• Značenje relacije može da se navede kao stereotip relacije na dijagramu, npr. < <call > >
ili <<create>>.

• Ako klasa Client koristi usluge klase Supplier tako što poziva operacije objekata ove klase (odnos klijent-server), onda ona tipično ,,vidi" ove objekte kao argumente svojih operacija. U ovom slučaju, za implementaciju na jeziku C++, interfejsu klase Client nije potrebna definicija klase Supplier, već samo njenoj implementaciji:

```
class Supplier;
class Client {
 public:
   //. . .
    void aFunction (Supplier*);
} ;
// Implementacija:
void Client::aFunction (Supplier* s) {
   //. . .
  s->doSomething( ) ;
}
```

• Pri implementaciji na jeziku C++, ako je potrebno dobiti notaciju prenosa po vrednosti, a zadržati navedenu pogodnost slabije zavisnosti između modula, onda se argument prenosi preko reference na konstantu:

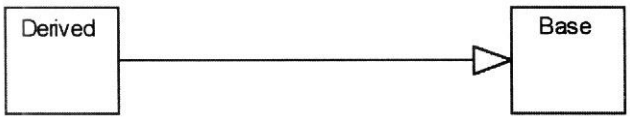
```
void Client: :aFunction (const Supplier& s) {
s.doSomething ( ) ;
}
```

• Ako klasa Client instancijalizuje klasu Supplier, onda je realizacija nalik na:

```
Supplier* Client : : createSuppllier (/*some _arguments*/) {
return new Supplier(/*some_ arguments*/);
}
```

Nasleđivanje (generalizacija/specijalizacija)

• Nasleđivanje (engl. *inheritance)* predstavlja relaciju generalizacije, odnosno specijalizacije, zavisno u kom smeru se posmatra. Oznaka:



• Relacija generalizacije/specijalizacije (engl. *generalization/specialization)* predstavlja relaciju koja ima dve važne semantičke manifestacije:

(a) Nasleđivanje: nasleđena (izvedena) klasa implicitno poseduje (nasleđuje) sve atribute, operacije i asocijacije osnovne klase (važi i tranzitivnost).

(b) Supstitucija (engl. *substitution):* objekat (instanca) izvedene klase može se naći svugde gde se očekuje objekat osnovne klase (važi i tranzitivnost). **Za** struktumi aspekt sistema ovo pravilo ima sledeću bitnu manifestaciju: ako u nekoj asocijaciji učestvuje osnovna klasa, onda u nekoj vezi kao instanci te asocijacije mogu učestvovati objekti svih klasa izvedenih (neposredno ili posredno) iz te klase.

• Realizacija:

class Derived : public Base //...

• Zbog ovako definisane semantike osnovnih relacija između klasa (prvenstveno asocijacije i nasleđivanja), softverski sistemi (aplikacije) koji su modelovani objektno imaju jedno opšte svojstvo: njihova struktura u vreme izvršavanja može se apstraktno posmatrati kao *tipizirani graf,* jer se sastoji iz objekata (instanci klasa) povezanih vezama (instancama asocijacija). Dakle, objekti predstavljaju čvorove, a veze grane jednog grafa. Pri tom, objekti kao čvorovi grafa imaju svoje tipove (to su klase čije su ovo instance), kao i veze koje su instance odgovarajućih asocijacija. Na stranama svake veze koja je instanca neke asocijacije nalaze se instance onih klasa koje povezuje ta asocijacija, ili klasa izvedenih iz njih (uključujući i tranzitivnost nasleđivanja).

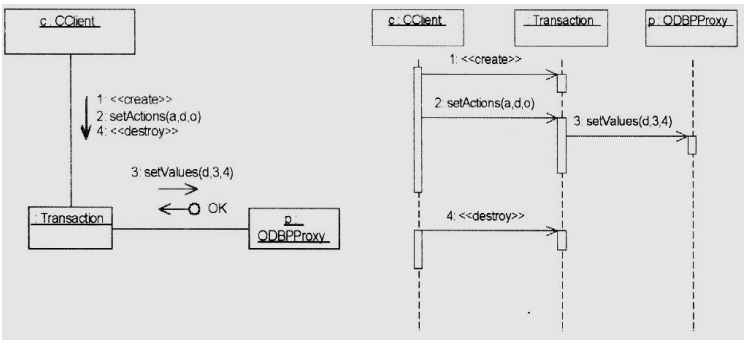
Modelovanje ponašanja - interakcije

• Ponašanje sistema se uglavnom realizuje interakcijama između objekata. *Interakcija* (engl. *interaction)* je ponašanje koje se sastoji od skupa poruka koje se razmenjuju između objekata u nekom kontekstu da bi se ispunila neka svrha. *Poruka* (engl. *message)* je specifikacija komunikacije između objekata koja prenosi informaciju sa očekivanjem da će se pokrenuti neka aktivnost.

• *Dijagram interakcije* (engl. *interaction diagram)* opisuje ponašanje i odnosi se na neki kontekst - deo sistema čije ponašanje opisuje. To može biti neka funkcionalnost sistema ili neka operacija klase. Postoje dve vrste dijagrama interakcije: dijagram kolaboracije i dijagram sekvence. Ovi dijagrami predstavljaju dva različita pogleda na semantički istu stvar (istu interakciju), samo što naglašavaju različite aspekte te interakcije.

• *Dijagram kolaboracije* (engl. *collaboration diagram)* je dijagram interakcije koji naglašava *strukturnu organizaciju objekata koji razmenjuju poruke. Grafički, ovaj dijagram je* skup čvorova koji predstavljaju objekte i skup linija koji ih povezuju i preko kojih idu poruke.

• *Dijagram sekvence* (engl. *sequence diagram)* je dijagram interakcije koji naglašava vremenski redosled poruka. Ovaj dijagram prikazuje objekte poređane po x osi, dok se poruke ređaju kao horizontalne linije po y osi, pri čemu vreme raste nadole.



PRAKTIKUM
ZA
LABORATORIJSKE VEŽBE IZ

PROGRAMSKIH JEZIKA 2

PRVA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

1. Programski jezik C++ JE OBJEKTNO ORIJENTISANA NADGRADNJA JEZIKA C. C++ kombinuje moć i ekonomičnost izraza jezika C sa novim dostignućima objektno orijentisanog programiranja. Ovaj izuzetan jezik brzo sebi krči put u oblasti profesionalnog programiranja. Posebno je značajan njegov prodor u razvoj aplikacija za Microsoftov Windows.

Osnovne prednosti jezika C++ su poboljšana produktivnost i pouzdanost. I jedno i drugo je proisteklo iz ideje *objekta* kao, u suštini, klasične strukture podataka koja se ponaša na odreden način. Objekti olakšavaju ponovno korišćenje koda iz dva razloga: bogatiji su i kompletniji od klasičnih biblioteka potprograma i mogu se proširivati. Viosual C++ i Borland-ov C++ 3.1 su na tržištu vodeći C++ prevodioci za PC platformu. Dokje Zortech C++ bio prvi pravi prevodilac za C++ na PC računaru, Borland je prvi C++ prevodilac koji je postigao sledeće rezultate:

- Veliku zastupljenost na PC računarima
- Mo`e da radi i sa DOS-om i sa Windows-om jer ima obimne biblioteke klasa
- Dostupan je u jeftinijoj verziji, pogodnoj za ku}nu i {kolsku upotrebu

Najvažniji alati objektno orijentisanog programiranja :

Tri najvažnija koncepta objektno orijentisanog programiranja su apstrakcija (engl. *abstraction),* nasleđivanje (engl. *inheritance)* i polimorfizam (engl. *polymorphism).* Ova tri osnovna pojma objektno orijentisanog programiranja koje smo sada uveli pojavljivaće se kroz ceo naš dalji rad i od vitalnog su značaja za rad sa jezikom C++.

U objektno orijentisanom programiranju softver projektujemo korišćenjem komponenti koje se zovu objekti (engl. *objects).* Objekti se sastoje od podataka članova i grupe rutina koje upravljaju podacima, a zovu se funkcije članice (engl. *member functions).* Kombinovanje podataka sa rutinama čini objekte robusnim i kompletnim, tako da mogu da vrše svoju ulogu u različitim okruženjima. Međutim, ne treba zaboraviti i tri vrlo važna alata kao delova objektno orijentisane tehnologije jezika C++. To su konstruktori (engl. *constructors*) koji omogućavaju da se konstruiše kako se kreiraju objekti, šabloni (engl. *templets*) koji omogućavaju kreiranje klasa koje imaju isti kod ali za različite tipove, i prijatelji (engl. *friends*) koji ublažavaju pristupna pravila jezika C++.

Smernice za objektno orijentisano projektovanje

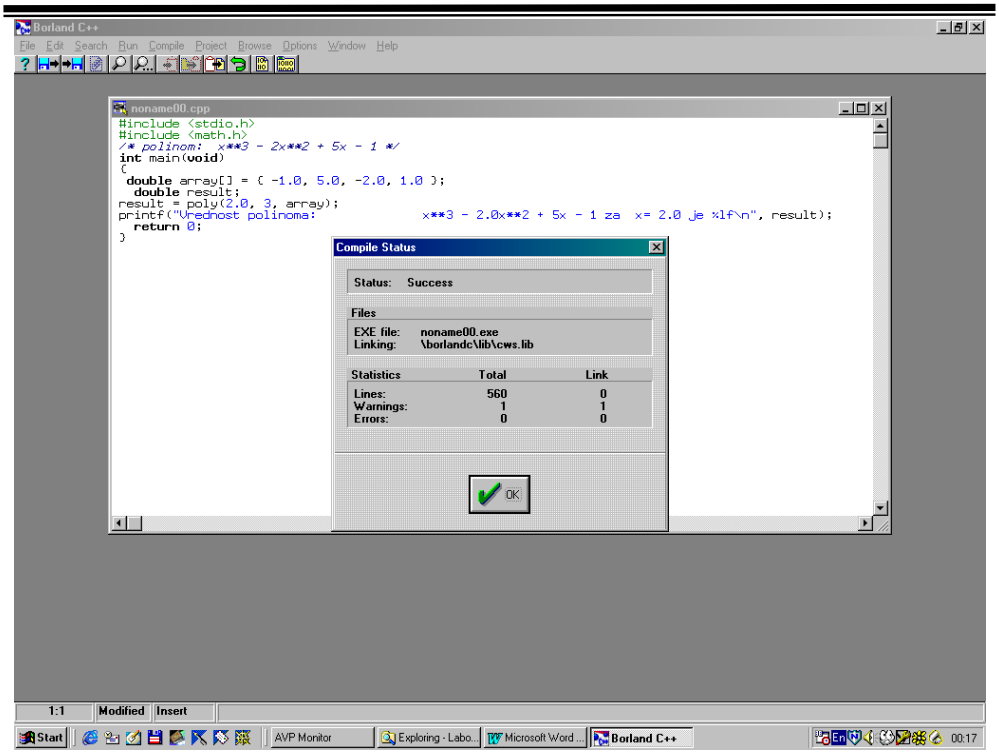
Suština objektno orijentisanog projektovanja je da se začne grupa objekata koja može da se realizuje kao grupa klasa i koja rešava određeni problem. Većim delom veština objektno orijentisanog projektovanja stiče se iskustvom kroz rad sa objektima, i dobrim poznavanjem oblasti problema. Nemam jednostavan odgovor za očigledno težak problem, ali imam kratku listu sugestija koje mogu da vam skrate vreme potrebno za učenje:

- Objekti treba da budu aktivni. Obezbedite da svaki objekat nešto radi.
- Objekti treba da odgovaraju nekom važnom entitetu iz domena problema.
- Objekti treba da rade dovoljno da imaju značaj i važnost, ali ne toliko da postanu preveliki i kompleksni. Objekti treba da se odnose na rešiv deo problema.
- Objekti treba da imaju ulogu koju je lako objasniti nekome ko se ne bavi razvojem softvera.
- Objekti treba da budu oblikovani tako da mogu da se ponovo koriste.
- Objekti treba da se oslanjaju sami na sebe. Nepotrebnu zavisnost od okruženja treba svesti na najmanju moguću meru.
- Objekti treba da budu kompletni. Objekat treba da čuva, u razumnoj meri, sve informacije koje su neophodne za njegovo ispravno funkcionisanje.
- Objekti treba da koriste kontrolu pristupa tako da označe svoj javni interfejs, a da skrivaju svoje interne podatke i interfejse.
- Objekti treba da koriste nasledivanje i polimorfizam da bi kreirali familije specijalizovanih objekata.
- U jeziku C++ objekti treba da defiu konstruktore kako bi se uvek inicijalizovali na ispravan način.
- U jeziku C++ objekti treba da defiu destruktore koji obično treba da budu polimorfni i koji uvek mogu ponovo da oslobode memoriju i izvrše sve neophodne zadatke čišćenja.
- U jeziku C++ posebnu pažnju treba obratiti na konstruktor, konstruktor za kopiranje i operator =() funkciju članicu. Osobine ova tri člana imaju najveći uticaj na ispravan rad objekta.

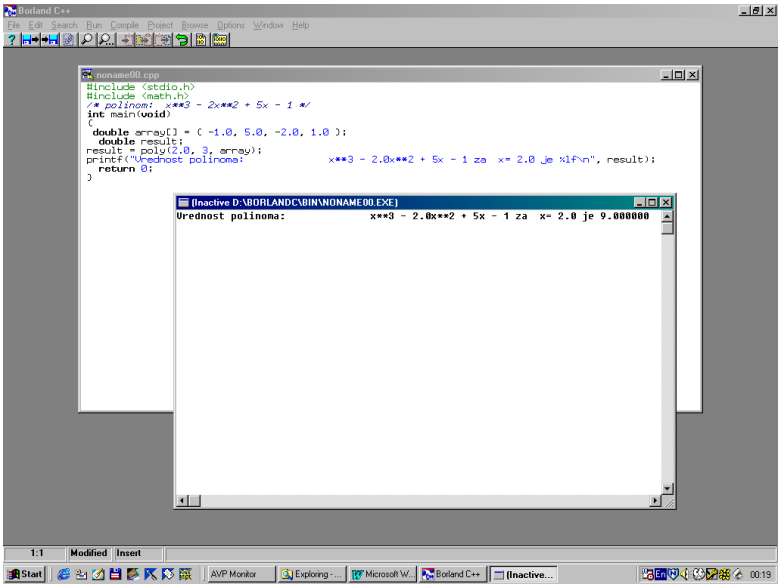
2. Tipovi podataka su slični sa tipovima podataka u C jeziku:

Tip	Dužina	Rang		
unsigned char	8 bits	0	to	255
char	8 bits	-128	to	127
enum	16 bits	-32,768	to	32,767
unsigned int	16 bits	0	to	65,535
short int	16 bits	-32,768	to	32,767
int	16 bits	-32,768	to	32,767
unsigned long	32 bits	0	to	4,294,967,295
long	32 bits	-2,147,483,648	to	2,147,483,647
float	32 bits	3.4 * (10**-38)	to	3.4 * (10**+38)
double	64 bits	1.7 * (10**-308)	to	1.7 * (10**+308)
long double	80 bits	3.4 * (10**-4932)	to	1.1 * (10**+4932)

3. Izvorni program unosimo kroz Visual C ili Borlandov editor koji otvaramo opcijom iz glavnog menija File/New. Vršimo kompajliranje i linkovanje komandom i ako u programu postoje greške moramo ih ispraviti (kao i kod C kompajlera) a ako je program bez grešaka možemo ga startovati opcijom iz glavnog menija.



Rezultati programa pojaviće se u prozoru za tu namenu:



4. Unećemo ovih nekoliko primera za objašnjenje biblioteke matematičkih funkcija u Borlandov C++ editor, izkompajlirati i linkovati programe i prepisati rezultate.

```
/* 1.primer za funkciju cos */
 #include <stdio.h>
#include <math.h>
int main(void)
{
   double result;
   double x = 0.5;
   result = cos(x);
   printf("kosinus od %lf je %lf\n", x, result);
   return 0;
}
```

```
/* 2. primer za funkciju abs  */

#include <stdio.h>
#include <math.h>
int main(void)
{
  int broj = -1234;
  printf("broj: %d absolutna vrednost: %d\n", broj, abs(broj));
  return 0;
}
```

```
/* 3. primer za funkciju log */
#include <math.h>
#include <stdio.h>
int main(void)
{
   double result;
   double x = 8.6872;
   result = log(x);
   printf("Prrirodni logaritam od %lf je %lf\n", x, result);
   return 0;
}
```

```
/* 4. primer za funkciju div */
#include <stdlib.h>
#include <stdio.h>
div_t  x;
int main(void)
{
   x = div(10,3);
   printf("10 podeljeno sa 3 = %d ostatak je %d\n", x.quot, x.rem);
   return 0;
}
```

```
/* 5. primer za funkciju pow */

#include <math.h>
#include <stdio.h>

int main(void)
{
  double x = 2.0, y = 3.0;

 printf("%lf stepenovano na %lf je %lf\n", x, y, pow(x, y));
  return 0;
}
```

```
/* 6. primer za funkciju poly */
#include <stdio.h>
#include <math.h>
/* polinom:  x**3 - 2x**2 + 5x - 1 */
int main(void)
{
 double array[] = { -1.0, 5.0, -2.0, 1.0 };
  double result;
result = poly(2.0, 3, array);
printf("Vrednost polinoma: x**3 - 2.0x**2 + 5x - 1 za  x= 2.0 je %lf\n", result);
  return 0;
}
```

5. Otkucati, kompajlirati i linkovati program za nalaženje najmanjeg od tri zadata cela broja. Upoznati se sa strukturom programa na osnovu koje se dobijaju izlazni rezultati.

```
#include <iostream.h>
int main()
{
cout<<"a=";
double a;
cin>>a;
if(!cin)
{cout <<"Greska, los ulaz! \n";
return 1;//prestanak rada programa
}

cout << "b=";
double b;
cin>>b;
if(!cin)
{cout <<"Greska, los ulaz! \n";
return 1;//prestanak rada programa
}

cout << "c=";
double c;
cin>>c;
if(!cin)
{cout <<"Greska, los ulaz! \n";
return 1;//prestanak rada programa
}

//nalazenje minimuma sa validnim promenljivima a,b i c

double min=a;
if (b<min) min=b;
if (c<min) min=c;
cout<<"min{"<<a<<", "<<b<<", "<<c<<"}="<<min<<"\n";
return 0;
}
```

Napomena: Uneti brojeve sa tastature kao a=3 b=4 i c=2

Iz korisničkog prozora prepisati rezultate rada programa:

6. Otkucati, kompajlirati i linkovati program i uporediti rezultate rada programa. Upoznati se sa strukturom programa na osnovu koje se dobijaju izlazni rezultati.

```
#include <stdio.h>
#include <string.h>
#define I 555
#define R 5.5
int main(void)
{
  int i,j,k,l;
  char buf[7];
  char *prefix = buf;
  char tp[20];
  printf("prefix  6d     6o     8x       10.2e       "
       "10.2f\n");
  strcpy(prefix,"%");
  for (i = 0; i < 2; i++)
  {
    for (j = 0; j < 2; j++)
      for (k = 0; k < 2; k++)
        for (l = 0; l < 2; l++)
        {
          if (i==0)  strcat(prefix,"-");
          if (j==0)  strcat(prefix,"+");
          if (k==0)  strcat(prefix,"#");
          if (l==0)  strcat(prefix,"0");
          printf("%5s |",prefix);
          strcpy(tp,prefix);  // Kopira string scr na odrediste
          strcat(tp,"6d |");   // Dodaje jedan string do drugoga
          printf(tp,I);
          strcpy(tp,"");
          strcpy(tp,prefix);
          strcat(tp,"6o |");
          printf(tp,I);
          strcpy(tp,"");
          strcpy(tp,prefix);
          strcat(tp,"8x |");
          printf(tp,I);
          strcpy(tp,"");
          strcpy(tp,prefix);
          strcat(tp,"10.2e |");
          printf(tp,R);
          strcpy(tp,prefix);
          strcat(tp,"10.2f |");
          printf(tp,R);
          printf("  \n");
          strcpy(prefix,"%");
        }
    }
  return 0;}
```

REZULTATI RADA PROGRAMA:

```
prefix  6d      6o      8x        10.2e        10.2f
%-+#0 |+555   |01053  |0x22b    |+5.50e+00  |+5.50      |
 %-+# |+555   |01053  |0x22b    |+5.50e+00  |+5.50      |
 %-+0 |+555   |1053   |22b      |+5.50e+00  |+5.50      |
  %-+ |+555   |1053   |22b      |+5.50e+00  |+5.50      |
 %-#0 |555    |01053  |0x22b    |5.50e+00   |5.50       |
  %-# |555    |01053  |0x22b    |5.50e+00   |5.50       |
  %-0 |555    |1053   |22b      |5.50e+00   |5.50       |
   %- |555    |1053   |22b      |5.50e+00   |5.50       |
 %+#0 |+00555 |001053 |0x00022b |+005.50e+00 |+0000005.50 |
  %+# |  +555 | 01053 |   0x22b | +5.50e+00 |     +5.50 |
  %+0 |+00555 |001053 |0000022b |+005.50e+00 |+0000005.50 |
   %+ |  +555 |  1053 |     22b | +5.50e+00 |     +5.50 |
  %#0 |000555 |001053 |0x00022b |005.50e+00 |0000005.50 |
   %# |   555 | 01053 |   0x22b |  5.50e+00 |      5.50 |
   %0 |000555 |001053 |0000022b |005.50e+00 |0000005.50 |
    % |   555 |  1053 |     22b |  5.50e+00 |      5.50 |
```

Napomena: Ispreviti grešku u programu kako bi se dobila pregledna tabela i napisati nove programske linije u kojima je urađena ispravka.

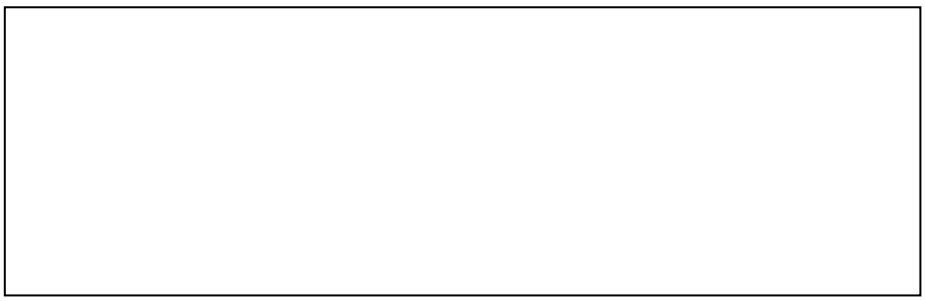
DRUGA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

2.1. Preko editora uneti sledeći program, izkompajlirati ga i linkovati, ispraviti moguće greške i upoznati se sa osnovnim tipovima promenljivih u C++ jeziku.

```
#include<iostream.h>
int main(){
char c='A';
unsigned char cu='A';
int i=100;
unsigned int iu=100;
short int is=100;
short iis=100; // Same as short int
unsigned short int isu=100;
unsigned short iisu=100;// Same as unsigned short int
long int il=100;
long iil=100; // Same as long int
unsigned long int ilu=100;
unsigned long iilu=100; //Same as long int
float f=12345.54321;
double d=12345.54321;
long double ld=12345.54321;
cout<< "\n char= " << sizeof(c);//Odredjuje broj u bajtima za zauzece memorije
cout<< "\n unsigned char = " << sizeof(cu);
cout<< "\n int = " << sizeof(i);
cout<< "\n unsigned int = " << sizeof(iu);
cout<< "\n short = " << sizeof(is);
cout<< "\n unsigned short = " << sizeof(isu);
cout<< "\n long = " << sizeof(il);
cout<< "\n unsigned long = " << sizeof(ilu);
cout<< "\n float = " << sizeof(f);
cout<< "\n double = " << sizeof(d);
cout<< "\n long double = " << sizeof(ld);
cout<< "\n char= " <<c;
cout<< "\n unsigned char = " <<cu;
cout<< "\n int = " <<i;
cout<< "\n unsigned int = " <<iu;
cout<< "\n short = " <<is;
cout<< "\n unsigned short = " <<isu;
cout<< "\n long = " <<il;
cout<< "\n unsigned long = " <<ilu;
cout<< "\n float = " <<f;
cout<< "\n double = " <<d;
cout<< "\n long double = " <<ld;
}
```

2.2. Sastavićemo, kompajlirati i linkovati program koji prikazuje rad sa globalnim i lokalnim promenljivima. Naravno, u praktikum prepisujemo rezultate rada programa.

```
// Program koji prikazuje pristup globalnoj promenljivoj //(istog imena kao i lokalna)
#include <iostream.h>

float suma(float f1, float f2);
float f= 10.5;

main()
{ float a=10.5, b=10.5;

cout<<"Suma " << a <<" + " << b << " + " << f << " je: " << suma( a, b);

return 0;
}
float suma(float f1,float f2)
{ float f= 20.5;

return(f1 + f2 + (::f) );
}
```



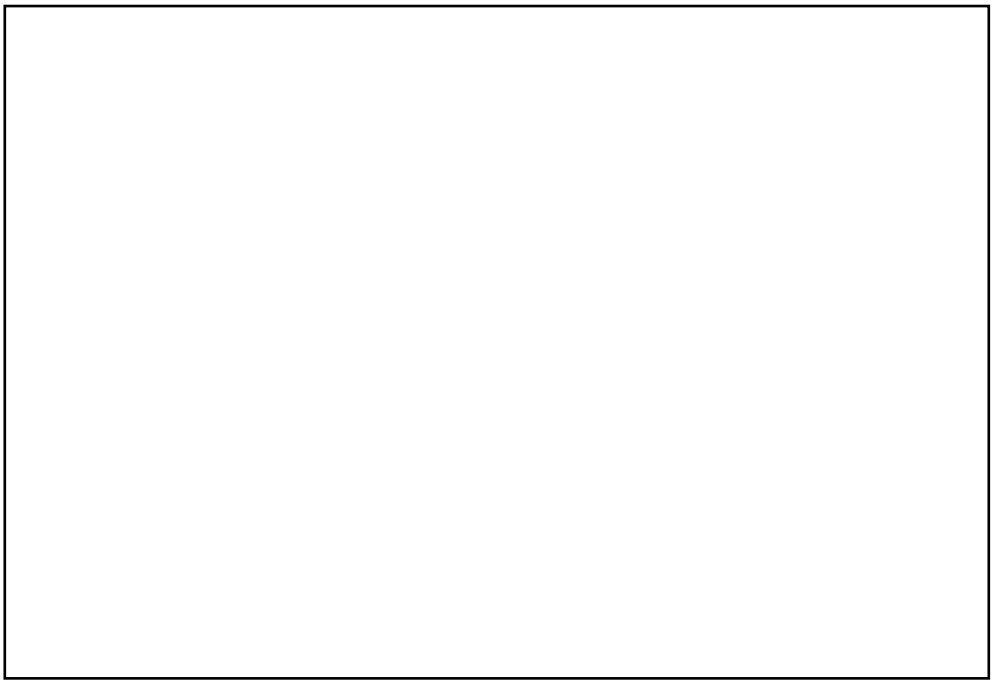
// 2.3. Program koji koristi funkcije za konverziju i formatiranje ulaza/izlaza

```
#include <iostream.h>      //biblioteka funkcija ulaza/izlaza
#include <iomanip.h> //biblioteka funkcija za I/O formatiranje #include <string.h>

void stampa_r_broj( void );
main( )
{
char slovo = 'A';
char string[] = "Ovaj program ilustruje formatiranje C++ ulaza/izlaza";
int broj_int = 10;
float broj_float = 123.4567;

 //******************KONVERZIJA*******************

//1. stampanje samog slova A
stampa_r_broj();
cout << slovo;
```

```
//2. stampanje ASCII koda slova A
stampa_r_broj();
cout << (int)slovo;

//3. stampanje znaka cija je ASCII vrednost zadata (slovo a)
stampa_r_broj();
cout << (char)97;

//4. stampanje stringa
stampa_r_broj();
cout << string;

//5. stampanje odredjenog broja prvih karaktera stringa
stampa_r_broj();
cout.write(string,5);//vraca definisan niz karaktera

//6. stampanje celog broja u oktalnom obliku
stampa_r_broj();
cout.setf(ios::oct);//koristi se sa setf za levo i desno
                                        //poravnanje
cout << broj_int;           //moze i ovako: cout << oct << broj_int;
cout.unsetf(ios::oct);
//7.stampanje celog broja u heksa obliku (malim slovima)
stampa_r_broj();
cout.setf(ios::hex);//koristi se sa setf za levo i desno
                                        //poravnanje
cout << broj_int; //moze i ovako: cout << hex << broj_int;

//8. stampanje celog broja u heksa obliku (velikim slovima)
stampa_r_broj();
cout.setf(ios::uppercase); //funkcije clanice
cout << broj_int;
cout.unsetf(ios::hex);

//9. stampanje celog broja u decimalnom obliku
stampa_r_broj();
cout << broj_int; //moze i ovako: cout << dec << broj_int;

//10. stampanje celog broja, u polju zadate sirine, desno poravnanje
stampa_r_broj();
cout.width(10); //setuje ga na tekucu sirinu
cout << broj_int;
```

```
//11. stampanje celog broja, u polju zadate sirine, levo //poravnanje
stampa_r_broj();
cout.width(10); //setuje ga na tekucu sirinu
cout.setf(ios::left);
cout << broj_int;
cout.unsetf(ios::left);

//12. stampanje celog broja, desno popravnanje, nule ispred //broja
stampa_r_broj();
cout.width(10);
cout.fill('0');//vraca tekuci karakter za popunu
cout << broj_int;
cout.fill(' ');//vraca tekuci karakter za popunu

//13. stampanje realnog broja, sa standardnom preciznoscu
stampa_r_broj();
cout << broj_float;

//14. stampanje realnog broja, u polju zadate sirine,
stampa_r_broj();
cout.width(10);
cout << broj_float;

//********** FORMATI-PRECIZNOSTI  **********

//15. stampanje realnog broja, sa fiksiranom preciznoscu
stampa_r_broj();
cout.width(10);
cout.setf(ios::fixed);
cout.precision(2); //prosirena tacnost
cout << broj_float;
cout.unsetf(ios::fixed);

//16. stampanje realnog broja, sa naucnom preciznoscu
stampa_r_broj();
cout.setf(ios::scientific);//naucna tacnost
cout << broj_float;
cout.unsetf(ios::scientific);

return 0;
}

void stampa_r_broj (void)
{
static int redni_broj = 0;
cout << "\n";
cout.width(2);
cout << ++redni_broj << ". ";
}
```

Napisati rezultate rada programa i protumačiti po tačkama svaki izlazni rezultat.



TREĆA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

3.1. U editoru otkucati izvorni C++ program za demonstraciju korišćenja prefix i postfix inkrementnih i dekrementnih operatora, kompajlirati i linkovati program i prepisati izlazne rezultate.

```
// Listing 3.1 - demonstracija koriscenja
// prefix i postfix inkrementnih i
// dekrementnih operatora
#include <iostream.h>
int main()
{
   int myAge = 39;     // inicijalizacija dva integer broja
   int yourAge = 39;
   cout << "Ja sam: " << myAge << " godina star.\n";
   cout << "Ti si: " << yourAge << " godina star.\n";
   myAge++;         // postfix increment
   ++yourAge;       // prefix increment
   cout << "Posle jedne godina...\n";
   cout << "Ja sam: " << myAge << " godina star.\n";
   cout << "Ti si: " << yourAge << " godina star.\n";
   cout << "Posle jos jedne godine\n";
   cout << "Ja sam: " << myAge++ << " godina star.\n";
   cout << "Ti si: " << ++yourAge << " godina star.\n";
   cout << "Stampacemo ponovo.\n";
   cout << "Ja sam: " << myAge << " godina star.\n";
   cout << "Ti si: " << yourAge << " godina star.\n";
   return 0;
}
```

IZLAZ:

3.2. Napisati program u programskom jeziku C++ za izračunavanje aritmetičke sredine μ i sume razlike kvadrata σ članova reda i aritmetičke sredine niza celih brojeva $x_1,x_2,x_3,x_4,x_5,.....,x_n$ po obrascima:

$$\mu = \frac{1}{N}\sum_{1\le i\le N} x_i \qquad i \qquad \sigma^2 = \frac{1}{N}\sum_{1\le i\le N} x_i{}^2 - \mu^2$$

U programu postaviti N=10 a vrednosti za x uneti u program korišćenjem generatora slučajnih brojeva rand() koji treba organizovati u obliku potprograma.

```
//listing 3.2. Izracunavanje aritmeticke sredine i sume //razlike kvadrata
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
typedef int Number;
Number randNum()
        { return rand(); }
main()
{
        int i, N = 10;
        float m1 = 0.0, m2 = 0.0;
        Number x;
for (i = 0; i < N; i++)
{
                x = randNum();
                printf("Vrednost x je:%d\n",x);
                m1 += ((float) x)/N;
                m2 += ((float) x*x)/N;
}
printf(" Srednja vrednost: %f\n", m1);
printf ("Standardna greska: %f\n", sqrt (m2-m1*m1) ) ;
}
```

Prepisati rezultate rada programa i sagledati njegovu strukturu.

3.3. Sastaviti program na C++ jeziku za formiranje Eratostenovog sita koje iz skupa celih brojeva {1,2,3,4,5,6,.....,N} izdvaja samo proste brojeve. Program uraditi za slučaj kada je zadato N=31. Na osnovu programa sastaviti matematički model za izdvajanje prostih brojeva iz skupa celih brojeva, odnosno formulisati Eratostenov postulat.

```
//3.3. Formiranje Eratostenovog sita
#include <math.h>
#include<stdio.h>
#define N 31
main()
{
int i,j,a[N];

printf("U skupu od %d.  broja   prosti  brojevi su:   \n", N);

        for (i = 2; i < N; i++)

        a[i] = 1;

                for (i = 2; i < N; i++)

        if (a[i] )

                for(j = i; i*j < N; j++)

        a[i*j] = 0;

                for (i = 2; i < N; i++)

        if (a[i])


printf("%4d ", i);

printf ("\n") ;
}
```

Rezultati programa su:

3.4. Sastaviti program za prikaz celobrojnog i realnog deljenja dva broja. U okviru funkcija realizovati ova deljenja, pa ih kao takva pozvati u glavni program. Prepisati rezultate.

```
//3.4.Primer celobrojnog i realnog deljenja
 #include <iostream.h>

 void intDiv(int x, int y)
{
     int z = x / y;
     cout << "U celobrojnom deljenju z: " << z << endl;
}

 void floatDiv(int x, int y)
{
    float a = (float)x;    // old style
    float b = (float)y;    // preferred style
    float c = a / b;

    cout << "U realnom deljenju c: " << c << endl;
}

 int main()
{
     int x = 5, y = 3;
     intDiv(x,y);
     floatDiv(x,y);
     return 0;
}
```

IZLAZ:

3.5. Upoznati se sa korišćenjem funkcija sa podrazumevanim vrednostima argumenata.

```
//3.5. Program koji koristi funkciju sa podrazumevanim //vrednostima argumenata
#include<iostream.h>
void podr_argumenti(char c ='X',int broj1=0,float broj2= 0.5);
main()
{ podr_argumenti('A', 1, 1.5);   //stampa: A, 1, 1.5
podr_argumenti('B', 2);          //stampa: B, 2, 0.5
podr_argumenti('C');                    //stampa: C, 0, 0.5
podr_argumenti();                              //stampa: poruku, 0, 0.5
return 0; }

void podr_argumenti(char c, int brojl, float broj2)
{ if( c == 'X')
cout << "\nNije izabrana opcija" << endl;
else
cout << "\nIzabrana opcija: " << c << endl;
cout << "Ceo broj: " << brojl << endl;
cout << "Realan broj: " << broj2 << endl;
}
```

3.6. Upoznati se sa korišćenjem funkcija sa rad sa nizovima celih i realnih brojeva.

```
//3.6.Program koji koristi dve preklopljene funkcije za //izracunavanje srednje vrednosti
niza
#include <iostream.h>
#define MAX 4
int srednja_vrednost( int niz_int[], int n);
float srednja_vrednost( float niz_float[], int n);

main()
{ int a[MAX]={1,2,3,4};
float b[MAX] = { 1.5, 2.5, 3.5, 4.5};
cout <<"Srednja vrednost celih brojeva:" << srednja_vrednost(a, MAX) << endl;
cout <<"Srednja vrednost realnih brojeva:" <<srednja_vrednost(b, MAX)<< endl;
return 0; }
int srednja_vrednost( int niz_int[], int n)
{ int s = 0;
for(int i=0; i<n; i++)
s += niz_int[i] ;
return(s/n);
}
float srednja_vrednost( float niz_float[], int n )
{ float s = 0;
for(int i=0; i<n; i++ )
s += niz_float[i] ;
return (s/n); }
```

Prepisati rezultate rada programa!!!!!!!!!!!!

ČETVRTA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

4.1. Napisati program koji za veličinu x - realno i n - celobrojno izračunava sumu:

$$s = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots\ldots + \frac{x^n}{n!}$$

```
// Program Zad28.cpp
#include <iostream.h>
int main()
{cout << "x= ";
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n";
return 1; }
cout << "n= ";
short n;
cin >> n;
if(!cin)
{cout << "Error. Bad input! \n";
return 1; }
if(n <= 0)
{cout << "Incorrect input! \n";
return 1; }
double x1 = 1;
double s = 1;
for (short i=1; i<= n; i++)
{x1 = x1 * x / i;
s = s + x1; }
cout << "s= " << s << "\n";
return 0;
}
```

Prepisati reziltate rada programa za x=1. i n=2 i za x=2. i n=4:

4.2. Neka su m i n dati celi brojevi, n>=1 i m>1. Napisati program koji daje broj elemenata serije brojeva $i^3 + 7i^2 + n^3$, i=1,2,.....,n koji su deljivi sa m.

```
// Program Zad29.cpp
#include <iostream.h>
int main()
{cout << "n= ";
int n;
cin >>n;
if (!cin)
{cout << "Error. Bad input! \n"; return 1; }
if (n < 1)
{cout << "Incorrect input! \n"; return 1; }
cout << "m= ";
int m;
cin >> m;
if (!cin)
{cout << "Error. Bad input! \n"; return 1; }
if (m <= 1)
{cout << "Incorrect input! \n"; return 1; }
int br = 0;
for (int i = 1; i <= n; i++)
{cout<< i*i*i + 7*i*i + n*n*n<<"\n";
if ((i*i*i + 7*i*i + n*n*n) % m == 0) br++;
cout << "br= " << br << "\n"; }
return 0;
}
```

Napomena: Kontrolu uraditi za n=2,m=2; n=3,m=2 i n=4,m=4 i prepisati rezultate.

4.3. Dat je celi broj n, n>=1. Napisati program koji nalazi najveći broj iz serije brojeva: $i^2(n+i/n)$, i=1,2,3,.....,n. U algoritmu predpostaviti da je prvi najve}i broj za i=1.

```
// Program Zad30.cpp
#include <iostream.h>
#include <math.h>
int main()
{cout << "n= ";
int n;
cin >> n;
if (!cin)
{cout << "Error. Bad input! \n"; return 1;}
if (n < 1)
{cout << "Incorrect input! \n"; return 1;}
double max = (n+1.0/n);
for (int i = 2; i <= n; i++)
{double p = i*i*(n+(double)i/n);
cout<<"p="<<p<<"\n";
if (p > max) max = p; }
cout << "max= " << max << "\n";
return 0; }
```

4.4. Liniju po liniju razmotriti izvorni kod za demonstraciju polimorfizma. Prepisati rezultate rada programa.

```
// Listing 5.8 - demonstracija
// polimorfizma

#include <iostream.h>
int Double(int);
long Double(long);
float Double(float);
double Double(double);
int main()
{
   int     myInt = 6500;
   long    myLong = 65000;
   float   myFloat = 6.5F;
   double  myDouble = 6.5e20;

   int     doubledInt;
   long    doubledLong;
   float   doubledFloat;
   double  doubledDouble;

   cout << "myInt: " << myInt << "\n";
   cout << "myLong: " << myLong << "\n";
   cout << "myFloat: " << myFloat << "\n";
   cout << "myDouble: " << myDouble << "\n";

   doubledInt = Double(myInt);
   doubledLong = Double(myLong);
   doubledFloat = Double(myFloat);
   doubledDouble = Double(myDouble);

   cout << "doubledInt: " << doubledInt << "\n";
   cout << "doubledLong: " << doubledLong << "\n";
   cout << "doubledFloat: " << doubledFloat << "\n";
   cout << "doubledDouble: " << doubledDouble << "\n";

    return 0;
}
int Double(int original)
{
 cout << "In Double(int)\n";
 return 2 * original;
}
long Double(long original)
{
 cout << "In Double(long)\n";
 return 2 * original;
}
```

```
float Double(float original)
{
  cout << "In Double(float)\n";
  return 2 * original;
}

double Double(double original)
{
  cout << "In Double(double)\n";
  return 2 * original;
}
```

4.5. Na}i najveći element niza $x_1,x_2,.....,x_n$ (n<=50). U programu učitati da je n=10 i proizvoljno uneti deset celobrojnih elemenata niza. Prepisati rezultate.

```
// Program koji odredjuje max za zadati dinamicki niz celih brojeva
#include <iostream.h>
#include <stdlib.h>
#define MAX 50
main()
{ int i, n, max, *niz;
do
{cout << "\nBroj elemenata niza: " << endl; cin >> n;}
while(n<1 || n>MAX);

niz = new int[n]; //dodela memorije za niz
if(niz==NULL)
{ cerr << "Nije uspela dodela memorije" << endl; exit(1); }

cout << "\nUnesite vrednosti elemenata niza: ";
for(i=0; i<n; i++)
cin >> niz[i];

max = niz[0];
for(i=1; i<n; i++)
if(niz[i]>max)
max = niz[i];
cout<< "Najveca vrednost zadatog niza je " << max << endl;

delete(niz); //uklanjanje niza iz memorije
return 0;
}
```

4.5. Neka je a realan broj. Napisati program koji pribli`no rešava kvadratni koren po a korišćenjem Njutnove metode:

$x_0=1$ i $x_{i+1}=(x_i+a/x_i)$, i=1,2,3,.....

```
// Program Zad41.cpp
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << "a= ";
double a;
cin >> a;
if (!cin)
{cout << "Error! Bad Input! \n"; return 1; }
if (a < 0)
{cout << "Incorrect Input! \n"; return 1; }
cout << "eps= ";
double eps;
cin >> eps;
if (!cin)
{cout << "Error! Bad Input! \n"; return 1; }
if (eps <= 0 || eps > 0.5)
{cout << "Incorrect Input! \n"; return 1; }
double x0;
double x1 = 1;
do
{x0 = x1;
x1 = 0.5*(x0 + a/x0); }
while (fabs (x1-x0) >= eps);
cout << setprecision(6) << setiosflags(ios :: fixed);
cout << "sqrt(" << a << ")= " << setw(10) << x1 << "\n";
return 0;
}
```

PETA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

5.1. Napisati program na C++ jeziku koji proverava da li je niz dužine ne veće od 19 karaktera palindrom ili ne. Palindromi su reči ili rečenice koje se čitaju isto i sa leva na desno i obrnuto (naprimer: ana voli milovana). U programu koristiti funkcije strlen(x) koja vraća integer broj karaktera unetog niza karaktera i funkciju strcmp(x,y) za upoređenje niza x i niza y. Ako je niz x manji od niza y vratiće -1, ako je niz x jednak nizu y vratiće 0 i ako je niz x veći od niza y vratiće 1.

```
//Program Zad61.cpp
//Palindrom!!!!
#include<iostream.h>
#include<string.h>
int main()
{
char a[20],b[20];
cout<<"a= ";
cin>>a;
int n=strlen(a);
int i;
for(i=n-1;i>=0;i--)
{b[n-i-1]=a[i];
b[n]='\0';}
if(!strcmp(a, b)) cout<<"Niz je palindrom\n";
else cout<<"Niz nije palindrom\n";
return 0;
}
```

Prepisati rezultate rada programa za proizvoljno unete stringove:

5.2. Napisati program na C++ jeziku koji proverava da li je kvadratna matrica du`ine (4x4) sastavljena od niza karaktera simetrična u odnosu na glavnu dijagonalu.

$$a = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix}$$ što bi značilo da je a(i,j)=a(j,i), i=1,..,4;

j=1,...,4

```
//Program Zad66.cpp
//Simetricnost matrice
#include <iostream.h>
#include <string.h>
int main()
{
char a[1][4][4];
cout << "n= ";
int n;
cin >> n;            //unecemo n=3 zbog zauzeca memorije
int i, j;
for (i = 0; i <= n-1; i++)
for (j = 0; j <= n-1; j++)
{cout << "a [" << i << "][" << j << "]= " ;
cin >> a[i][j]; }    //unos elemenata matrice
cout<<"Elementi matrice su:\n";
for (i = 0; i <= n-1; i++)
for (j = 0; j <= n-1; j++)
{cout << a[i][j]<<"\t";      //stampanje elemenata date matrice
if(j!=0 && j%(n-1)==0)cout<<"\n";}
i = 0;
int p;
do
{i++;
j=-1;
do
{j++;
p = !strcmp(a[i][j], a[j][i]); }
while (p && j < i-1); }
while (p && i < n-1);
if (p) cout << "yes\n";
else cout << "no \n";
return 0;
}
```

Prepisati rezultate rada programa za proizvoljne podatke matrice 3x3 :

5.3. Napisati program na C++ jeziku koji izračunava aritmetičku sredinu niza karaktera predstavljenih celim brojevima {'1','2', '3', '4',.....}. Aritmetička sredina se računa po obrascu:

$$S = \frac{\sum_{i=1}^{n} x_i}{n}$$

```
//Program Zad65.cpp
//Aritmeticka sredina niza brojeva
#include <iostream.h>
#include <stdlib.h>
#include<iomanip.h>
int main()
{
char a[1][6];
cout << "n= ";
int n;
cin >> n;            //unecemo n=4 zbog zauzeca memorije
int i;
for (i = 0; i <= n-1; i++)
{cout << "a [" << i <<"]= " ;
cin >> a[i]; }       //unos elemenata niza
cout<<"Elementi niza su:\n";
for (i = 0; i <= n-1; i++)
cout << setw(10)<<a[i]<<"\t";       //stampanje elemenata datog niza
double average = 0;
for(i=0;i<=n-1;i++)
average=average+atoi(a[i]);
cout <<"\nAritmeticka sredina niza je:"<< average/n<<"\n";
return 0;
}
```

Prepisati rezultate rada programa za sledeće parametre:

za n=4 elementi niza 1: {4,2,6,8}	za n=4 elementi niza 2: {7,2,6,1}

5.4. Napisati program na C++ jeziku koji izra~unava sumu svih elemenata u svakoj koloni pojedina~no, pravougaone matrice a[nxn].

$$A=\begin{vmatrix} a_{11} & a_{12} & a_{1n} \\ a_{21} & a_{22} & a_{2n} \\ a_{n1} & a_{n2} & a_{nn} \end{vmatrix}$$

```
//Program Zad56.cpp
//Suma elemenata matrice
#include <iostream.h>
#include <iomanip.h>
int main()
{
int a[4][4];
cout << "n= ";
int n;
cin >> n;            //unecemo n=4 zbog zauzeca memorije
if(!cin){cout<<"Greska u unosu!\n";return 1;}
if(n<1 || n>5){cout<<"Nekorektan unos!\n";return 1;}
int i,j;
for (i = 0; i <= n-1; i++)
for (j = 0; j <= n-1; j++)
{cout << "a [" << i << "][" << j << "]= " ;
cin >> a[i][j];      //unos elemenata matrice
if(!cin){cout<<"Greska u unosu!\n";return 1;}}
cout<<"Elementi matrice su:\n";
for (i = 0; i <= n-1; i++)
for (j = 0; j <= n-1; j++)
{cout << a[i][j]<<"\t";      //stampanje elemenata date matrice
if(j!=0 && j%(n-1)==0)cout<<"\n";}
for (j = 0; j <= n-1; j++)
{int s=0;
for (i = 0; i <= n-1; i++)
s+=a[i][j];
cout<<setw(10)<<"\nKolona:"<<j+1<<setw(10)<<"Sumaje: "<< s << "\n"; }
return 0;
}
```

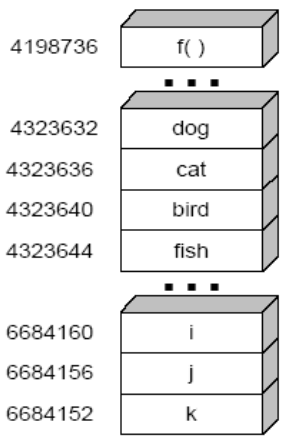
Prepisati rezultate rada programa:

ŠESTA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

6.1. Sagledavajući programske linije jednu po jednu, upoznati se sa funkcijama pokazivača.

```
//: Cpp6.1.:YourPets2.cpp
#include <iostream.h>
int dog, cat, bird, fish;
void f(int pet)
{
cout << "pet id number: " << pet << endl;
}
int main()
{
int i, j, k;
cout << "f(): " << (long)&f << endl;
cout << "dog: " << (long)&dog << endl;
cout << "cat: " << (long)&cat << endl;
cout << "bird: " << (long)&bird << endl;
cout << "fish: " << (long)&fish << endl;
cout << "i: " << (long)&i << endl;
cout << "j: " << (long)&j << endl;
cout << "k: " << (long)&k << endl;
}
```

Izlaz iz programa bi bio predstavljen u obliku adresa:



6.2. Na osnovu sledećeg programa razmotriti funkciju pokazivača redirekcije * i pokazivača adresa &.

```
//: Cpp6.2.:PassAddress.cpp
//Rad sa pointerima
#include <iostream.h>
void f(int *p) // funkcija sa pokazivacem redirekcije
{
cout << "p = " << p << endl;
cout << "*p = " << *p << endl;
*p = 5;
cout << "p = " << p << endl;
}
int main()
{
int x = 47;
cout << "x = " << x << endl;
cout << "&x = " << &x << endl;
f(&x);
cout << "x = " << x << endl;
}
```

IZLAZ:

```
x = 47
&x = 0x1ae4
p = 0x1ae4
*p = 47
p = 0x1ae4
x = 5
```

6.3. Sledeći primer programa ukazuje na adresne pokazivače sa vrednošću po referenci.

```
//: Cpp6.3.:PassReference.cpp
#include <iostream.h>
void f(int &r) // funkcija sa pokazivacem adresa
{
cout << "r = " << r << endl;
cout << "&r = " << &r << endl;
r = 5;
cout << "r = " << r << endl;
}
int main()
{
int x = 47;
cout << "x = " << x << endl;
cout << "&x = " << &x << endl;
f(x); // Izgleda kao prelazna vrednost,
        // ali je u stvari vrednost po referenci
cout << "x = " << x << endl;
}
```

//IZLAZ:
//x = 47
//&x = 0x1ae4
//r = 47
//&r = 0x1ae4
//r = 5
//x = 5

6.4. Sa ukazivačima je moguće baratati kao i sa običnim tipovima podataka. Naravno korišćenjem inkrementa i dekrementa dobijamo sada sasvim drugačije rezultate.

```
//: Cpp6.4.:PointerIncrement.cpp
#include <iostream.h>
int main()
{
int i[10];
double d[10];
int *ip = i;
double *dp = d;
cout << "ip = " << (long)ip << endl;
ip++;
cout << "ip = " << (long)ip << endl;
cout << "dp = " << (long)dp << endl;
dp++;
cout << "dp = " << (long)dp << endl;
}
```

IZLAZ:
ip = 1556028580
ip = 1556028582
dp = 1556028500
dp = 1556028508

6.5. Sa ukazivačima je moguće baratati kao i sa običnim tipovima podataka. Naravno ako koristimo ukazateljsku aritmetiku, dobijamo sada sasvim drugačije rezultate.

```
//: Cpp6.5.:PointerArithmetic.cpp
#include <iostream.h>
#define P(EXP) \
cout << #EXP << ": " << EXP << endl; //The # (pound sign) //indicates a preprocessor
directive when it occurs as the //first non-whitespace character on a line.
int main()
{
int a[10];
for(int i = 0; i < 10; i++)
a[i] = i; // Give it index values
int* ip = a;
P(*ip);
```

```
P(*++ip);
P(*(ip + 5));
int* ip2 = ip + 5;
P(*ip2);
P(*(ip2 - 4));
P(*--ip2);
}
```

IZLAZ:
***ip: 0**
***++ip: 1**
***(ip+5): 6**
***ip2: 6**
***(ip2-4): 2**
--ip2: 5

6.6. Primer za različite pozive funkcija u zavisnosti od tipa argumenta:

```
// Program koji koristi poziv funkcije na tri nacina: po //vrednosti,referenci pomocu
pokazivaca i po referenci pomocu //C++ referentnog tipa.
#include <iostream.h>
#define MAX_BROJ 100
#define MAX_IME 30
struct student  //Definisanje strukture student
   {char ime[MAX_IME+1];
            int reg_broj;
            int god_upisa;
            int polozenih; };
void po_vrednosti(student x); //Definisanje tri razlicite
void po_ref_pok(student *ptr_x); //funkcije
void po_ref_ref(student &ref_x);
main()
{ student s1 ;
s1.polozenih = 10;

//Predaja funkciji kopije strukture
po_vrednosti(s1);
cout <<"Direktna kopija strukture S1="<< sl.polozenih << endl; //broj_polozenih je i
dalje 10
//Predaja funkciji pokazivaca na strukturu
po_ref_pok(&s1);
cout <<"Pokazivac na strukturu S1="<< sl.polozenih << endl; //broj_polozenih izmenjen:
11
//Predaja funkciji reference strukture
po_ref_ref(s1);
cout <<"Po referenci strukture S1="<< sl.polozenih << endl; //broj_polozenih izmenjen:
12
```

```
return 0;
}
void po_vrednosti(student x)
{ x.polozenih++;            //sintaksa strukture
}
void po_ref_pok(student *ptr_x)
{ ptr_x->polozenih++;                    //sintaksa pokazivaca
}
void po_ref_ref(student &ref_x)
{ ref_x.polozenih++;                     //sintaksa reference
}
```

1. Zadatak za razmišljanje: Mogućnosti upotrebe naredbe return.

```
//: C03:Return.cpp
// Koriscenje "return"
#include <iostream.h>
char cfunc(int i)
{
if(i == 0)
return 'a';
if(i == 1)
return 'g';
if(i == 2)
return 'z';
if(i == 3)
return 'c';
}
int main()
{
cout << "Unesite celi broj: ";
int val;
cin >> val;
cout << cfunc(val) << endl;
}
```

2. Zadatak za razmišljanje: Mogućnosti upotrebe naredbe break i continue.

```
//: C03:Menu.cpp
// Simple menu program demonstrating
// the use of "break" and "continue"
#include <iostream.h>
int main() {
char c; // To hold response
while(1) {
cout << "MAIN MENU:" << endl;
cout << "l: left, r: right, q: quit -> ";
```

```
cin >> c;
if(c == 'q')
break; // Out of "while(1)"
if(c == 'l') {
cout << "LEFT MENU:" << endl;
cout << "select a or b: ";
cin >> c;
if(c == 'a') {
cout << "you chose 'a'" << endl;
continue; // Back to main menu
}
if(c == 'b') {
cout << "you chose 'b'" << endl;
continue; // Back to main menu
}
else {
cout << "you didn't choose a or b!"<< endl;
continue; // Back to main menu
}
}
if(c == 'r') {
cout << "RIGHT MENU:" << endl;
cout << "select c or d: ";
cin >> c;
if(c == 'c') {
cout << "you chose 'c'" << endl;
continue; // Back to main menu
}
if(c == 'd') {
cout << "you chose 'd'" << endl;
continue; // Back to main menu
}
else {
cout << "you didn't choose c or d!"
<< endl;
continue; // Back to main menu
}
}
cout << "you must type l or r or q!" << endl;
}
cout << "quitting menu..." << endl;}
```

SEDMA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

KLASE:

- *Klasa* je realizacija apstrakcije koja ima svoju internu predstavu (svoje atribute) i operacije koje se mogu vršiti nad njenim instancama. Klasa defie tip. Jedan primerak takvog tipa (instanca klase) naziva se *objektom* te klase (engl. *class object).*
- Podaci koji su deo klase nazivaju se *podaci članovi* klase (engl. *data members).* Funkcije koje su deo klase nazivaju *sefunkcije članice* klase (engl. *member functions).*
- Članovi (podaci ili funkcije) klase iza ključne reči *private* : zaštićeni su od pristupa spolja (enkapsulirani su). Ovim članovima mogu pristupati samo funkcije članice klase. Ovi članovi nazivaju se *privatnim članovima klase* (engl. *private class members).*
- Članovi iza ključne reči *pub1ic* : dostupni su spolja i nazivaju se *javnim članovima klase* (engl. *public class members).*
- Članovi iza ključne reči *protected* : dostupni su funkcijama članicama date klase, kao i klasa izvedenih iz te klase, ali ne i korisnicima spolja, i nazivaju se *zaštićenim članovima klase* (engl. *protected class members)*

Definicija klase predstavlja navođenje svih članova klase. Na osnovu te definicije mora da se zna veličina potrebanog memorijskog prostora za smeštanje pojednih objekata tipa te klase. Klasa se defie opisom *class* čiji je opšti oblik:

class *Identifikator* { *član član* -
public: *član član* -
private: *član član* - -) };

Deklaracijom klase se samo naznači da neki identifikator predstavlja klasu, ali se ta ne kaže o sadržaju klase. Opšti oblik deklaracije klase je:

class *Identifikator* ;

Posle deklaracije klase mogu da se defiu pokazivači i upućivači na objekte tipa te klase, ali ne mogu da se defiu objekti tipa te klase. Za definisanje objekata neophodno je da prethodno bude navedena potpuna definicija klase. Pri definisanju objekata vrši se i dodela memorije, a za to je potrebno je da se zna veličina objekata!

Identifikator klase služi za identifikaciju klase koja se defie ili deklariše. Ima status identifikatora tipa, pa može samostalno da se koristi u naredbama za definisanje podataka i na drugim mestima gde se očekuje oznaka tipa.

Javni i privatni delovi klase, kao što znamo, razgraničavaju se oznakama *public* i *private*. Ista oznaka sme i više puta da se koristi. Početni deo klase, pre prve oznake public, je privatan. Članovi klase navedeni u privatnim delovima su privatni članovi, a članovi navedeni u javnim delovima javni članovi. Svi članovi klase su podrazumevano privatni - po *default_u* .

7.1. Sagledti pristup javnim članovima jednostavne klase.

```
// Program demonstrira deklaraciju klase i
// definiciju objekta klase.

#include <iostream.h>     // za cout

class Cat                       // deklarise objekat klase
{
public:
        int itsAge;             // clanovi koji slede su javni
        int itsWeight;
};

void main()
{
Cat Frisky;
Frisky . itsAge =5;             // dodeli promenljivoj clanici
cout << "Frisky je macka koja ima " ;
cout << Frisky . itsAge << "godina.\n";
}
```

IZLAZ:

Frisky je macka koja ima 5 godina.

Linija 6 programa sadrži ključnu reč *class*. Ovo govori kompajieru 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 klju~čnu 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 itsWeight.

Linija 14 započinje glavnu funkciju programa. Frisky 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.

NAPOMENA: Probajte da pretvorite u komentar liniju 8 i probajte do ponovo kompajlirate. Dobićete grešku u liniji 17, jer itsAge više neće imati javni pristup.

7.2. Sagledti pristup javnim i privatnim članovima jednostavne klase, uz definiciju metoda klase.

```
//Implementacija metode jednostavne klase
//Demonstrira deklaraciju klase i
// definiciju metoda klase.

#include <iostream.h>

class Cat // zapocinje deklaraciju klase
{
public: // zapocinje javnu sekciju

        int GetAge(); // funkcija pristupa
        void SetAge (int age); // funkcija pristupa
        void Meow(); // opsta funkcija
private: // pocinje privatnu sekciju
        int itsAge; // promenljiva clanica
};

//GetAge, Javna funkcija pristupa
//vraca vrednost clana itsAge
int Cat::GetAge()
{
return itsAge;
}

// definicija SetAge, javna
//funkcija pristupa
//postavlja clan itsAge
void Cat::SetAge(int age)
{
//postavi promenljivu clanicu njegovu starost na
//vrednost koja je predata 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
// mijauk, saopstava nam njenu starost, a onda ponovo pravi //mijauk.
int main()
{
```

```
Cat Frisky;
//Frisky.SetAqe(5);
Frisky.Meow();
cout << "Frisky je macka koja ima " ;
cout << Frisky.GetAge() << " godina.\n.";
Frisky.Meow();
return 0;
}
```

IZLAZ:
Mijau.
Frisky je macka koja ima 5 godina.
Mijau.

7.3. Sagledti predhodni primer samo uz upotrebu konstruktora i destruktora.

```
//Implementacija metode jednostavne klase
//demonstrira deklaraciju klase i
// definiciju metoda klase.
//Upotreba konstruktora za inicijalizaciju objekata
//Upotreba destruktora za uklanjanje objekata i oslobadjanje memorije
#include <iostream.h>

class Cat // zapocinje deklaraciju klase
{
public: // zapocinje javnu sekciju
        Cat(int initialAge);//konstruktor
    ~Cat();//destruktor
        int GetAge(); // funkcija pristupa
        void SetAge (int age); // funkcija pristupa
        void Meow(); // opsta funkcija
private: // pocinje privatnu sekciju
        int itsAge; // promenljiva clanica
};
// konstruktor klase Cat
Cat::Cat(int initialAge)
{
itsAge=initialAge;
}

Cat::~Cat()//destruktor koji ne radi nista
{
}

//GetAge, Javna funkcija pristupa
//vraca vrednost clana itsAge
int Cat::GetAge()
```

```
{
return itsAge;
}

// definicija SetAge, javna
//funkcija pristupa
//postavlja clan itsAge
void Cat::SetAge(int age)
{
//postavi promenljivu clanicu njegovu starost na
//vrednost koja je predata 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
// mijauk, saopstava nam njenu starost, a onda ponovo pravi //mijauk.
int main()
{
Cat Frisky(5);
Frisky.Meow();
cout << "Frisky je macka koja ima " ;
cout << Frisky.GetAge() << " godina.\n.";
Frisky.Meow();
Frisky.SetAge(7);
cout << "Sada Frisky ima " ;
cout << Frisky.GetAge() << " godina.\n.";
Frisky.Meow();
return 0;
}
```

7.4. Korišćenje konstruktora i destruktora u klasi Tree.

```
//: C06:Constructor1.cpp
// Constructors & destructors
#include <iostream.h>

class Tree {
int height;
public:
Tree(int initialHeight); // Constructor
~Tree(); // Destructor
```

```
void grow(int years);
void printsize();
};
Tree::Tree(int initialHeight) {
height = initialHeight;
}
Tree::~Tree() {
cout << "inside Tree destructor" << endl;
printsize();
}
void Tree::grow(int years) {
height += years;
}
void Tree::printsize() {
cout << "Tree height is " << height << endl;
}
int main() {
cout << "before opening brace" << endl;
{
Tree t(12);
cout << "after Tree creation" << endl;
t.printsize();
t.grow(4);
cout << "before closing brace" << endl;
}
cout << "after closing brace" << endl;
}
```

```
//Here's the output of the above program:
//before opening brace
//after Tree creation
//Tree height is 12
//before closing brace
//inside Tree destructor
//Tree height is 16
//after closing brace
```

OSMA LABORATORIJSKA VEŽBA IZ PROGRAMSKOG JEZIKA C++

Napisati kompletan kod za primer apstrakcija Osoba, Zena i Maloletnik iz skripta.

```
// Primer. 1: Osnovni koncepti: klase, konstruktori,
//nasledjivanje, polimorfizam
#include <stdio.h>
// Klasa: Osoba
class Osoba
{ public:
        Osoba(char* ime, int godine);
        virtual void koSi();
protected:
char* ime;
int god;
};
Osoba::Osoba (char* i, int g)
{
ime=i ;
god=((g>=0 && g<=100)?g:0);
}
void Osoba::koSi()
{
printf("Ja sam %s i imam %d godina.\n",ime,god);
}
// Klasa: Maloletnik
class Maloletnik : public Osoba
{
public:
        Maloletnik(char* ime,char* staratelj,int godine);
        void koOdgovara();
private:
        char* staratelj ;
};
Maloletnik::Maloletnik(char* i , char*s,int g):Osoba(i,g)
{
staratelj=s;
}
void Maloletnik::koOdgovara ()
{
printf("Ja sam %s i za mene odgovara %s.\n",ime,staratelj);
}
// Klasa: Zena
class Zena : public Osoba
```

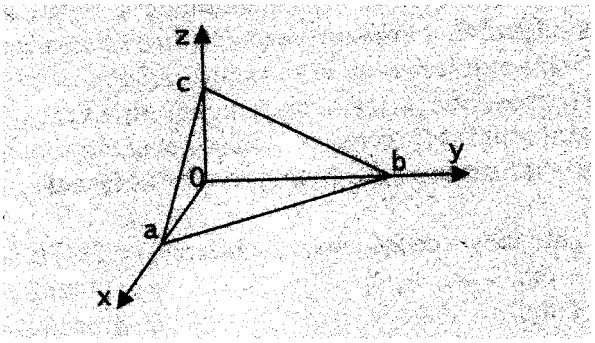
```
{
public:
        Zena(char* ime, char* devojacko, int godine);
        virtual void koSi();
private:
        char* devojacko;
};
Zena::Zena (char *i,char *d,int g) : Osoba(i,g)
{
devojacko=d;
}
void Zena::koSi ()
{
printf("Ja sam %s, devojacko prezime %s.\n",ime,devojacko);
}
// Funkcija: ispitaj
void ispitaj (Osoba *hejTi)
{
hejTi->koSi() ;
}
// Glavni program
void main ()
{
Osoba otac("Petar Petrovic",40);
Zena majka("Milka Petrovic","Mitrovic",35);
Maloletnik dete("Milan Petrovic","Petar Petrovic",12);
ispitaj(&otac) ;
ispitaj (&majka) ;
ispitaj (&dete) ;
dete.koOdgovara() ;
}
```

Zadatak :
Sačiniti jednostavnu realizaciju klase kompleksnih brojeva.

```
// Primer 2: osnovni koncepti
#include <stdio.h>
// Klasa: Complex
class Complex
{
public:
        Complex(float real, float imag) ;
        Complex add(Complex);
        Complex sub(Complex);
float Re();
float Im();
void print();
private:
float real,imag;
```

```
};
Complex::Complex (float r, float i){real=r; imag=i;}
Complex Complex::add (Complex c)
{
return Complex(real+c.real,imag+c.imag) ;
}
Complex Complex::sub (Complex c)
{
return Complex(real-c.real,imag-c.imag) ;
}
float Complex::Re() {return real;}
float Complex::Im() {return imag;}

void Complex::print()
{
printf("(%f,%f)",real,imag) ;
}

// Glavni program
void main()
{
Complex c1(3.5 ,-17.5),c2(-3.5,17.5),c3(0,0) ;
c3=c1.add(c2);
c1=c2.sub(c3);
printf ("C1=") ; c1.print(); printf ("\n") ;
printf ("C2=") ; c2.print(); printf ("\n") ;
printf ("C3=") ; c3.print(); printf ("\n") ;
}
```

Zadatak:
Definisati klase "tačka u prostoru - Point" i "piramida - Piramid" sa potrebnim podacima i funkcijama članicama. Napisati program u C++ jeziku koji za n tačaka u prostoru ustanovljava da li se nalaze na piramidi predstavljenoj crte`om:



```
// Program Zadl19.cpp
#include <iostream.h>
class Point {public:
void Read ( ) ;
double GetX() const;
double GetY() const;
double GetZ() const;
private:
double x, y, z;
} ;
void Point::Read()
{cout << "x= "; cin >> x;
cout << "y= "; cin >> y;
cout << "z= "; cin >> z; }
double Point::GetX() const
{return x; }
double Point::GetY() const
{return y; }
double Point::GetZ() const
{return z; }

class Piramid
{public:
void Read();
bool IsPointIn(Point const &p) const;
private:
double a, b, c; } ;
void Piramid::Read()
{cout << "a= "; cin >> a;
cout << "b= "; cin >> b;
cout << "c= "; cin >> c; }
bool Piramid::IsPointIn(Point const &p) const
{return p.GetX()>=O && p.GetY()>=O && p.GetZ()>=O &&
p.GetX()/a + p.GetY()/b + p.GetZ()/c <=1;}

int main()
{Piramid p;
cout << "Input piramid \n";
p.Read() ;
Point pt[100];
cout << "number of points: ";
int n; cin >> n;
for (int i=O; i<=n-l; i++)
{cout << "Point " << i << ": \n";
pt[i].Read(); }
i = 0;
while (i<=n-2 && p.IsPointIn(pt[i])) i++;
if(p.IsPointIn(pt[i])) cout << "Yes\n"; else cout <<"No\n";
return 0; }
```

Zadatak:

Definisati klasu "Tačka u ravni - Point" sa dve članice podataka - dve dekartove koordinate tačke i pripadajućom funkcijom članicom. Napisati program u C++ jeziku koji:

a) za n unetih tačaka u ravi svakoj vrši translaciju za (2, 4) i štampa translirane tačke,

b) izračunava rastojanje između svake dve tačke (sve jedno da su stare ili nove).

```
// Program Zadl18.cpp
#include <iostream.h>
#include <math.h>
class point
{private:
double x;
double y;
public:
point() ;
point(double, double);
void read();
void move(double, double);
double get_x() const
{return x; }
double get_y() const
{return y; }
void print() const;
} ;
point::point()
{x = 0;
y = 0; }
point::point(double a, double b)
{x = a;
y = b; }
void point::read()
{cout << "x= ";
cin >> x;
cout << "y= ";
cin >> y; }
void point::print() const
{cout << "(" << x << ", " << y << ")" << endl; }
void point::move(double dx, double dy)
{x = x + dx;
y = y + dy; }
double dist(point X, point Y)
{return sqrt(pow(Y.get_x()-X.get_x(), 2) +
        pow(Y.get_y()- X.get_y(), 2));
}                1
int main()
{// a)
cout << "n= ";
int n;
cin >> n;
```

```
point table[10];
for (int i=0; i<=n-1; i++)
table[i].read();
for (i=0; i<=n-1; i++)
table[i].print() ;
cout << endl;
for (i=0; i<=n-1; i++)
{table[i].move(2,4) ;
table[i].print(); }
//b)
for (i=0; i<=n-2; i++)
for (int j=i+1; j<=n-1; j++)
cout << dist(table[i], table[j]) << endl;
return 0;
}
```

DODATNA VEŽBA IZ PROGRAMSKOG JEZIKA C++

9.1 Obrada tačaka u ravni

Zadatak:

Projektovati na jeziku *C++* klasu sa elementarnim operacijama nad tačkama u ravni. Sastaviti na jeziku *C++* glavni program za prikazivanje mogućnosti te klase.

Rešenje a:

Program predstavlja definiciju jedne jednostavne klase za obradu tačaka u ravni. Kao privatni atributi postoje *Dekartove* koordinate tačke x i y.
Od javnih metoda na prvom mestu nalazi se metoda tacka ( ) za stvaranje tačke od koordinata koje su tipa double. Ova radnja je neka vrsta konverzije tipa iz osnovnih tipova u tipove koje je definisao korisnik.

Sličnu ulogu imaju, ali u suprotnom smeru, metode aps ( ) za nalaženje apscise i ord () za nalaženje ordinate zadate tačke. Pomoću njih može da se dolazi do delova sadržaja tačke kao složenog objekta. U širem smislu, tu mogu da se svrstavaju i metode poteg ( ) za nalaženje udaljenosti tačke od koordinatnog početka i nagib ( ) za nalaženje nagiba (ugla) potega u odnosu na x-osu. Ova dva podatka su, u stvari, polarne koordinate posmatrane taćke.

```
// Definicija klase tacaka u ravni (Tacka).
class Tacka {
double x, y;
public:
void tacka (double a,double b) {x = a;y = b;} // Stvaranje tacke.
double aps () const { return x; } // Apscisa tacke.
double ord ( ) const { return y; } // Ordinata tacke.
double poteg ( ) const ; //Odstojanje od koordinatnog pocetka.
double nagib ( ) const ; // Nagib potega u odnosu na x-osu .
double rastojanje (Tacka) const ; // Odstojanje od zadate tacke.
Tacka najbliza (const Tacka *,int) const ;
// Najbliza tacka u nizu tacaka.
void citaj();        //Citaj  tacku.
void pisi() const; // Pisi tacku.
};
```

Program 9.1 -Definicija klase tačaka u ravni (*tacka1. h)*

U klasi Tacka postoje dve metode koje među argumentima imaju objekte tog tipa. Metoda rastojanje() nalazi rastojanje između zadate tačke i tekuće tačke, dok metoda najbliza( ) nalazi najbližu tačku u zadatom nizu tačaka tekućoj tački. Vrednost ove metode je pronađena tačka (tačnije, kopija pronađene tačke), dakle objekat tipa Tacka.

Na kraju, metode citaj ( ) i pisi ( ) predviđene su za ulaznu i izlaznu konverziju vrednosti objekata tipa Tacka. Metoda citaj ( ) čita preko glavnog ulaza računara koordinate

x i y i smešta ih u odgovarajuće članove tekuće tačke. Metoda pisi ( ) ispisuje na glavnom izlazu računara koordinate tekuće tačke izrneđu para okruglih zagrada, međusobno razdvojenih zarezom.

Od svih metoda samo dve menjaju vrednost objekta za koji su pozvani, tacka ( ) i citaj ( ). Kod svih ostalih metoda dodat je modifikator const iza spiska argurnenta da bi se označilo da ne menjaju vrednost svojih skrivenih argumenata.

Pošto definicija klase mora da stoji na raspolganju prevodiocu prilikom prevođenja bilo kog programskog modula u kome se koristi data klasa, tekst programa 9.1 treba da se smešta u zaglavlje (datoteku .h). U ovom slučaju to je datoteka tacka1. h.

Za neke od gore nabrojanih metoda u programu 9.1 navedene su definicije. Za njih se, s toga, podrazumeva neposredno ugrađivanje u kod. To su one, stvarno, najjednostavnije: tacka ( ) ,aps ( ) i ord ( ). Prevodi njihovog sadržaja sastoje se od svega jedne ili dve mašinske naredbe. Bilo bi vrlo neefikasno uobličavati ih u prave funkcije.

Skreće se pažnja, da sve metode imaju i jedan skriveni argument, tekuću tačku. Koordinate x i y predstavljaju atribute te tekuće tačke, tj. this->x i this->y.

Definicije preostalih metoda su izdvojene u zasebnu datoteku, čiji sadržaj treba odvojeno prevoditi. Te definicije su prikazane u programu 9.2. Pošto se definicije nalaze izvan dosega klase Tacka, ispred identifikatora svake metode dodato je Tacka : : .

Pošto neke od funkcija u programu 9.2 koriste matematičke funkcije, u prevođenje je uključeno zaglavlje za matematičke funkcije <math. h>. Zaglavlje <iostream.h> je potrebno zbog ulazno-izlaznih operacija u metodama citaj ( ) i pisi ( ). Na kraju, i definiciju klase Tacka trebalo je staviti na raspolaganje prevodiocu iz zaglavija "tacka1.h".

```
// Definjcije metoda klase Tacka.
#include <math.h>
#include <iostream.h>
#include "tackal.h.
// Odstojanje tekuce tacke od koordinatnog pocetka .
double Tacka::poteg () const { return sqrt(x*x + y*y); }
// Nagib potega u odnosu na x-osu .
double Tacka::nagib() const {return(x==0&&y==0)? 0:atan2(y,x); }
// Odstojanje tekuce tacke od tacke a.
double Tacka::rastojanje (Tacka a) const
{return sqrt(pow(x-a.x,2)+ pow(y-a.y,2));}
// Najbliza tacka u nizu tacaka a u odnosu na tekucu tacku.
Tacka Tacka::najbliza (const Tacka *a, int n) const {
Tacka t = a[0]; double r, m = rastojanje (t);
for(int i=1;i<n;i++) if((r=rastojanje(a[i]))<m){m=r;t=a[i];}
return t;
}
//Citanje koordinata tekuce tacke preko glavnog ulaza.
void Tacka::citaj() { cin >> x >> y; }
// pisanje koordinata tekuce tacke preko glavnog izlaza .
void Tacka::pisi() const {cout<<'(' << x <<','<< y <<')'; }
```

Program 9.2 -Definicije metoda klase Tacka (*tacka1.c)*

Sve metode u programu 9.2, sa izuzetkom metode najbliza( ), su dovoljno male da ne bi bilo nerazumno zahtevati njihovo neposredno ugrađivanje u kod. S druge strane, pošto svaka od njih poziva bar još neku drugu funkciju, priloženo rešenje nije nerazumno neefikasno.

Metoda rastojanje() ima jedan argument tipa Tacka, naravno pored skrivenog argumenta *this, koji je istog tipa i predstavlja tekuću tačku čije se rastojanje odargumenta traži. Izraz x-a.x se, dakle, tumači kao (this->x) -(a .x) . Prvi vidljivi argument metode najbliza ( ) je niz tačaka koji je predstavljen pokazivačem a na objekat tipa Tacka. Kao obično, adresa i-tog elementa niza je a+i, a sam element može da se označi sa a (i). U toj metodi defie se i lokalni objekat t tipa Tacka koji se inicijalizuje početnim elementom niza tačka (Tacka t=a [0];). Na kraju, skreće se pažnja da se i za metode podrazumeva tekući objekat. To znači da pozivanje metode rastojanje (...) tumači se kao this->rastojanje (...) .

Funkcije (metode) definisane u programu 9.2 prevode se nezavisno. Korisniku klase Tacka dovoljno je isporučiti samo dobijeni prevedeni oblik, izvorni tekst mu nije potreban. Potrebni su i dovoljni samo prototipovi koji se nalaze u zaglavlju (program 9.1). Ovo omogućava izmenu ostvarenja date klase bez potrebe za izmenama u prograrnima koji koriste tu klasu, sve dok se ta ne promeni u javnom delu te klase. U posmatranom primeru, atributi x i y mogli bi da se zamene atributima r i fi koji bi predstavljali polarne koordinate tačke.

Korisnik klase ta od toga ne bi primetio. To znači da ta ne bi trebalo da se izmeni u izvomim tekstovima programa koji koriste klasu Tacka. Bez obzira na to, svi korisnikovi programi trebalo bi ponovo da budu prevedeni u slučajevima izmena u ostvarenju klase Tacka.

Program 9.3 predstavlja primer za koršćenje klase Tacka.

```
// Primer koriscenja klase Tacka.
#include <iostream.h>
#include"tacka1.h"
int main() {
int n; cout << "\nBroj tacaka? "; cin >> n;
Tacka *niz = new Tacka [n] ;
cout << "Niz tacaka? ";for(int i=0;i<n;i++)niz[i].citaj();
double x, y; cout <<"\nReferentna tacka? " ;cin >> x >> y;
Tacka t; t.tacka (x, y);
cout <<"Koordinate:(" << t.aps()<<','<<t.ord()<<")\n";
cout<<"Poteg i nagib:"<<t.poteg()<<','<<t.nagib()<<endl;
Tacka w = t.najbliza (niz, n);
cout <<"\nNajbliza tacka: "; w.pisi(); cout<< endl;
cout<<"Udaljenost od referentnetacke:"<<t.rastojanje(w)<<endl; delete [] niz;
return 0 ;}
```

Program 9.3 -*Primer korišćenja klase Tacka* (*tacka1t* .cpp)

Posle čitanja broja tačaka n, operatorom new traži se dodela memorije u dinamičkoj zoni za niz od n objekata tipa Tacka. Rezultat operatora dodeljuje se pokazivaču niz na objekte tipa Tacka.

U ciklusu za čitanje podataka o tačkama, pozivanjem metode citaj ( ) izrazom niz [i].citaj ( ), niz [i] postaje tekuća tačka (objekat) te metode.

Posle toga čitaju se koordinate jedne referentne tačke kao dva realna broja x i y od kojih se pozivom t .tacka (x, y) stvara sadržaj tačke t. Ta tačka se u nastavku koristi kao tekuća tačka pri pozivanju metoda aps(), ord(), poteg() i nagib(). Svi ti pozivi koriste se kao operandi operatora << za ispisivanje na glavnom izlazu.

Tačka w inicijalizuje se tačkom koja je u nizu tačaka niz najbliža tački t. Taj podatak dobija se kao vrednost metode najbliza ( ) za tekuću tačku t. Rezultujući sadržaj

tačke w ispisuje se na glavnom izlazu pomoću w.pisi ( ). Zatim se korišćenjem izraza

t.rastojanje (w) kao operand operatoru << ispisuje veličina tog najmanjeg rastojanja.
Na kraju programa, iz principijelnog razloga, operatorom delete oslobađa se memorija u dinamičkoj zoni, koja je na početku programa bila dodeljena nizu tačaka. Ovo nije neophodno pred sam kraj programa. Pri završetku programa ionako se sva dodeljena memorija u dinamičkoj zoni oslobađa automatski.

Rezultat 9.1 prikazuje primer rada programa 9.3.

```
Broj tacaka? 5
Niz tacaka? 1 2   3 4   5 1   3 2   6 3
Referentna tacka ? 3 5
Koordinate: (3, 5)
Poteg i nagib: 5.830952, 1.03.0377 i
Najbliza tacka: (3,4)
Udaljenost od referentne tacke: 1
```

Ršenje b: Moguće je sve spojiti u jedan program, kompajlirati i linkovati zajedno i izvršiti.

```
class Tacka {
 double x, y;
public:
void tacka (double a,double b) {x = a;y = b;} // Stvaranje tacke.
double aps () const { return x; } // Apscisa tacke.
double ord ( ) const { return y; } // Ordinata tacke.
double poteg ( ) const ; //Odstojanje od koordinatnog pocetka.
double nagib ( ) const ; // Nagib potega u odnosu na x-osu .
double rastojanje (Tacka) const ; // Odstojanje od zadate tacke.
Tacka najbliza (const Tacka *,int) const ;
// Najbliza tacka u nizu tacaka.
void citaj();         //Citaj  tacku.
void pisi() const; // Pisi tacku.
};

// Definjcije metoda klase Tacka.
#include <math.h>
#include <iostream.h>
//#include "tackal.h.
// Odstojanje tekuce tacke od koordinatnog pocetka .
double Tacka::poteg () const { return sqrt(x*x + y*y); }
// Nagib potega u odnosu na x-osu .
double Tacka::nagib() const {return(x==0&&y==0)? 0:atan2(y,x); }
// Odstojanje tekuce tacke od tacke a.
double Tacka::rastojanje (Tacka a) const
{return sqrt(pow(x-a.x,2)+ pow(y-a.y,2));}
// Najbliza tacka u nizu tacaka a u odnosu na tekucu tacku.
Tacka Tacka::najbliza (const Tacka *a, int n) const {
Tacka t = a[0]; double r, m = rastojanje (t);
for(int i=1;i<n;i++) if((r=rastojanje(a[i]))<m){m=r;t=a[i];}
return t;
}


//Citanje koordinata tekuce tacke preko glavnog ulaza.
```

```
void Tacka::citaj() { cin >> x >> y; }
// pisanje koordinata tekuce tacke preko glavnog izlaza .
void Tacka::pisi() const {cout<<'(' << x <<','<< y <<')'; }

// Primer koriscenja klase Tacka.
#include <iostream.h>
//#include"tacka1.h"
int main() {
int n; cout << "\nBroj tacaka? "; cin >> n;
Tacka *niz = new Tacka [n] ;
cout << "Niz tacaka? ";for(int i=0;i<n;i++)niz[i].citaj();
double x, y; cout <<"\nReferentna tacka? " ;cin >> x >> y;
Tacka t; t.tacka (x, y);
cout <<"Koordinate:(" << t.aps()<<','<<t.ord()<<")\n";
cout<<"Poteg i nagib:"<<t.poteg()<<','<<t.nagib()<<endl;
Tacka w = t.najbliza (niz, n);
cout <<"\nNajbliza tacka: "; w.pisi(); cout<< endl;
cout<<"Udaljenost od referentnetacke:"<<t.rastojanje(w)<<endl; delete [] niz;
return 0 ;}
```

Zadatak 9.4. Napisati program koji defie sledeće karakteristike računara: ime modela (name), cenu (price) i ocenu (score) između 1 i 100. Kada se obrade informacije izveštaj treba da bude sortiran u opadajućem nizu ocena/cena.

Prva dva nivoa abstrakcije definisaćemo kroz klasu product, a primitivne operacije definisaćemo koroz sledeće funkcije članice:

void read(); - za unošenje informacija o računaru (baza podataka)
void print() const; - za štampanje informacija o računaru (baza podataka)
bool is_better_from(product const &) const; - proverava da li tekući računar ima bolji odnos ocena/cena od onoga ukazanog formalnim parametrom
double get_price() const; - postavlja cenu računara
int get_score()const; - postavlja ocenu računara

Osnovni modul za sortiranje podataka neka je procedura sorttable.

```
// Program Zad120.cpp
#include <iostream.h>
#include <iomanip.h>
#include <string.h>

class product
{public:
void read();
void print() const;


is_better_from(product const &) const;
```

```
double get_price() const;
int get_score() const;
private:
char name[21];
double price;
int score; } ;
void sorttable(int n, product* []);
int main()
{cout << setprecision(4)<<setiosflags(ios::fixed);
product table[300];
product* ptable[300];
int n;
do
{cout << "number of products? " ;
cin >> n; }
while (n<1 || n>300);
int i;
for (i = 0; i <= n-1; i++)
{table[i].read();
ptable[i] = &table[i];
}
cout << "table: \n";
for (i = 0; i <= n-1; i++)
{table[i].print();
cout << endl; }
sorttable(n, ptable);
cout << "\n New Table: \n";
for (i = 0; i <= n-1; i++)
{ptable[i]->print();
cout << setw(7)
<< ptable[i]->get_score()/ptable[i]->get_price()
<<endl;
}
return 0; }

void product::read()
{cout << "name: ";
cin >> name;
cout << "price: ";
cin >> price;
cout << "score: ";
cin >> score; }

void product::print() const
{cout << setw(25) << name
<< setw(10) << price
<< setw(12) << score; }

product::is_better_from(product const & x) const
{return score/price > x.score/x.price; }


double product::get_price() const
```

```
{return price; }
int product::get_score() const
{return score; }

void sorttable(int n, product* a[])
{for (int i = 0; i <= n-2; i++)
{int k = i;
product* max = a[i];
for (int j = i+1; j <= n-1; j++)
if (a[j]->is_better_from(*max))
{max = a[j];
k = j; }
max = a[i]; a[i] = a[k]; a[k] = max; }
}
```

DODATNA VEŽBA IZ PROGRAMSKOG JEZIKA C++

10.1. Primer konstruktora osnovne klase

```
#include <iostream.h>
class base
{private: int a1;
protected: int a2;
public:
base()
{cout << "constructor base() \n";
a1 = 0;
a2 = 0; }
base(int x, int y)
{cout << "constructor base(" << x << "," << y << ")\n";
a1 = x;
a2 = y;}
void a3()
{cout << "a1: " << a1 << endl
<< "a2: " << a2 << endl;}
};
class der : public base
{private: base d1;
protected: base d2;
public:
der(int x, int y) : base(x, y)
{cout << "constructor der\n"; }
void d3()
{d1.a3() ;
d2.a3 () ;
cout << "a2: " << a2 << endl;
cout << "a3():" << endl;
a3();}
} ;
void main()
{der x(1, 2);
x.d3() ; }
```

10.2. Primer destruktora osnovne klase

Program 10.2 prikazuje primer klase sa destruktorom za znakovne nizove. Pošto su znakovni nizovi međusobno vrlo različitih dužina, objekti za njihovo predstavljanje obično sadrže samo pokazivač na sam tekst. To je učinjeno i u ovom primeru.

Jedini atribut klase String je pokazivač na sam niz znakova. Zbog toga se prilikom inicijajizacije (konstruktorom String (char*)) vrši dodela memorijskog prostora potrebne

veličine i kopiranje niza znakova u tako dodeljenu memoriju. Zadatak destruktora (~String

( ) je da oslobodi taj memorijski prostor prilikom utavanja objekta.

Podrazumevani konstruktor (String ()) osigurava da svaki objekat tipa String bude inicijalizovan do te mere da bi kasnije smelo da se na njega primeni destruktor. Naime, posledice primene operatora **delete** na pokazivač slučajnog sadržaja su nepredvidljive, ali primena operatora **delete** na pokaziva ~ NULL (0) je uvek bezbedna.

Naredba niz=0; u destruktoru nije potrebna ako se destruktor poziva samo implicitno prilikom utavanja objekata. Međutim, poželjno je da objekat bude ostavljen u " ispravnom praznom " stanju zbog eventualnog eksplicitnog pozivanja destruktora. Tada će objekat, verovatno, i dalje postojati pa bi moglo da zasmeta ako pokazivački atribut pokazuje na nešto u dinamičkoj zoni memorije što više ne postoji.

Konstruktor kopije (String(String&)) samo treba da prekopira celokupan sadržaj svog pravog argumenta u skriveni argument (to je argument koji se inicijalizuje). Pošto se vrši stvaranje novog objekta, ne pretpostavlja se ta o zatečenom sadržaju parčeta memorije koje se inicijalizuje.

U glavnom programu na kraju programa, prvo se stvara objekat pozdrav tipa String koji se inicijalizuje običnim znakovnim nizom (char*). Tu će se pozivati konstruktor String(char*) koji je, u stvari, konverzija iz tipa **char*** u tip String. Posle toga, objekat a tipa String se inicijalizuje kopijom sadržaja objekta pozdrav pomoću konstruktora kopije String (String&) , dok objekat b se podrazumevanim konstruktorom String ( ) inicijalizuje kao "prazan" objekat. Ono što je bitno, atribut niz u njemu se postavlja na nulu, pa implicitno pozivanje destruktora (b.~String()) na kraju programa, neće da napravi nikakvu štetu.

```
class String {
char *niz; // Pokazivac na sam tekst.
public:
String () { niz = 0; } // Inicijalizacija praznog niza.
String (const char *) ; // Inicijalizacija nizom znakova.
String (const String &) ; // Inicijalizacija tipom String.
~String () ; // Unistavanje objekta tipa String.
void pisi () const; // Ispisivanje niza.
};
#include <string.h>
#include <iostream.h>
String:: String (const char *t)
{ if ( (niz = new char [strlen(t) +1] ) != 0) strcpy(niz,t) ;}
String::String (const String &s)
{if ((niz = new char [strlen(s.niz)+1]) != 0) strcpy (niz, s.niz); }
String::~String ()
{ delete [] niz; niz = 0; }
void String::pisi () const
{ cout << niz; }
int main () {
String pozdrav("Pozdrav svima") ; // Poziva se String (char *)
String a = pozdrav; // Poziva se String (String &).
String b; // Poziva se String ().
cout << "pozdrav = "; pozdrav.pisi (); cout << endl;
cout << "a = "; a.pisi () ; cout << endl;
return 0;}
// Ovde se poziva destruktor za sva tri objekta.
```

10.3. U definisanim klasama People, Student i PStudent sagledati upotrebu konstruktora i

destruktora.

```
// Program Zad178.cpp
#include <iostream.h>
#include <string.h>
//deklaracija klase People
class People
{public:
People(char * = "", char * = "");
void PrintPeople() const;
~People();
private:
char * name;
char * egn;
};
//definicija konstruktora klase People
People::People(char *str, char *num)
{name = new char[strlen(str)+1];
strcpy(name, str);
egn = new char[11];
strcpy(egn, num); }
//definicija metode PrintPeople
void People::PrintPeople() const
{cout << "Ime: " << name << endl;
cout << "Jedinstveni maticni broj: " << egn << endl; }
//definicija destruktora klase People
People::~People()
{cout << "~People() : " << endl;
delete name;
delete egn; }
//deklaracija klase Student
class Student : public People
{ public:
Student(char * = "", char * = "", long = 0, double = 0);
void PrintStudent() const;
~Student()
{cout << "~Student() : " << endl; }
private:
long facnom;
double usp; };
//definicija konstruktora klase Student
Student::Student(char *str, char * num, long facn,
double u) : People(str, num)
{facnom = facn;
usp = u; }
//definicija metode PrintStudent
void Student::PrintStudent() const
{PrintPeople();
cout << "Fac. nomer: " << facnom << endl;
cout << "Uspeh: " << usp << endl; }


// deklaracija klase PStudent
```

```
class PStudent : public Student
{ public:
PStudent(char * = "", char * = "", long = 0,
double = 0, double = 0);
~PStudent()
{cout << "~PStudent() \n"; }
void PrintPStudent() const;
protected:
double tax; } ;
//definicija konstruktora klase PStudent
PStudent::PStudent(char *str, char *num, long facn,
double u, double t) : Student(str, num, facn, u)
{tax = t; }
//definicija metode PrintPStudent
void PStudent::PrintPStudent() const
{PrintStudent();
cout << "Tax: " << tax << endl; }
void main()
{People pe;
pe.PrintPeople();
PStudent PStud("Bora Milosevic", "8206123422", 42444, 10.0, 4567);
PStud.PrintPStudent();
}
```

10.4. Obrada ure|enih skupova *Zadatak:*

Projektovati na jeziku *C++* klasu za obradu uređenih skupova realnih brojeva. Sastaviti na jeziku *C++* glavni program za prikazivanje mogućnosti te klase.

Re{enje:

Skupovi su zbirke podataka koji ne sadrže međusobno jednake vrednosti. Obično se smatra da redosled elemenata u skupu nije bitan. U ovom zadatku je uzeto da su elementi skupa uređeni što omogućava efikasnije izvodenje osnovnih operacija nad skupovima: nalaženje preseka, unije i razlike.

Prograrn 10.4. prikazuje definiciju klase Skup za obradu uređenih skupova realnih brojeva. Sadržaj skupa čine privatni atributi koji predstavljaju broj elemenata skupa *vel* i pokazivač na *niz* u dinamičkoj zoni memorije za smeštanje samih elemenata skupa.

U privatnom delu klase postoje još dve pomoćne metode za kopiranje sadržaja zadatog skupa u tekući objekat kopiraj( ) i za utavanje sadržaja tekućeg objekta brisi( ) . Njih će da pozivaju neke od javnih metoda k1ase. U većini k1asa koje koriste dinamičko dodeljivanje memorije za smeštanje delova sadržaja korisno je da se prave ovakve dve metode.

Treba obratiti pažnju na prenošenje skupa u metodu kopiraj ( ) (i u sve kasnije metode klase Skup) -pomoću upućivača (Skup&). Modifikator const označava da metoda ne menja vrednost upućivanog objekta. To je preporučljivi način prenošenja primeraka klasa u funkcije umesto prenošenja objekata (Skup), jer se time izbegne gubitak vremena i prostora za inicijalizaclju formalnog argumenta kopijom stvarnog argumenta uz pomoć konstruktora kopije. Pošto metoda za pražnjenje skupa brisi( ) može da se koristi na više mesta, a ne samo prilikom konačnog utavanja objekata, važno je obezbediti da po povratku iz te metode tekući objekat bude ispravan prazan objekat. To je u programu 10.4. učinjeno postavljanjem atributa niz i vel na nulu posle oslobađanja memorije u dinamičkoj zoni operatorom delete.

U javnom delu klase prvo su navedena tri konstruktora: podrazumevani konstruktor,

konstruktor konverzije i konstruktor kopije. Podrazumevani konstruktor samo postavlja vrednosti oba atributa na nule. Konstruktor konverzije stvara skup od jednog elementa, s tim da u slučaju neuspeha u dinamičkom dodeljivanju memorije prekida program sa završnim statusom jednakim 1. Konstruktor kopije vrednost svog argumenta kopira u tekući objekat pozivanjem već pomenute privatne metode kopiraj ( ) .Pošto su sva tri konstruktora vrlo jednostavna, definisana su unutar definicije klase, što znači da će biti ugrađivani neposredno u kod. Sledeći, praktično obavezni element klasa koje koriste dinamičko dodeljivanje memorije je destruktor. Destruktor samo poziva privatnu metodu brisi( ) , pa se i on neposredno ugrađuje u kod.

Iza destruktora slede metode presek(), unija() i razlika() za ostvarivanje odgovarajućih skupovnih operacija nad skupovima koji se navode kao argumenti (konstantni upućivači) i stavljaju rezulat u tekući objekat. Pored njih postoje još metode pisi( ) za ispisivanje sadržaja tekućeg objekta preko glavnog izlaza računara (cout) i za čitanje , skupa (broj elemenata i same elemente) preko glavnog ulaza računara (cin). Sve ove metode su relativno složene, pa su njihove definicije odložene za kasnije. Pošto metoda pisi( ) ne menja vrednost svog skrivenog argumenta, dodat je modifikator const na kraju njenog prototipa.

Poslednja metoda u klasi Skup je mala uslužna javna metoda velicina( ) za dohvatanje privatnog atributa vel, za slučaj da nekoga interesuje trenutni broj elemenata tekućeg skupa. Ovakve metode koje ne rade ta već samo dohvataju neki privatni atribut klase su vrlo često u praksi. Njihovo ostavrenje kao pravih funkcija bilo bi vrlo neefikasno, pa ih uvek treba pretvideti za neposredno ugrađivanje u kod.

Pored dohvatanja privatnih atributa, u praksi se često sreću i javne metode koje samo postavljaju vradnost po jednog privatnog atributa. Pored toga, po potrebi, one mogu i da proveravaju da li navedena vrednost može da se prihvati kao novi sadržaj atributa.

O metodi kopiraj( ) nema mnogo da se kaže. Posle uspešne dodele potrebne količine memorije (s.vel) u dinamičkoj zoni, prenose se elementi niza iz pravog argumenta (s.niz) u skriveni argument. Ako dodela memorije ne uspe, program se prekine (exit(l)).

Metoda presek( ) je već interesantnija. Prvo se defie lokalni skup s koji, u odsustvu inicijalizatora, inicijalizuje se podrazumevanim konstruktorom kao prazan skup. Pošto presek dva skupa sadži elemente koji se pojavljuju u oba skupa istovremeno, rezultat ne može da ima više elemenata od onoga koliko ima manji od ta dva skupa. Upravo se tolika memorija dodeljuje lokalnom skupu s. Pošto su po pretpostavci skupovi uređeni, njihov presek može da se obrazuje u jednom prolasku kroz elemente skupova. Ciklus traje dok se ne stigne do kraja jednog od skupova (nizova u kojima su elementi skupova). U svakom prolasku kroz ciklus upoređuju se tekući elementi oba skupa (s1. niz [i] prema s2 .niz [j]). Ako je element prvog skupa manji od elementa drugog skupa, isti može da se odbaci jer u nastavku drugog skupa se nalaze sarno još veći elementi. Slično, ako je element prvog skupa veći od elementa drugog skupa, element drugog skupa može da se odbaci. Ako su tekući elementi oba skupa medusobno jednaki, potrebno je tu zajedničku vrednost smestiti u rezultujući skup (s.niz [k++]) i preći na sledeće elemente oba početna skupa istovremenim povećavanjem indeksa i i j. Po izlasku iz ciklusa, k predstavlja veličinu rezultujućeg skupa i to se stavlja u alribut s.vel lokalnog skupa. Ta veličina će najčešće biti manja od prostora koji je dodeljen na početku metode. Preostaje još na se rezultat ozvaniči. Pre svega, potrebno je utiti zatečeni sadržaj u tekućem objektu (skrivenom argumentu) koji je ispravan skup i kao takav koristi i prostor u dinamičkoj zoni memorije koji više neće biti potreban. To se postiže pozivanjem privatne metode brisi( ). Potom se sadržaj lokalnog skupa s metodom kopiraj( ) prekopira u tekući objekat. Pošto ta metoda veličinu skupa određuje na osnovu vrednosti atributa vel svog argumenta (u ovom slučaju to je s.vel), kopiji će dodeliti tačno toliko memorije u dinamičkoj zoni koliko je mesta u nizu s.niz popunjeno, a ne koliko je prostora dodeljeno na osnovu procene očekivane veličine rezultujućeg skupa. Na kraju,

prilikom povralka iz metode presek( ), pošto se tada napušta doseg lokanog prolaznog

objekta s, destruktor će da oslobodi ceo prostor u dinamičkoj zoni memorije dodeljen objektu s, a ne samo s.vel komponenata (što inače fizički nije moguće). Na taj način trajno zauzeće memorije biće upravo toliko koliko je potrebno za smeštanje elemenata rezultujućeg skupa, bez obzira što njihov broj unapred nije mogao da se tačno odredi pa je privremeno zauzet nešto veći prostor. Još jedna završna primedba: Ovakvo napravljena metoda presek( ) omogućava da se rezullat operacije stavi u treći skup (c.presek(a,b)), ali može da se stavlja i u jedan od početnih skupova (a.presek(a,b) ili b.presek(a,b), pošto se početni sadržaj odredišnog skupa (skrivenog argumenta metode) utava tek kada je već rezultat napravljen u lokalnom objektu s.

Osnovna ideja metoda unija( ) i razlika( ) je slična metodi presek( ) pa se njihova detaljna analiza prepušta vama. Metoda pisi( ) na uobičajeni način ispisuje elemente skupa, između para vitičastih zagrada međusobno razdvojenih zarezima (na primer: { 1 , 3 , 4 .6 }).

Poslednja metoda citaj( ) prvo uti stari sadržaj tekućeg objekta i posle pročita broj elemenata u lokalnu promenljivu vel (koja sakriva atribut vel!) i same elemente skupa. Elementi se jedan po jedan čitaju u lokalnu promenljivu broj odakie se metodom unija( ) stavljaju u tekući objekat. Argumenti metode su tekući objekat (*this) i rezultat konverzije sadržaja promenljive broj tipa double u skup od jednog elementa automatskim pozivanjem konstruktora konverzije (Skup (double)). Na ovaj način rezultat čitanja će sigurno biti uređeni skup, bez obzira da li korisnik unese korektan niz vrednosti. Naravno, ako među unetim vrednostima bude medusobno jednakih, broj elemenata skupa neće biti jednak najavljenom broju na početku metode citaj( ). Ono što je najvažnije da je obezbeđeno da ni na koji način ne može da se pojavi objekat tipa Skup koji nema sva obeležja tog skupa. Prikazano rešenje je dosta nefikasno imajući na umu šta se sve radi dok se nađe unija dva skupa, ali je ovako najlakše za programiranje. Uz malo više programerskog truda moglo bi da se ostvari uređivanje niza čitanih brojeva i preskakanje eventualnih duplikata u samoj metodi citaj(), bez pozivanja drugih metoda. To bi svakako dovelo do manjeg utroška procesorskog vremena uz nešto većeg utroška memorije zbog dodatnog koda.

U programu za prikazivanje mogućnosti klase Skup, čitaju se parovi skupova i nalaze se redom njihovi preseci, unije i razlike pozivanjem odgovarajućih metoda. Dobijeni rezultati se ispisuju pozivanjem metode pisi ( ). Skreće se pažnja da se svi skupovi stvaraju kao prazni skupovi, pa se njihov sadržaj menja kao rezultat pozivanja metoda klase Skup. Sadržaj skupa s se menja čak tri puta. Takode, važno je uočiti da se na kraju svakog prolaska kroz ciklus napušta doseg identifikatora definisanih skupova, pa se automatskim pozivanjem destruktora njihov sadržaj utava, tj. oslobađa se memorija koja je u dinamičkoj zoni dodeljena za smeštanje elemenata skupa.

Rezultat 4.10. prikazuje primer rada programa sa dva kompleta ulaznih podataka. U prvom kompletu oba uneta niza podataka su stvarno uređeni skupovi. Napominje se da u svakom nizu brojeva prvi predstavlja broj elemenata niza, a tek od drugog broja počinju vrednosti namenjene za umetanje u skup.

U drugom kompletu me|u podacima se nalaze i brojevi sa me|usobno jednakim vrednostima, a brojevi nisu ni po rastu}em redosledu svojih vrednosti. Bez obzira na to, rezultati njihovih ~itanja su pravi uredeni skupovi.

```
// Definicija klase za uredjene skupove (Skup)
#include<stdlib.h>
#include<mem.h>
class Skup {
int vel ; double *niz ; // Velicina i elementi skupa
void kopiraj (const Skup &); // Kopiranje skupa.
void brisi() {delete [] niz;niz=0;vel=0;} // Praznjenje skupa.


public:
```

```
Skup () { niz = 0; vel = 0; } // Stvaranje praznog skupa.
Skup (double a) { // Konverzija broja u skup.
if ((niz = new double [vel = 1]) == 0) exit (1);
niz[0] = a; }
Skup (const Skup &s) { kopiraj (s) ; } // Inicijalizacija skupom.
~Skup() { brisi (); } // Unistavanje skupa.
void presek (const Skup &, const Skup &) ; // Presek dva skupa.
void unija (const Skup &, const Skup &) ; // Unija dva skupa.
void razlika (const Skup & ,const Skup &) ; // Razlika dva skupa
void pisi () const; // Ispisivanje skupa.
void citaj (); // citanje skupa.
int velicina () const { return vel ; } // Velicina skupa .
}
// Definicije metoda klase Skup
//#include "skup.h"
#include <iostream.h>
void Skup::kopiraj (const Skup &s) { // Kopiranje skupa.
if ((niz = new double [vel = s.vel]) == 0) exit (1);
for (int i=0; i<vel; i++) niz[i] = s.niz[i]; }
void Skup::presek(const Skup &s1,const Skup &s2){//Presek dva
Skup s;
if ((s.niz=new double [s1.vel<s2.vel?s1.vel:s2.vel])==0) exit(1);
for (int i=0, j=0, k=0; i<s1.vel && j<s2.vel; )
    if (s1.niz[i] < s2.niz[j] ) i++;
else if (s1.niz[i] > s2.niz[j]) j++;
else s.niz[k++] = s1.niz[j++,i++];
s.vel = k; brisi (); kopiraj (s) ; }
void Skup::unija (const Skup &s1,const Skup &s2){// unija dva
Skup s;
if ((s.niz=new double [s1.vel+s2.vel]) == 0) exit(1);
for (int i=0, j=0, k=0; i<s1.vel || j<s2.vel; )
s.niz[k++] = (i<s1.vel && j<s2.vel)
?(s1.niz[i]<s2.niz[j] ? s1.niz[i++] :
 s1.niz[i]>s2.niz[j] ? s2.niz[j++] :
s1.niz[j++,i++]):
(i < s1.vel ? s1.niz[i++] : s2.niz[j++]) ;
s.vel = k; brisi () ; kopiraj (s) ; }
void Skup::razlika(const Skup &s1, const Skup &s2){// Razlika dva
Skup s ;
if ((s.niz = new double [s1.vel]) == 0) exit (1);
for (int i=0, j=0, k=0; i<s1.vel; )
if (j >= s2.vel) s.niz[k++] = s1.niz[i++];
else if (s1.niz[i] < s2.niz[j] ) s.niz[k++] = s1.niz[i++] ;
else if (s1.niz[i] > s2.niz[j]) j++;
else { i++; j++; }
s.vel = k; brisi () ; kopiraj (s); }
void Skup::pisi () const { // Ispisivanje skupa.
cout<<'{';
for (int i=0; i<vel; i++) { cout << niz[i]; if (i < vel-1) cout << ',';}
cout <<'}';


}
```

```
void Skup::citaj () { // citanje skupa
brisi ( ) ;
int vel;cin>>vel;
double broj;
for (int i=0; i<vel; i++) { cin >> broj; unija (*this, broj);}
}
// Program za ispisivanje klase Skup.
//#include "skup.h"
#include <iostream.h>
int main () {
char jos;
do {
Skup s1; cout << "niz1? "; s1.citaj ();
Skup s2; cout << "niz2? "; s2.citaj ();
cout << "s1 ="; s1.pisi (); cout<<endl;
cout << "s2 ="; s2.pisi (); cout <<endl;
Skup s; s.presek(s1,s2); cout << "sl*s2=";s.pisi();cout<<endl;
s.unija(s1,s2); cout << "sl+s2="; s.pisi();cout<<endl;
s.razlika(s1,s2); cout << "sl-s2="; s.pisi(); cout <<endl;
cout << "\nJos? "; cin >> jos; }
while (jos=='d' || jos=='D');
return 0;
}
```

```
niz1? 4 1 2 3 4
niz2? 5 3 4 5 6 7
s1 ={1,2,3,4}
s2 ={3,4,5,6, 7}
s1*s2={3,4}
s1+s2={1,2,3,4,5,6,7}
s1-s2={1,2}

Jos? d
niz1? 6 9 3 5 1 7 3
niz2? 4 6 2 8 8
s1 ={1,3,5,7,9}
s2 ={2, 6, 8}
s1*s2={}
s1+s2={1,2,3,5,6,7,8,9}
s1-s2={1,3,5,7,9}
Jos? n
```

ZBIRKA URAĐENIH ZADATAKA

KONTROLA ULAZA / IZLAZA I KONVERZIJA TIPOVA

Korišćenje direktive - Namespaces

Biblioteka **standard library** je proširenи set rutina koje su napisane da bi mogli izvršavati različite zadatke: naprimer, rad sa input i output sistemima, izvršavanje osnovnih matematičkih operacija, itd. Umesto da sami pišemo te rutine, jednostavno ih možemo pobrati iz standardnih biblioteka I koristiti ih u našem kodu.. Fajl iostream je samo jedan deo header files koji sadrži rutine standardne biblioteke. (Možemo videti ceo set fajlova standardne biblioteke u MSDN online help fajlovima u **Visual C++ Documentation \ Reference \ C/C++ Language and C++ Libraries \ Standard C++ Library Reference**.)

Kodovi svih rutina standardnih biblioteka sadržani su u naredbi **namespace** std. Tasko, svaka standardna rutina pripada opciji namespace std. Svaki header file standardne biblioteke prilaže nekoliko rutina pozivom namespace std.

Kod našeg programa ne pripada funkciji namespace std. Prema tome, prilikom korišćenja izlaznih rutina, (naprimer iostream header fajlova), moramo reći računaru da ove rutine pripadaju direktivi namespace std. Koristićemo:

using namespace std;

Ovom linijom u našem programu,kompajler zna da ćemo koristiti rutine koje pripadaju standardnoj biblioteci.

Međutim, u jeziku C++ možemo normalno raditi i sa već poznatim direktivama za korišćenje ulaza/izlaza

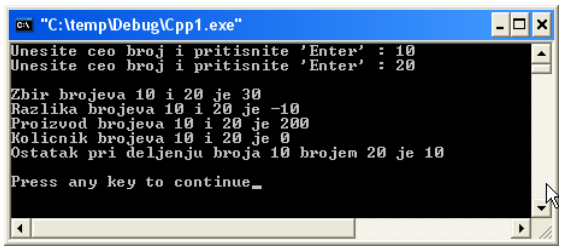
#include<iostream.h>

1 Zadatak:

Napisati program koji zahteva unos dva cela broja, a zatim ih: sabira, oduzima, množi, deli i prikazuje ostatak pri deljenju, a zatim ispisuje rezultate na ekranu.
U ovom programu se demonstrira način unošenja podataka sa konzole.

```
// Program ilustruje upotrebu celobrojnih promenljivih u programskom
// jeziku C++.
// Korisnik unosi dva cela broja.
// Program sabira, oduzima, mnozi i deli unete cele brojeve
// i prikazuje rezultate izracunavanja.
#include <iostream>
using namespace std;
int main()
{
// Deklarisanje promenljivih
int i1,
i2,
zbir,
razlika,
proizvod,
kolicnik,
```

```
ostatak;
// Ucitavanje podataka sa konzole
cout << "Unesite ceo broj i pritisnite 'Enter' : ";
cin >> i1;
cout << "Unesite ceo broj i pritisnite 'Enter' : ";
cin >> i2;
// Izracunavanja
zbir = i1 + i2;
razlika = i1 - i2;
proizvod = i1 * i2;
kolicnik = i1 / i2;
ostatak = i1 % i2;
// Ispisivanje rezultata
cout << endl;
cout << "Zbir brojeva " << i1 << " i " << i2 << " je " << zbir <<endl;
cout << "Razlika brojeva " << i1 << " i " << i2 << " je "
<< razlika << endl;
cout << "Proizvod brojeva " << i1 << " i " << i2 << " je "
<< proizvod << endl;
cout << "Kolicnik brojeva " << i1 << " i " << i2 << " je "
<< kolicnik << endl;
cout << "Ostatak pri deljenju broja " << i1 << " brojem " << i2
<< " je " << ostatak << endl << endl;
return 0;
}
```



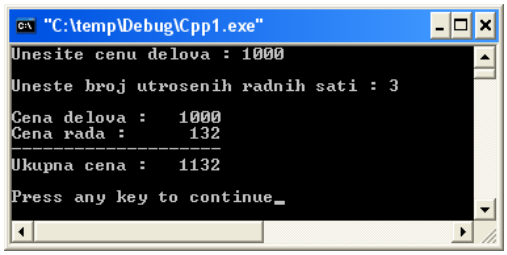
2 Zadatak:

Napisati program koji izračunava cenu servisiranja uređaja ako se cena delova i broj radnih sati unose sa tastature, dok se cena radnog sata defie kao konstanta. Sve vrednosti su celi brojevi. Koristiti formatirano ispisivanje iznosa, poravnati ih po desnoj strani.

U programu je pokazan način definisanja konstantne veličine – veličine koja ne menja svoju vednost za vreme izvršavanja programa.

```
// Program izracunava ukupnu cenu delova i rada
#include <iostream.h>
#include <iomanip.h>
int main()
{
// Deklarisanje promenljivih
```

```
const int CENA_PO_SATU = 44;
int delovi, // Cena delova
sati, // Broj utrosenih radnih sati
rad, // Cena rada
ukupno; // Ukupo za naplatu
// Unos podataka
cout << "Unesite cenu delova : ";
cin >> delovi;
cout << endl;
cout << "Uneste broj utrosenih radnih sati : ";
cin >> sati;
cout << endl;
// Izracunavanja
rad = CENA_PO_SATU * sati;
ukupno = delovi + rad;
// Ispisivanje rezultata
cout << "Cena delova : " << setw(6) << delovi << endl;
cout << "Cena rada : " << setw(8) << rad << endl;
cout << "--------------------" << endl;
cout << "Ukupna cena : " << setw(6) << ukupno << endl << endl;
return 0;
}
```

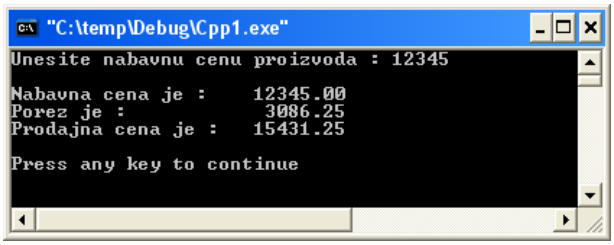


3 Zadatak:

Napisati program koji izračunava porez na nabavnu cenu i maloprodajnu cenu proizvoda. PDV stopa poreza je konstantna, dok se nabavna cena unosi sa tastature. Rezultate prikazati u formatiranom ispisu.
Za promenljive koristiti realne brojeve tipa double.

```
// Program izracunava porez i maloprodajnu cenu proizvoda
#include <iostream.h>
#include <iomanip.h>
int main()
{
// Deklarisanje promenljivih
const double STOPA_POREZA = 0.25;
double nabavna_cena, // Nabavna cena proizvoda
porez, // Porez na posmatrani proizvod
cena; // Prodajna cena (nabavna_cena +
// porez)
```

```
//Podesavanje izlaznog niza za prikazivanje iznosa
cout << setprecision(2)     // defie broj decimalnih mesta
<< setiosflags(ios::fixed)  //govori da će ispis biti u fiksnom obliku
<< setiosflags(ios::showpoint);  /*‘traži’ od kompajlera da
                                    upotrebi decimalnu tačku da bi razdvojio celobrojni deo
od decimalnog*/
// Unos podataka
cout << "Unesite nabavnu cenu proizvoda : ";
cin >> nabavna_cena;
// Izracunavanja
porez = nabavna_cena * STOPA_POREZA;
cena = nabavna_cena + porez;
// Ispisivanje rezultata
cout << endl;
cout << "Nabavna cena je : " << setw(11) << nabavna_cena << endl;
cout << "Porez je : " << setw(18) << porez << endl;
cout << "Prodajna cena je : " << setw(10) << cena << endl << endl;
return 0;
}
```

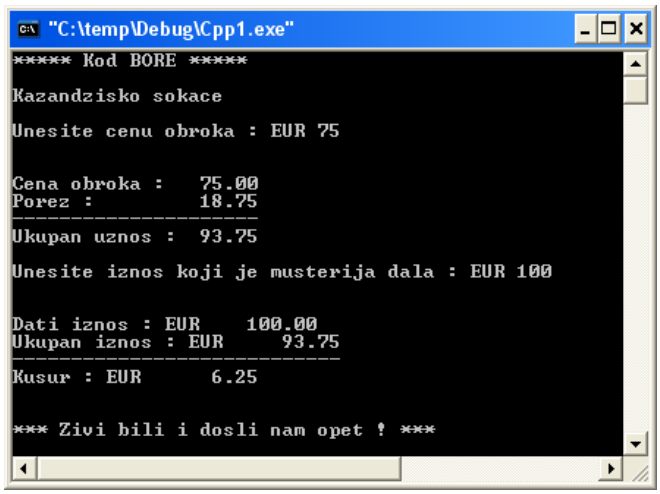


4 Zadatak:

Napisati program za kasu u restoranu koji izračunava maloprodajnu cenu obroka, kao i kusur koji je potrebno vratiti gostu restorana na osnovu cene obroka i datog iznosa. Izvršiti ispisivanje pozdravne i završne poruke.
Za promenljive koristiti realne brojeve tipa double.

```
// Program za kasu u restoranu
#include <iostream.h>
#include <iomanip.h>
int main()
{
// Deklarisanje promenljivih
const double STOPA_MALOPRODAJNOG_POREZA = 0.25;
double cena_obroka, // Nabavna cena obroka za restoran
iznos_poreza, // Iznos poreza
ukupno, // Ukupno za naplatu
dati_iznos, // Iznos koji je dala musterija
kusur; // Kusur: dati_iznos - ukupno
//Podesavanje izlaznog formata za ispisivanja iznosa
cout << setprecision(2)
<< setiosflags(ios::fixed)
```

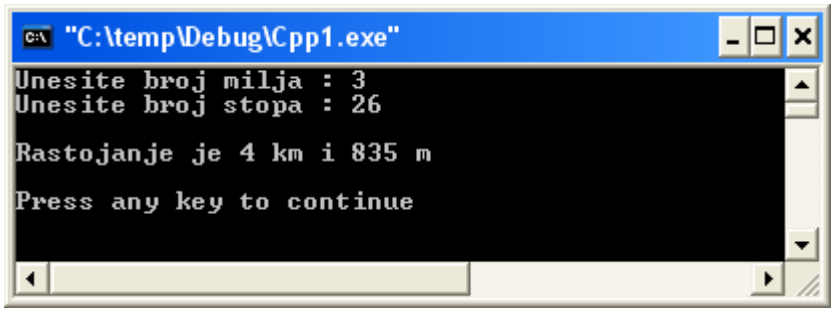
```
<< setiosflags(ios::showpoint);
// Ispisivanje naziva restorana i unos cene obroka
cout << "***** Kod BORE *****" << endl << endl;
cout << "Kazandzisko sokace" <<endl << endl;
cout << "Unesite cenu obroka : EUR ";
cin >> cena_obroka;
cout << endl;
// Izracunavanje poreza i ukupne cene
iznos_poreza = cena_obroka * STOPA_MALOPRODAJNOG_POREZA;
ukupno = cena_obroka + iznos_poreza;
// Ispisivanje poreza i ukupne cene
cout << endl;
cout << "Cena obroka : " << setw(7) << cena_obroka << endl;
cout << "Porez : " << setw(13) << iznos_poreza << endl;
cout << "---------------------" << endl;
cout << "Ukupan uznos : " << setw(6) << ukupno << endl;
// Unos datog iznosa
cout << endl;
cout << "Unesite iznos koji je musterija dala : EUR ";
cin >> dati_iznos;
cout << endl;
// Izracunavanje kusura
kusur = dati_iznos - ukupno;
// Ispisivanje kusura
cout << endl;
cout << "Dati iznos : EUR " << setw(9) << dati_iznos
<< endl;
cout << "Ukupan iznos : EUR " << setw(9) << ukupno << endl;
cout << "----------------------------" << endl;
cout << "Kusur : EUR " << setw(9) << kusur << endl;
// Ispisivanje zavrsne poruke
cout << endl << endl;
cout << "*** Zivi bili i dosli nam opet ! ***" << endl << endl;
return 0;
}
```



5 Zadatak:

Napisati program koji konvertuje dužine unete u miljama i stopama u dužine izražene u kilometrima i metrima.
U programu se demonstrira kombinovano korišćenje promenljivih različitog tipa.

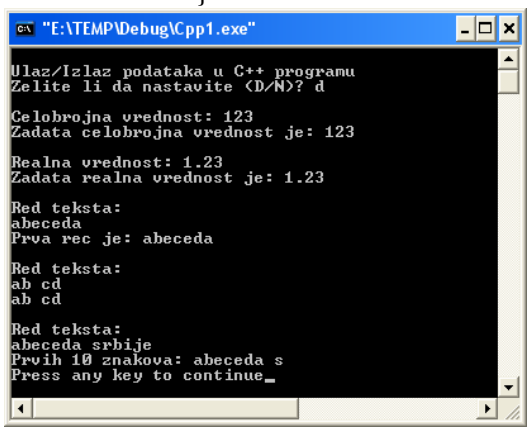
```
// Preracunavanje duzine iz anglosaksonskog u metricki sistem
#include <iostream.h>
int main()
{
// Deklarisanje promenljivih
const double METARA_PO_MILJI = 1609.35;
const double METARA_PO_STOPI = 0.30480;
int milje,
stope,
kilometri,
metri;
double ukupno_metri,
ukupno_kilometri;
// Unos podataka
cout << "Unesite broj milja : ";
cin >> milje;
cout << "Unesite broj stopa : ";
cin >> stope;
// pretvaranje unete duzine u metre
ukupno_metri = milje * METARA_PO_MILJI + stope * METARA_PO_STOPI;
// Izracunavanje broja kilometara
ukupno_kilometri = ukupno_metri / 1000;
kilometri = ukupno_kilometri; // odbacivanje decimalnog dela i
// dobijanje celog broja
// kilometara
// Preracunavanje decimalnog dela kilometara u metre
metri = (ukupno_kilometri - kilometri) * 1000;
// Ispisivanje rezultata
cout << endl;
cout << "Rastojanje je " << kilometri << " km i "
<< metri << " m" << endl << endl;
return 0;
}
```



6 Zadatak:

//Realizacija ulaza/izlaza (tastatura/ekran) za razlicite vrsta podataka

```
#include <iostream.h>
#include <ctype.h>
#define MAX1 80
#define MAX2 10
main( )
{       int znak, broj_i;    float  broj_f;    char   odgovor, string[MAX1+1];

        cout << "\nUlaz/Izlaz podataka u C++ programu" << endl;
        cout << "Zelite li da nastavite (D/N)? ";
        cin  >> odgovor;
        cin.ignore();                          //prihvata znak za prelaz u novi red
                if( toupper(odgovor) == 'D' )
        {       cout << "\nCelobrojna vrednost: ";
                cin  >>  broj_i;
                cout << "Zadata celobrojna vrednost je: " << broj_i << endl;
                cout << "\nRealna vrednost: ";
                cin  >>  broj_f;
                cout << "Zadata realna vrednost je: " << broj_f << endl;
                //Citanje do prvog belog znaka od zadatog reda znakova
                cout << "\nRed teksta:" << endl;
                cin  >> string;
                cout << "Prva rec je: " << string << endl;
                //Citanje ostatka od zadatog reda znakova
                cin.get(string, MAX1);      //prihvata ostatak do kraja reda
                cin.ignore();               //prihvata znak za prelaz u novi red
                //Citanje znak po znak do znaka za prelaz u novi red
                cout << "\nRed teksta:" << endl;
                while((znak = cin.get()) != '\n')
                cout.put((char)znak);       //moze i: cout << (char)znak;
                //Citanje zadatog broja znakova ili do znaka za prelaz u novi red
                cout << "\n\nRed teksta:" << endl;
                cin.get(string, MAX2);
          cout << "Prvih " << MAX2<< " znakova: " << string << endl;          }
        return 0;
        }
```

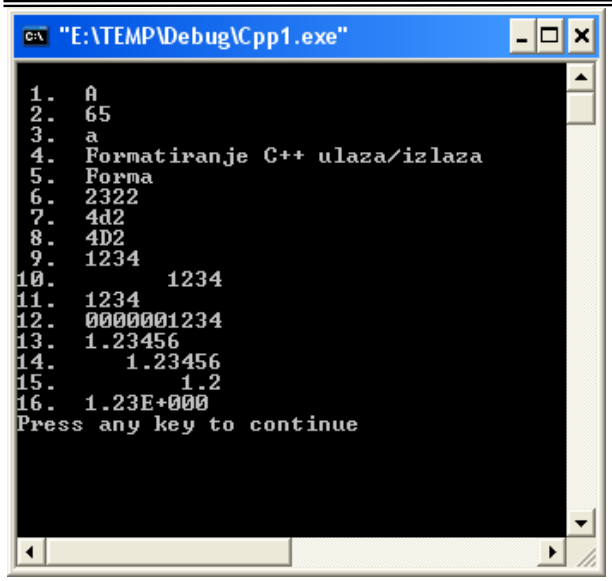


7 Zadatak:

//Primena funkcija za konverziju i formatiranje ulaza/izlaza

```
#include <iostream.h>
#include <iomanip.h>
void stampa_rbr( void );

main( )
{       char slovo = 'A';     char string[] = "Formatiranje C++ ulaza/izlaza";
        int broj_i = 1234;    double broj_d = 1.23456;

        //1. stampanje samog slova A
        stampa_rbr();
        cout << slovo;

        //2. stampanje ASCII koda slova A
        stampa_rbr();
        cout << (int)slovo;

        //3. stampanje znaka zadate ASCII vrednosti (slovo a)
        stampa_rbr();
        cout << (char)97;

        //4. stampanje niza znakova
        stampa_rbr();
        cout << string;

        //5. stampanje prvih 5 znakova od niza
        stampa_rbr();
        cout.write(string,5);

        //6. stampanje celog broja u oktalnom obliku
        stampa_rbr();
        cout.setf(ios::oct);
        cout << broj_i;                    //moze i: cout << oct << broj_i;
        cout.unsetf(ios::oct);

        //7. stampanje celog broja u heksa obliku (malim slovima)
        stampa_rbr();
        cout.setf(ios::hex);
        cout << broj_i;                    //moze i: cout << hex << broj_i;

        //8. stampanje celog broja u heksa obliku (velikim slovima)
        stampa_rbr();
        cout.setf(ios::uppercase);
        cout << broj_i;
        cout.unsetf(ios::hex);
```

```
        //9. stampanje celog broja u decimalnom obliku
        stampa_rbr();
        cout << broj_i;                         //moze i: cout << dec << broj_i;

        //10. stampanje celog broja,u polju zadate sirine, desno ravnanje
        stampa_rbr();
        cout.width(10);
        cout << broj_i;

        //11. stampanje celog broja, u polju zadate sirine, levo ravnanje
        stampa_rbr();
        cout.width(10);
        cout.setf(ios::left);
        cout << broj_i;
        cout.unsetf(ios::left);

        //12. stampanje celog broja, desno popravnanje, nule ispred broja
        stampa_rbr();
        cout.width(10);
        cout.fill('0');
        cout << broj_i;
        cout.fill(' ');

        //13. stampanje realnog broja, sa standardnom preciznoscu
        stampa_rbr();
        cout << broj_d;

        //14. stampanje realnog broja, u polju zadate sirine,
        stampa_rbr();
        cout.width(10);
        cout << broj_d;

        //15. stampanje realnog broja, sa 2. decimalne cifre
        stampa_rbr();
        cout.width(10);
        cout.precision(2);
        cout << broj_d;

        //16. stampanje realnog broja, sa eksponentom
        stampa_rbr();
        cout.setf(ios::scientific);
        cout << broj_d << endl;
        cout.unsetf(ios::scientific);

        return 0;
}
//Funkcija koja prikazuje redni broj na pocetku reda
void stampa_rbr (void)
{       static int redni_broj = 0;
        cout << "\n";
        cout.width(2);
        cout << ++redni_broj << ".  ";
}
```



8 Zadatak:

```
//Tabeliranje vrednosti izraza 10 stepenovan brojem x u zadatom opsegu
        // i sa zadatim korakom za x (levo i desno poravnanje vrednosti izraza)

        #include <iostream.h>
        #include <iomanip.h>
        #include <math.h>
        #define MIN             0
        #define MAX             20
        #define MAX1    2
        #define MAX2    10

        main()
        {       int xmin, xmax, dx;
                do
                {       cout << "\nUnesite cele brojeve ";
                cout << "xmin>=" << MIN << ", xmax<=" << MAX << " i dx: ";
                        cin >> xmin >> xmax >> dx;
                }while(xmin<MIN || xmax>MAX || dx>xmax);

                //Stampanje na ekranu zaglavlja tabele i tabele
                cout << "\nx\t10 stepenovano brojem x\t\t10 stepenovano brojem x"
                        << "\n\t(desno poravnanje)\t\t(levo poravnanje)"
                        << "\n================================"
                        << "================================\n\n";

                cout.setf(ios::fixed);
                cout.precision(0);

                for(int x = xmin; x<=xmax; x+=dx)
                {       //Desno poravnanje vrednosti x  u 1. koloni tabele
```

```
                cout.width(MAX1);
                cout.unsetf(ios::left);
                cout << x << "\t\t";

                //Desno poravnjanje vrednosti izraza u 2. koloni tabele
                cout.width(MAX2);
                cout.unsetf(ios::left);
                cout << pow(10,x) << "\t\t";

                //Levo poravnjanje vrednosti izraza u 3. koloni tabele
                cout.width(MAX2);
                cout.setf(ios::left);
                cout << pow(10,x) << "\n";
        }
        cout << "\n";
        return 0;
}
```

9 Zadatak:

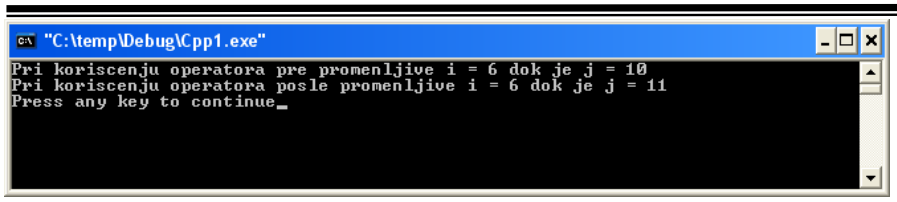
```
//Citanje niza celih brojeva iz komandne linije i prikaz na ekranu
//njegovih elemenata u oktalnom, heksadecimalnom i binarnom obliku

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>

void stampa_bin( int decimal_value );

main(int argc, char *argv[])
{       int decimalni;

        if( argc < 2 )
        {       cerr << "\nUneti ime programa i niz celih brojeva." << endl;
                exit(0);
        }

        cout << "\nZadati niz brojeva u raznim oblicima:\n\n";
```

```
        cout << "dec\tokt\theksa\tbin" << endl;

        for(int i=1; i<argc; i++)
        {       decimalni = atoi( argv[i]);
                cout << "\n" << decimalni << "\t";
                cout << oct << decimalni << "\t";
                cout << hex << decimalni << "\t";
                stampa_bin(decimalni);
        }

        return 0;
}
//Funkcija koja prikazuje na ekranu niz u binarnom obliku
void stampa_bin(int broj)
{       int broj_cifara = 0;
        int niz_binarnih[50];

        while( broj != 0 )
        {       niz_binarnih[broj_cifara++] = broj % 2;
                broj /= 2;
        }

        for( int i=broj_cifara-1; i>=0; i-- )
                cout << dec << niz_binarnih[i];
}
```

10 Zadatak:

Naredni program ilustruje upotrebu unarnih operatora ispred i iza promenljive. Upotreba operatora kao u sledećem primeru je veoma česta u C++ jeziku. Zato je neophodno dobro razumeti sledeći primer.

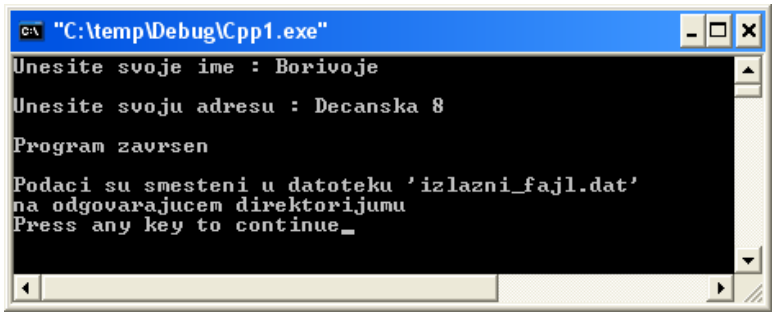
```
// Program ilustruje razliku izmedju primene opearotra pre
// i posle promenljive
#include <iostream.h>
int main()
{// Deklarisanje promenljivih
int i,j;
// Izracunavanja i ispisivanje rezultata
i = 7; // Dodeljivanje inicijalne vrednosti
j = 4 + --i; // prvo se od i oduzme 1 a zatim se
             // sabere sa 4
cout << "Pri koriscenju operatora pre promenljive i = "
<< i << " dok je j = " << j << endl;
i = 7; // Ponovna inicijalizacija promenljive
// i na 7
j = 4 + i--; // prvo se i sabere sa 4 a zatim mu se
// oduzme 1
cout << "Pri koriscenju operatora posle promenljive i = "
<< i << " dok je j = " << j << endl;
return 0;}
```



11 Zadatak:

Napisati program koji upisuje podatke u fajl.

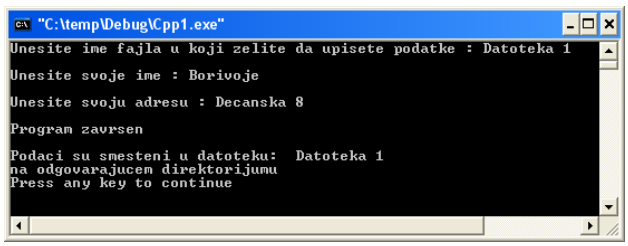
```
// Program prikazuje upis podataka u fajl
#include <fstream>
#include <iostream>
using namespace std;

int main()
{
char buffer[81];
ofstream izlazni_fajl("izlazni_fajl.dat");
cout << "Unesite svoje ime : ";
cin.getline(buffer, 81);
izlazni_fajl << buffer << endl;
cout << endl;
cout << "Unesite svoju adresu : ";
cin.getline(buffer, 81);
izlazni_fajl << buffer << endl;
izlazni_fajl.close();
cout << endl;
cout << "Program zavrsen" << endl << endl;
cout << "Podaci su smesteni u datoteku 'izlazni_fajl.dat'" << endl;
cout << "na odgovarajucem direktorijumu" << endl;
return 0;
}
```



12 Zadatak:

Napisati program koji upisuje podatke u fajl i proverava da li postoji greška prilikom otvaranja fajla.

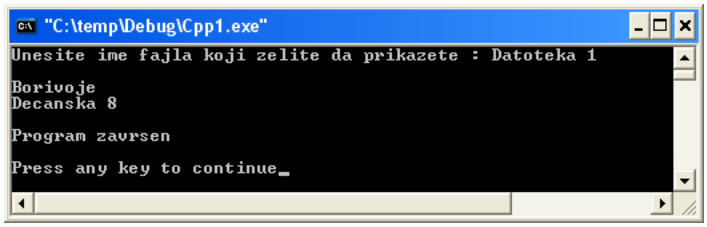
```
// Program demonstrira upis podataka u fajl
// Proverava da li postoji greska prilikom otvaranju fajla
#include <fstream>
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
char buffer[81];
char ime_fajla[81];
cout << "Unesite ime fajla u koji zelite da upisete podatke : ";
cin.getline(ime_fajla, 81);
ofstream izlazni_fajl(ime_fajla);
if (!izlazni_fajl)
{
cerr << endl;
cerr << "GRESKA: Fajl ne moze biti otvoren." << endl;
cerr << "Program zavrsen" << endl << endl;
exit(1);
}
cout << endl;
cout << "Unesite svoje ime : ";
cin.getline(buffer, 81);
izlazni_fajl << buffer << endl;
cout << endl;
cout << "Unesite svoju adresu : ";
cin.getline(buffer, 81);
izlazni_fajl << buffer << endl;
izlazni_fajl.close();
cout << endl;
cout << "Program zavrsen" << endl << endl;
cout << "Podaci su smesteni u datoteku:  "<< ime_fajla << endl;
cout << "na odgovarajucem direktorijumu" << endl;
return 0;
}
```



13 Zadatak:

Napisati program koji čita podatke iz fajla.

```
// Program prikazuje ucitavanje podataka iz fajla
#include <fstream>
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
char buffer[81];
char ime_fajla[81];
cout << "Unesite ime fajla koji zelite da prikazete : ";
cin.getline(ime_fajla, 81);
ifstream ulazni_fajl(ime_fajla);
if (!ulazni_fajl)
{
cerr << endl;
cerr << "GRESKA: Fajl ne moze biti otvoren." << endl;
cerr << "Program zavrsen" << endl << endl;
exit(1);
}
cout << endl;
while (!ulazni_fajl.eof())
{
ulazni_fajl.getline(buffer, 81);
cout << buffer << endl;
}
ulazni_fajl.close();
cout << "Program zavrsen" << endl << endl;
return 0;
}
```

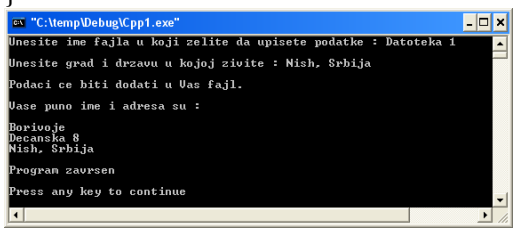


14 Zadatak:

Napisati program koji upisuje podatke u fajl, čita podatke iz fajla i dodaje podatke u fajl.

```
// Program ilustruje upis podataka u fajl kao i citanje iz fajla.
// Fajl je otvoren u modu za upis, citanje i dodavanje podataka.
#include <fstream>
#include <iostream>
```

```
#include <cstdlib>
using namespace std;
int main()
{char buffer[81];
char ime_fajla[81];
cout << "Unesite ime fajla u koji zelite da upisete podatke : ";
cin.getline(ime_fajla, 81);
cout << endl;
cout << "Unesite grad i drzavu u kojoj zivite : ";
cin.getline(buffer, 81);
cout << endl;
cout << "Podaci ce biti dodati u Vas fajl." << endl;
ofstream mod_za_dodavanje(ime_fajla, ios::app);
if (!mod_za_dodavanje)
{cerr << endl;
cerr << "GRESKA: Fajl ne moze biti otvoren "
<< "u modu za dodavanje podataka." << endl;
cerr << "Program zavrsen" << endl << endl;
exit(1);}
mod_za_dodavanje << buffer << endl;
mod_za_dodavanje.close();
cout << endl;
cout << "Vase puno ime i adresa su :" << endl << endl;
ifstream mod_za_upisivanje_i_citanje(ime_fajla);
if (!mod_za_upisivanje_i_citanje)
{cerr << endl;
cerr << "GRESKA: Fajl ne moze biti otvoren." << endl;
cerr << "Program zavrsen" << endl << endl;
exit(1);}
while (!mod_za_upisivanje_i_citanje.eof())
{mod_za_upisivanje_i_citanje.getline(buffer, 81);
cout << buffer << endl;}
mod_za_upisivanje_i_citanje.close();
cout << "Program zavrsen" << endl << endl;
return 0;
}
```

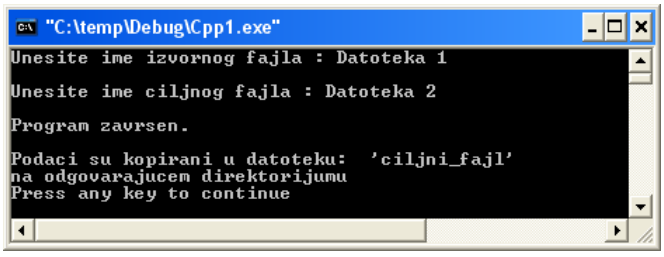


15 Zadatak:

Napisati program koji kopira jedan fajl u drugi koristeći get() and put() naredbe.

```
// Program kopira jedan fajl u drugi koristeci get() and put()
#include <fstream>
#include <iostream>
```

```
#include <cstdlib>
using namespace std;
int main()
{char ime_fajla[81];
char znak;
cout << "Unesite ime izvornog fajla : ";
cin.getline(ime_fajla, 81);
ifstream izvorni_fajl(ime_fajla);
if (!izvorni_fajl)
{cout << endl;
cout << "GRESKA: Fajl ne moze biti otvoren." << endl;
exit(1);}
cout << endl;
cout << "Unesite ime ciljnog fajla : ";
cin.getline(ime_fajla, 81);
ofstream ciljni_fajl(ime_fajla);
if (!ciljni_fajl)
{cout << endl;
cout << "GRESKA: Fajl ne moze biti otvoren." << endl;
exit(1);}
while ( (znak = izvorni_fajl.get()) != EOF )
ciljni_fajl.put(znak);
izvorni_fajl.close();
ciljni_fajl.close();
cout << endl;
cout << "Program zavrsen." << endl << endl;
cout << "Podaci su kopirani u datoteku:  'ciljni_fajl'"<<endl;
cout << "na odgovarajucem direktorijumu" << endl;

return 0;}
```



KONVERZIJA TIPOVA

Na primer, pogledajmo sledeći C++ fragment, koji izračunava i ispisuje zbir recipročnih vrednosti svih brojeva od 1 do 1000:

```
double suma = 0;
for(int i = 1; i <= 1000; i++)
suma =suma+ 1. / i;
cout<< suma;
```

U ovom primeru, promjenljiva "i", deklarisana unutar **"for"** petlje, postoji samo dok se petlja ne završi. U C++-u je preporuka da se brojačke promjenljive koje upravljaju radom **"for"** petlji deklarišu isključivo na ovakav način. Na taj način se sprečava mogućnost da se brojačka promjenljiva nehotice neispravno upotrebi ili čak zloupotrebi izvan tela petlje.

Obratimo pažnju da smo u prethodnom primeru pisali "1." umjesto "1". Da to nismo uradili, imali bismo pogrešan rezultat. Naime, kako je "1" celobrojnog tipa, i kako je promjenljiva "i" takođe celobrojna, operator "/" u izrazu kao "1 / i" interpretirao bi se kao *celobrojno deljenje.* Ovako, kako je "1." realnog tipa, operator "/" se interpretira kao *klasično deljenje.* Ovo je dobar povod da kažemo nešto o ***konverziji tipova*** u C++-u. C++ podržava skoro sve automatske konverzije tipova koji se dešavaju i u jeziku C (uz retke izuzetke koji će biti posebno istaknuti). Na primer, celobrojni izrazi se automatski konvertuju u realne ukoliko se upotrebe u kontekstu u kojem se očekuje realan izraz, npr. ukoliko želimo dodeliti neki celobrojni izraz realnoj promenljivoj. Ovakvu automatsku konverziju, pri kojoj se "uži" tip konvertuje u "širi" obično zovemo *promocija.* S druge strane, realan izraz često možemo upotrebiti u kontekstu u kojem se očekuje celobrojni izraz, npr. kao argument neke funkcije koja očekuje celobrojni argument, ili pri dodeli realnog izraza celobrojnoj promenljivoj. Takve konverzije praćene su *gubitkom informacija* (u konkretaom primeru *odsecanjem decimald),* i obično ih nazivamo *degradacija.* C++ kompajleri imaju pravo da nailaskom na degradirajuću automatsku konverziju emituju *poruku upozorenja,* ali ne smeju odbiti da prevedu program.

Operator ***(tip)izraz*** se može koristiti i u jeziku C++. Pogledajmo, na primer, sledeći programski isečak koji ispisuje količnik dve celobrojne vrednosti unesene sa tastature:

```
int broj_l, broj_2;
cin>> brojl >> broj_2;
cout<< (double)brojl / broj_2;
```

U ovom isečku, pomoću eksplicitne konverzije privremeno smo promenili tip promenljive "brojl" u realni tip **"double"** da bismo izbegli da operator "/" bude interpretiran kao celobrojno deljenje. Mada ovakva sintaksa i dalje radi u jeziku C++, ona se više *ne preporučuje.* C++ uvodi dve nove sintakse za konverziju tipova:

- Prva je takozvana ***funkcijska* ili *konstruktorska* notacija,** koja izgleda ovako:

tip(izraz)

Dakle, u ovoj sintaksi, ime tipa se tretira kao *funkcija koja svoj argument prevodi u rezultat iste vrednosti, ali odgovarajućeg tipa.* Prethodni primjer bi se, u skladu sa ovom sintaksom, napisao ovako:

```
int brojl, broj_2;
cin >> broj_l >> broj_2;
cout« double(broj 1) / broj_2;
```

Ova sintaksa može biti mnogo praktičnija kod složenijih izraza. Naime, zbog vrlo visokog prioriteta operatora konverzije tipa u C stilu, često su svakako potrebne dodatne

zagrade. Na primer, neka želimo izračunati i ispisati celi dio produkta "2.541 * 3.17". Koristeći C notaciju, to bismo uradili ovako:

cout<< (int)(2.541 * 3.17);

Dodatne zagrade su neophodne, jer bi se u suprotnom operator konverzije "(int)" odnosio samo na podatak "2.541" (odnosno, kao rezultat bismo dobili vrijednost produkta "2 * 3.17"). Korištenjem flinkcijske notacije, istu stvar možemo uraditi ovako:

cout<< int(2.541 * 3.17);

Na ovaj način, ime tipa **"int**' se ujedno može interpretirati i kao *funkcija* koja vraća kao rezultat celi deo svog argumenta.

• Druga sintaksa za konverziju tipa uvedena u jeziku C++, koristi novu ključnu reč **"static_cast",** a izgleda ovako:

static_cast<tip>(izraz)

U skladu sa ovom sintaksom, prethodno napisani primeri izgledali bi ovako:

```
int brojl, broj_2;
cin >> brojl>> broj_2;
cout<< static_cast<double>(broj_l) / broj_2;
```

odnosno

cout<< static_cast<int>(2.541 * 3.17);

Ova sintaksa je dosta rogobatna, ali ona ima svoje prednosti. Prvo, mesta gde se vrše konverzije tipa su potencijalno opasna mjesta, koja često prave probleme kada se program treba preneti sa jednog tipa računara na drugi, ili sa jednog na drugi operativni sistem. Korištenjem ove sintakse, takva mesta u programu je lako locirati prostim traženjem ključne riječi "static_cast". Drugo, konverzija tipa se. za različite tipove u C++-u često vrši na suštinski različite načine, na šta programer ranije nije imao nikakvog uticaja. Sada su u C++-U uvedene četiri ključne reči za konverziju tipova (pored već spomenute **"static_cast",** imamo i "const_cast", **"reinterpret_cast"** i **"dynamic_cast**'), čime programer može eksplicitnije da iskaže tačnu nameru kakvu konverziju želi.

FORMATIRANJE IZLAZA

Možemo li pisati početak programa sa "void main()"?

Definicija void main() { /* ... */ } nije preporučljiva u konotaciji C++.

Uglavnom se koristi konotacija

```
int main() { /* ... */ }
```

ili

```
int main(int argc, char* argv[]) { /* ... */ }
```

Možemo implementirati više verzija main() konotacije, ali sve one moraju imati povratni tip **int. Int** vraćen iz main() je način da program vrati vrednost sistemu i obrati joj se. Kod sistema koji ne poseduju takvu olakšicu vrednost return se ignoriše, što ne čini opciju "void main()" legalnom za C++ ili C. I u slučaju da kompajler prihvati "void main()", to treba izbegavati, jer u suprotnom rizikujete da vas C i C++ programeri smatraju neznalicom.

U C++, main() ne mora da sadrži explicitno return naredbu. U ovom slučaju, vraćena vrednost 0, podrazumeva uspešno izvršenje. Naprimer:

```
#include<iostream>
using namespace std;

int main()
{
std::cout << "This program returns the integer value 0\n";
```

Imajte u vidu da ni ISO C++ ni C99 neomogućuju da izostavimo tip deklaracije. Tojest u poređenju sa C89 i ARM C++ ,"int" nije dozvoljeno da se ne pojavi u deklaraciji. Kao posledica toga:

```
#include<iostream.h>

main() { /* ... */ }
```

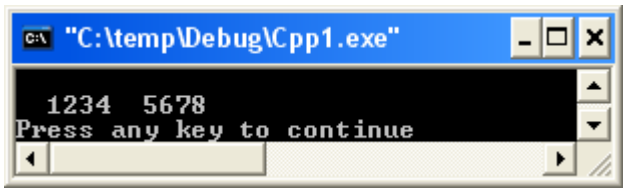
javiće se greška, jer return tip funkcije main() nedostaje.

Return naredba završava izvršenje funkcije i vraća kontrolu na funkciju koja ju je pozvala. Ako izostavimo return vrednost deklarisaćemo funkciju da ima **void** povratni tip i da takva funkcija ne vraća nikakvu vrednost, dok je u ostalim slučajevima povratni tip funkcije po difoltu **int.**

1 Zadatak:

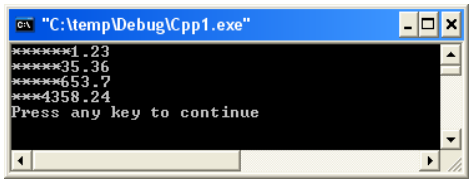
```
// manipulator setw()
#include <iostream.h>
#include <iomanip.h>
int main()
{
  int num1 =1234, num2 =5678;
  cout << endl;                    //Start sa novom linijom
```

```
 cout << setw(6) << num1 << setw(6) << num2;   /*Izlaz dve vrednosti sa     manipulatorom
širine setw*/
   cout << endl;                          //Start novom linijom
   return 0;                              //Exit programa
}
```



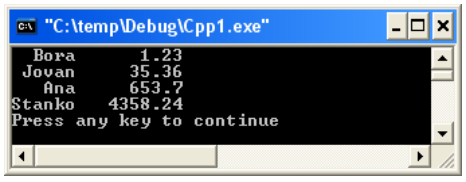
2 Zadatak:

```
//primena operatora width() i  metode fill
#include <iostream.h>
void main()
{
   double values[] = { 1.23, 35.36, 653.7, 4358.24 };
   for( int i = 0; i < 4; i++ )
{ cout.width(10); //formatiranje širine podatka funkcijom članicom width ( slično kao setw )
cout.fill( '*' );  //popuna praznih mesta f-je članice width u izlaznom formatu, naprimer
zvezdicama
cout << values[i] << '\n';  }
}
```



3 Zadatak:

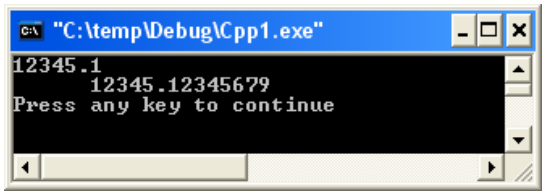
```
#include <iostream.h>
#include <iomanip.h>
void main()
{
   double values[] = { 1.23, 35.36, 653.7, 4358.24 };
   char *names[] = { "Bora", "Jovan", "Ana", "Stanko" };
   for( int i = 0; i < 4; i++ )
      cout << setw( 6 )  << names[i]
           << setw( 10 ) << values[i] << endl;
}
```



4 Zadatak:

```
//primena width i precision formata za IOS
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
int main()
{const double prom=12345.123456789;
cout << prom<<endl;
cout.width(20);     /* "20" nije broj razmaka koji se ispisuju ispred podatka, nego broj mesta
                       koje ce          zauzeti          podatak*/
cout<<setiosflags(ios::fixed);
cout.precision(8);
cout << prom<<endl;
return 0;}
```

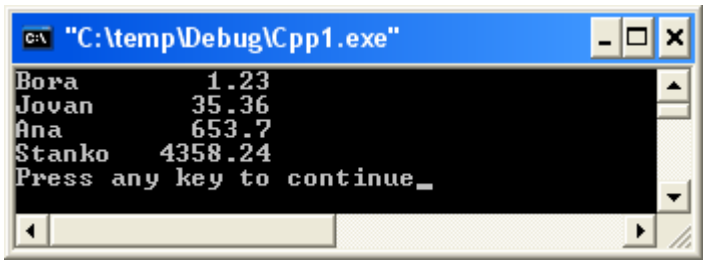
IZLAZ:



5 Zadatak:

```
//ravnanje izlaza levo ili desno
#include <iostream.h>
#include <iomanip.h>
int main()
{
  double values[] = { 1.23, 35.36, 653.7, 4358.24 };
  char *names[] = { "Bora", "Jovan", "Ana", "Stanko" };

for ( int i = 0; i < 4; i++ )
  cout << setiosflags( ios::left )
       << setw( 6 )  << names[i]
       << resetiosflags( ios::left )
       << setw( 10 ) << values[i] << endl;
return 0;}
```

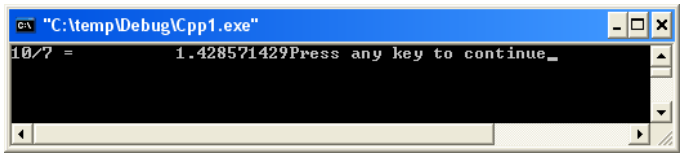
IZLAZ:



6 Zadatak:

```
#include <iostream.h>
#include <iomanip.h>
int main()
{cout << "10/7 = ";
cout.width(20);     /* “20” nije broj razmaka, nego broj mesta koje će zauzeti podatak*/
cout.precision(10);
cout << 10./7.;}
```

IZLAZ:



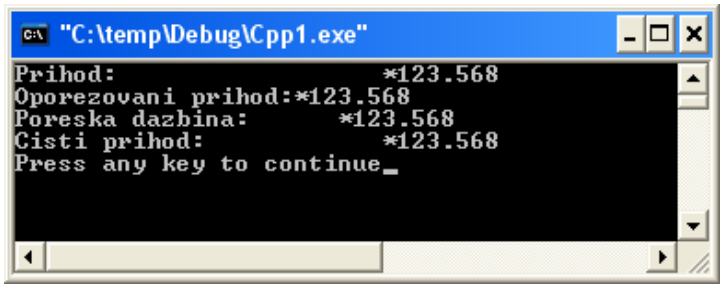
Međutim, mogli bi prosto napisati za predhodni zadatak:

```
cout << "1/7 = " << setw(20) << setprecision(10) << 10./7;
```

7 Zadatak:

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

int main()
{
double const prihod=123.56789;
double const oporezovani_prihod=123.56789;
double const porez=123.56789;
double const prihod_porez=123.56789;
cout.fill( '*' );  // popuna praznih mesta u formatu zvezdicama
cout << "Prihod:          " << setw(8) << prihod << endl;  //    prikaz osam mesta sa tackom
cout << "Oporezovani prihod:" << setw(8) << oporezovani_prihod << endl;
cout << "Poreska dazbina:      " << setw(8) << porez << endl;
cout << "Cisti prihod:              " << setw(8) << prihod_porez << endl;
return 0;}
```

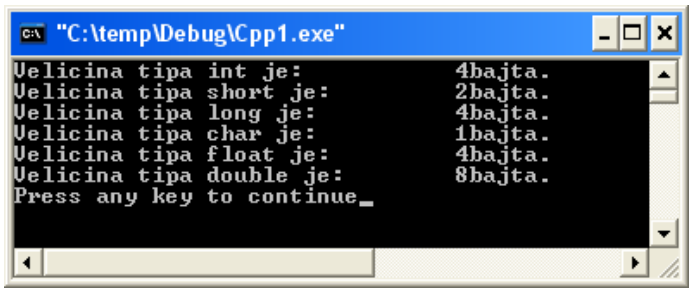
IZLAZ:



8 Zadatak:

```
//funkcije sizeof()
#include <iostream.h>
int main()
{cout<< "Velicina tipa int je: \t\t" << sizeof(int)     <<"bajta.\n";
cout << "Velicina tipa short je:\t\t" << sizeof(short) <<"bajta.\n";
cout << "Velicina tipa long je: \t\t" << sizeof(long)   <<"bajta. \n";
cout << "Velicina tipa char je: \t\t" << sizeof(char)   <<"bajta. \n";
cout << "Velicina tipa float je:\t\t" << sizeof(float) <<"bajta. \n";
cout << "Velicina tipa double je:\t" << sizeof(double) <<"bajta. \n";
return 0;}
```

IZLAZ:

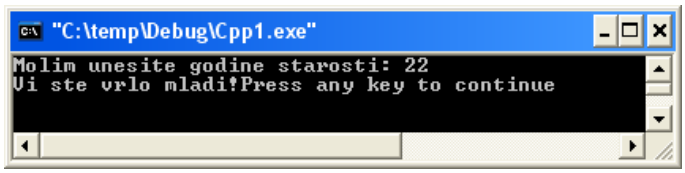


KONTROLA TOKA PROGRAMA

1 Zadatak:

Primer za IF naredbu uz upotrebu else strukture:

```
#include <iostream.h>
int main()                  //Početak programa!
{
 int age;                                          //Defiemo promenljivu...
 cout<<"Molim unesite godine starosti: ";     //Upit na monitoru
 cin>>age;                                         //Unosi se broj godina
 if(age<100)                          //Ako su manje od 100
 {    cout<<"Vi ste vrlo mladi!";    }
 else if(age==100)      //Koristi se  else za novi odnos
 {     cout<<"Vi ste stari";  }
 else
 {    cout<<" Vi ste vrlo stari ";      }
 return 0;}
```

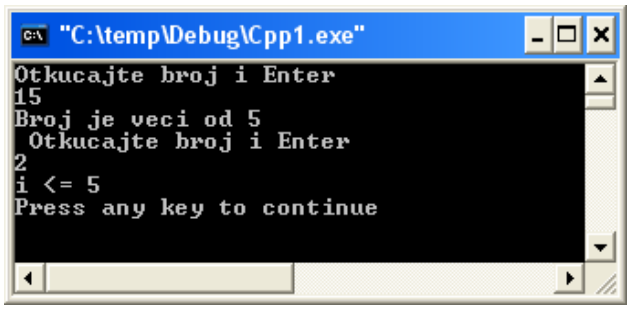


2 Zadatak:

Primer za IF naredbu uz upotrebu lokalnih promenljivih:

```
#include<iostream.h>
#include<math.h>
void main()
{
int i;
cout << "Otkucajte broj i Enter" << endl;
cin >> i;
if(i > 5)
cout << "Broj je veci od 5" << endl;
else
if(i < 5)
cout << "Broj je manji od 5 " << endl;
else
cout << "Broj je jednak 5 " << endl;
cout << " Otkucajte broj i Enter " << endl;
cin >> i;
if(i < 10)
if(i > 5) // "if" je samo sledeca naredba
```

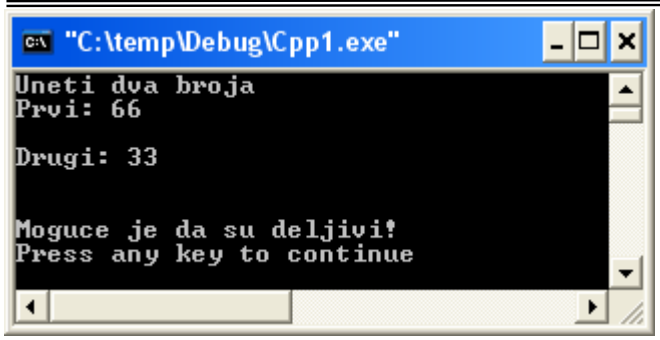
```
cout << "5 < i < 10" << endl;
else
cout << "i <= 5" << endl;
else // Poklapa se "if(i < 10)"
cout << "i >= 10" << endl;
}
```



3 Zadatak:

Kompleksno ugnježdena if naredba

```
#include <iostream.h>
 int main()
 {   // Unosimo dva broja
     // Brojevima dodeljujemo Veliki broj ili Mali broj
     // Ako je Veliki broj veci od  Mali broj
     // videti dali su moyda deljivisee if they are evenly divisible
     // I ako jesu videti dali su mozda to isti brojevi

    int prviBroj, drugiBroj;
    cout << "Uneti dva broja\nPrvi: ";
    cin >> prviBroj;
    cout << "\nDrugi: ";
    cin >> drugiBroj;
    cout << "\n\n";
    if (prviBroj >= drugiBroj)
    {
      if ( (prviBroj % drugiBroj) == 0) // eventualno deljivi?
      {
          if (prviBroj == drugiBroj)
              cout << "Ovo su isti brijevi!\n";
          else
              cout << "Moguce je da su deljivi!\n";
      }
      else
          cout << "Nije moguce da su deljivi!\n";
    }
    else
          cout << "Hey! Drugi broj je veci!\n";
    return 0;
 }
```



4 Zadatak:

Listing za demonstraciju zašto su zagrade važne za ugnježdene if naredbe - KOJI JE PROGRAM DOBAR?

```
#include <iostream.h>
int main()
{
  int x;
cout << "Unesi 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
     cout << " Manji od 10, Hvala!\n";

   return 0;
}
```

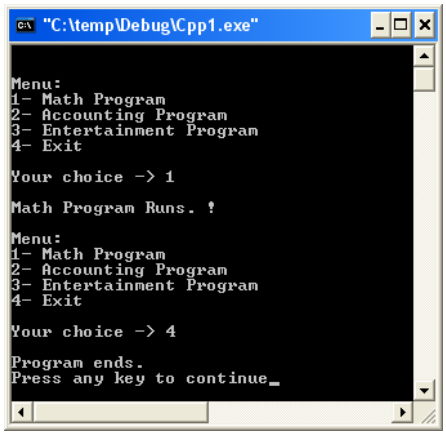
```
#include <iostream.h>
  int main()
  {
    int x;
cout << "Unesi broj manji od 10 ili veći od
100: ";
    cin >> x;
    cout << "\n";

    if (x >= 10)
      if (x > 100)
{cout << "Veći od 100, Hvala!\n";goto kraj;}
  // else                    // nepotrebno
     cout << "Manji od 10, Hvala!\n";
    kraj:
    return 0;
  }
```

5 Zadatak:

Prikaz kompleksne "if" statement

```
#include <iostream.h>
#include <stdlib.h>
main()
{int choice;
while(1)
 {
 cout<<"\n\nMenu:\n";
 cout<<"1- Math Program\n2- Accounting Program\n";
 cout<<"3- Entertainment Program\n4- Exit";
```

```
cout<<"\n\nYour choice -> ";
cin>>choice;
if(choice==1)
  cout<<"\nMath Program Runs. !";
else if(choice==2)
  cout<<"\nAccounting Program Runs. !";
else if(choice==3)
  cout<<"\nEntertainment Program Runs. !";
else if(choice==4)
  {    cout<<"\nProgram ends.\n";
  exit(0);
  }
else
  cout<<"\nInvalid choice";  }}
```

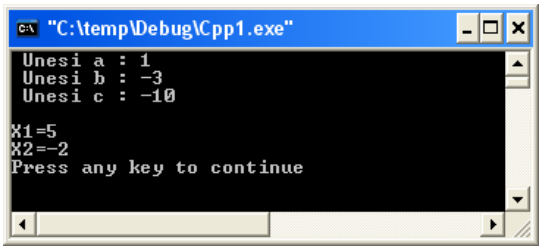


6 Zadatak:

Program za rešavanje kvadratne j-ne $ax^2+bx+c=0$.

```
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
void main()
{
double delta,a,b,c,x1,x2;
cout<<" Unesi a : ";
cin>>a;
cout<<" Unesi b : ";
cin>>b;
cout<<" Unesi c : ";
cin>>c;
delta=b*b-(4*a*c);
if(delta<0)
 {  cout<<"Jednacina nema resenja !\n";
 exit(0);  }
if(delta==0)
 {  x1=-b/(2*a);
```

```
 cout<<"Postoje dva identicna resenja !\n";
 cout<<"x1=x2=%"<<x1;
 exit(0);  }
x1=(-b+sqrt(delta))/(2*a);
x2=(-b-sqrt(delta))/(2*a);
cout<<"\nX1="<<x1;
cout<<"\nX2="<<x2<<endl;}
```

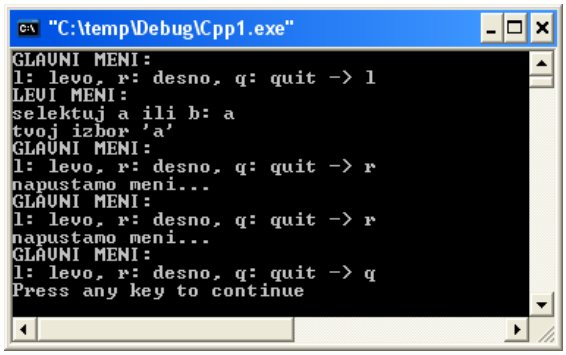


7 Zadatak:

Program za demonstraciju "break" i "continue"

```
#include <iostream.h>
void main()
{
char c; // Za definisanje izbora
while(1)
{
cout << "GLAVNI MENI:" << endl;
cout << "l: levo, r: desno, q: quit -> ";
cin >> c;
if(c == 'q')
break; // Izlaz iz "while(1)"
if(c == 'l') {
cout << "LEVI MENI:" << endl;
cout << "selektuj a ili b: ";
cin >> c;
if(c == 'a') {
cout << "tvoj izbor 'a'" << endl;
continue; // Povratak u glavni meni}
if(c == 'b') {
cout << " tvoj izbor 'b'" << endl;
continue; // Povratak u glavni meni}
else {cout << "nisi izabrao a ili b!"<< endl;
continue; // Povratak u glavni meni}}
if(c == 'r') {
cout << "DESNI MENI:" << endl;
cout << "selektuj c ili d: ";
cin >> c;
if(c == 'c') {
cout << " tvoj izbor 'c'" << endl;
continue; // Povratak u glavni meni}
```

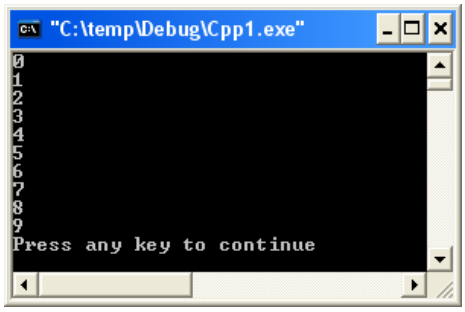
```
if(c == 'd') {
cout << " tvoj izbor 'd'" << endl;
continue; // Povratak u glavni meni}
else {
cout << " nisi izabrao c ili d!"
<< endl;
continue; // Povratak u glavni meni}}
cout << "moras otkucati l ili r ili q!" << endl;}
cout << "napustamo meni..." << endl;}}
```



8 Zadatak:

Primer za FOR naredbu uz upotrebu inkrementa:

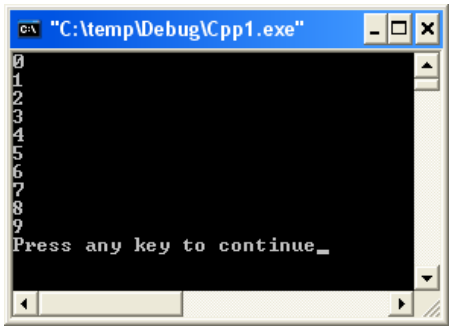
```
#include <iostream.h>
int main()
 {                          //Petlja se vrto dok je x<100, i x se u svakoj petlji povećava za 1
for(int x=0;x<10;x++) //Zapamtite da uslov petlje proverava promenljivu u petlji pre
{cout<<x<<endl;  }                  //Kada je  x jednako 100 petlja se prekida
                                    //Izlazi za  x
  return 0;
}
```



9 Zadatak:

Primer za WHILE naredbu :

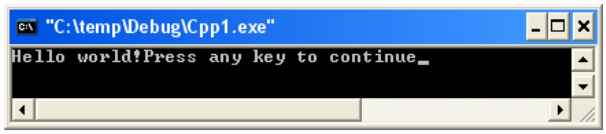
```
#include <iostream.h>
int main()
{
 int x=0;                          //Ne zaboravite da deklarišete promenljivu
 while(x<10)                                  //Dok je x manje od 100 uradi
 {    cout<<x<<endl;               //Isti izlaz kao u petlji gore
   x++;                            //Dodaje 1 na x svaki put kad se petlja ponovi

 }
 return 0;}
```



10 Zadatak:

Primer za DO WHILE naredbu :

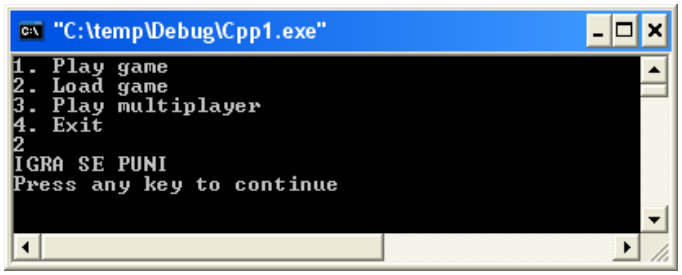
```
#include <iostream.h>
int main()
{  int x;
  x=0;
  do
  { cout<<"Hello world!"; }
while(x!=0);                          //Petlja radi dok x nije nula, ali se prvo izvrši kod u
sekciji:
                                                //Izlaz je "Hello world" jednom
  return 0;}
```



11 Zadatak:

Primer za SWITCH naredbu uz upotrebu case strukture:

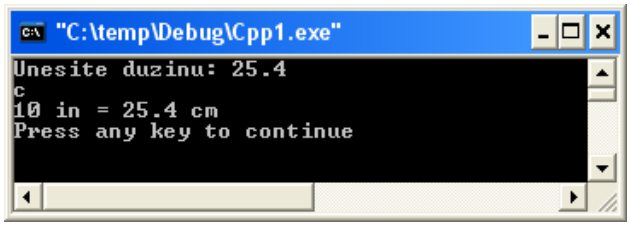
```
#include <iostream.h>
#include <conio.h>
void playgame();
void loadgame();
void playmultiplayer();
int main()
{  int input;
  cout<<"1. Play game"<<endl;
  cout<<"2. Load game"<<endl;
  cout<<"3. Play multiplayer"<<endl;
  cout<<"4. Exit"<<endl;
  cin>>input;
  switch (input)
  {  case 1: playgame();// Paziti na prototipove - obezbediti funkcije
              break;
   case 2:
              loadgame();
       break;
   case 3:
       playmultiplayer();
              break;
   case 4:
       return 0;
  default:   cout<<"GRESKA Los ulaz";
  }
  return 0;}

void playgame()
{cout<<"POCINJE IGRA"<<endl;}
void loadgame()
{cout<<"IGRA SE PUNI"<<endl;}
void playmultiplayer()
{cout<<"IGRA ZA VISE IGRACA"<<endl;}
```



12 Zadatak:

Još jedan primer upotrebe case naredbe

```
#include<iostream.h>
#include<math.h>
int main()
{const double factor = 2.54;                    // 1 inch je 2.54 cm
double x, in, cm;
char ch = 0;
cout << "Unesite duzinu: ";
cin >> x;                                       // citanje floating-point broja
cin >> ch;                                      // citanje sufix
switch (ch)
{
case 'i': // ako se unese i radi se o inch
in = x;
cm = x*factor;
break;
case 'c': // ako se unese i radi se o cm
in = x/factor;
cm = x;
break;
default:
in = cm = 0;
break;
}
cout << in << " in = " << cm << " cm\n";
return 0;}
```



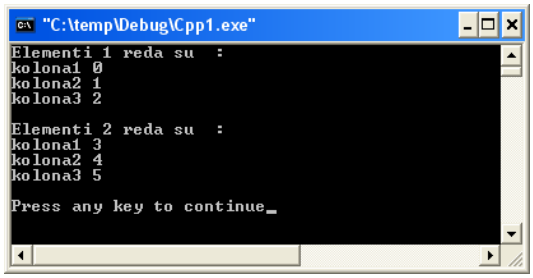
13 Zadatak:

Elementi matrice, kombinovanje for naredbi

```
#include<iostream.h>
#include<math.h>
void main()
{
static int mat[2][3]={{0,1,2},{3,4,5}};
int i,j;
for(i=1;i<=2;++i)
{
cout<<"Elementi "<< i<<" reda su  :\n";
```

```
for(j=1;j<=3;++j)
cout<<"kolona"<<j<<" "<<mat[i-1][j-1]<<"\n";
cout<<"\n";
}}
```

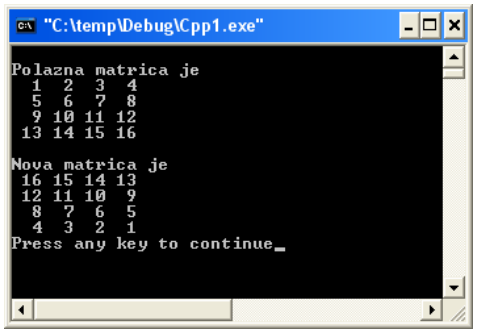
IZLAZ:



14 Zadatak:

Štampanje članova matrice i njenih članova unazad

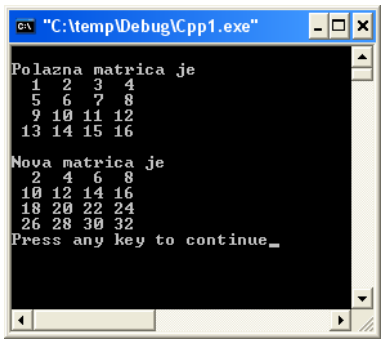
```
#include <iostream.h>
#include<math.h>
#include <iomanip.h>
void main()
{int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int i,j;
cout<<"\nPolazna matrica je\n";
for(i=0;i<4;++i)
{
 for(j=0;j<4;++j)
{cout<<setw(3)<<mat[i][j];}
cout<<"\n";}
cout<<"\nNova matrica je\n";
for(i=3;i>=0;--i)
{
 for(j=3;j>=0;--j)
{cout<<setw(3)<<mat[i][j];}
cout<<"\n";}}
```



15 Zadatak:

Štampanje članova matrice i njenih članova pomnoženih skalarom

```
#include <iostream.h>
#include <math.h>
#include <iomanip.h>
void main()
 {int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
 int i,j;
 cout<<"\nPolazna matrica je\n";
 for(i=0;i<4;++i)
{
 for(j=0;j<4;++j)
{cout<<setw(3)<<mat[i][j];}
cout<<"\n";}
 cout<<"\nNova matrica je\n";
for(i=0;i<4;++i)
{
 for(j=0;j<4;++j)
{cout<<setw(3)<<2*mat[i][j];}
cout<<"\n";}}
```



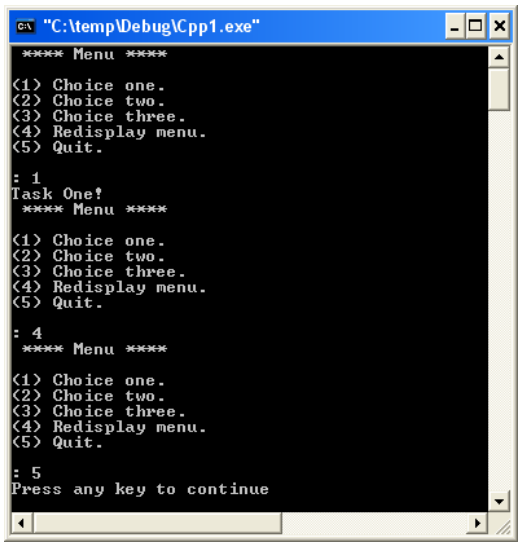
16 Zadatak:

Korišćenje neprekidne petlje koja se upravlja korisničkom interakcijom

```
#include <iostream.h>
// prototypes
int menu();
void DoTaskOne();
void DoTaskMany(int);

int main()
{

   bool exit = false;
   for (;;)
   {
```

```
        int choice = menu();
        switch(choice)
        {
          case (1):
              DoTaskOne();
              break;
          case (2):
              DoTaskMany(2);
              break;
          case (3):
              DoTaskMany(3);
              break;
          case (4):
              continue;  // redundant!
              break;
          case (5):
              exit=true;
              break;
          default:
              cout << "Please select again!\n";
              break;
        }         // end switch

        if (exit)
             break;
    }               // end forever
   return 0;
}                   // end main()



int menu()//definisanje funkcije
  {
     int choice;

     cout << " **** Menu ****\n\n";
     cout << "(1) Choice one.\n";
     cout << "(2) Choice two.\n";
     cout << "(3) Choice three.\n";
     cout << "(4) Redisplay menu.\n";
     cout << "(5) Quit.\n\n";
     cout << ": ";
     cin >> choice;
     return choice;
  }

  void DoTaskOne()
  {
     cout << "Task One!\n";
  }

  void DoTaskMany(int which)
  {
```

```
    if (which == 2)
        cout << "Task Two!\n";
    else
        cout << "Task Three!\n";
}
```

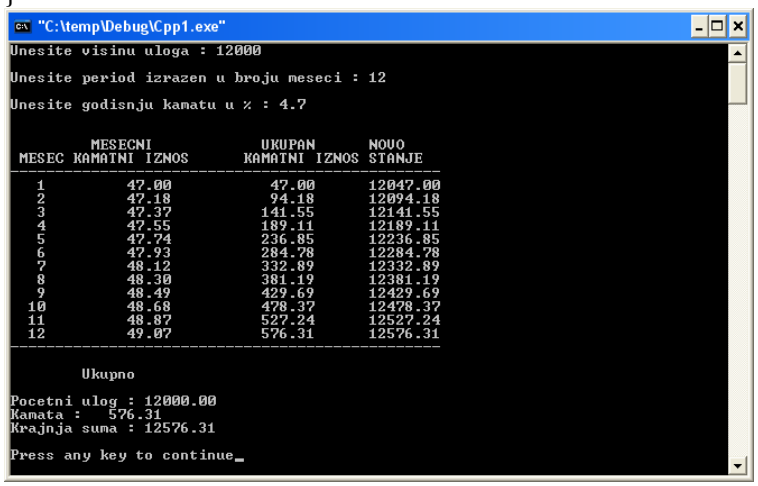


17 Zadatak:

Napisati program koji izračunava mesečnu kamatu na štednju primenom for petlje. Uslov za izvršavanje petlje se nalazi iza ključne reči for. Primetimo da se ispisivanje rezultata i izračunavanja takođe vrše u okviru for petlje, što se često koristi u cilju racionalizacije programskog koda.

```
// Program ilustruje upotrebu for petlje za izracunavanje mesecne kamate.
// Korisnik unosi visinu uloga, rok i kamatnu stopu.
// Program daje listu mesecnih kamata, ukupnu kamatu i stanje racuna za svaki period.
#include <iostream.h>
#include <iomanip.h>
int main()
{
//Deklarisanje promenljivih
double godisnja_kamata, // Godisnja kamata u %
mesecna_kamata, // Mesecna kamata
ulog, // Visina uloga
stanje_racuna, // Mesecno stanje racuna
kamatni_iznos, // Kamatni iznos
ukupna_kamata; // Ukupna kamata
int mesec, // Racuna period prikazan na izvodu sa racuna
period; // Period na izvodu sa racuna u mesecima
//Podesavanje izlaznog formata za ispisivanja iznosa
cout << setprecision(2)
<< setiosflags(ios::fixed)
```

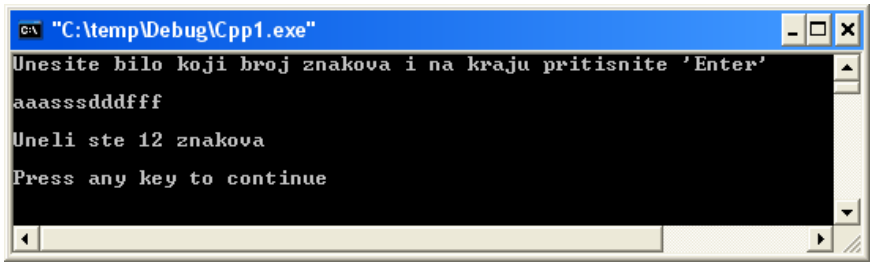
```
<< setiosflags(ios::showpoint);
// Unos podataka
cout << "Unesite visinu uloga : ";
cin >> ulog;
cout << endl;
cout << "Unesite period izrazen u broju meseci : ";
cin >> period;
cout << endl;
cout << "Unesite godisnju kamatu u % : ";
cin >> godisnja_kamata;
// Izracunavanja i ispisivanje rezultata
mesecna_kamata = (godisnja_kamata / 100) / 12;
ukupna_kamata = 0.00;
stanje_racuna = ulog;
cout << endl << endl;
cout << "        MESECNI          UKUPAN      NOVO" << endl;
cout << " MESEC KAMATNI IZNOS      KAMATNI IZNOS STANJE" << endl;
cout << "-----------------------------------------------";
for (mesec = 1; mesec <= period; ++mesec)
{kamatni_iznos = mesecna_kamata * stanje_racuna;
stanje_racuna += kamatni_iznos;
ukupna_kamata += kamatni_iznos;
cout << endl << setw(4) << mesec
<< setw(14) << kamatni_iznos
<< setw(16) << ukupna_kamata
<< setw(14) << stanje_racuna;}
cout << endl;
cout << "-----------------------------------------------" << endl
<< endl;
cout << "\tUkupno" << endl << endl;
cout << "Pocetni ulog : " << setw(8) << ulog << endl;
cout << "Kamata : " << setw(8) << ukupna_kamata << endl;
cout << "Krajnja suma : " << setw(8) << stanje_racuna << endl
<< endl;
return 0;
}
```



18 Zadatak:

Napisati program koji broji znakove unete sa konzole u jednoj liniji. U programu koristiti while petlju.
U prikazanoj while petlji, uslov koji mora biti ispunjen da bi se petlja izvršavala, nalazi se na početku petlje. Ukoliko uslov nije ispunjen, sekvenca naredbi u okviru petlje se neće ni jednom izvršiti.

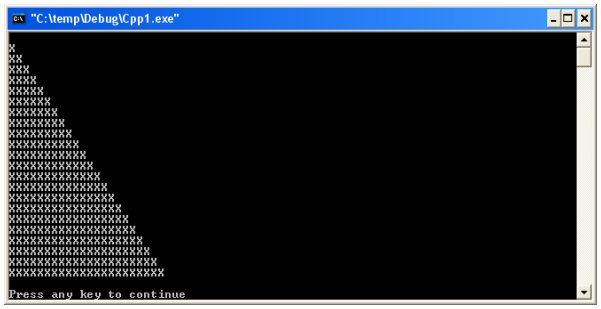
```
// Program broji unete znakove koje je upisao korisnik u jednoj liniji
// Linija se zavrsava pritiskom na Enter. Enter se ne racuna kao karakter
// Primer iteracije.
#include <iostream.h>
int main()
{
// Deklarisanje promenljivih
char znak; // Koristi se za prihvat unetog znaka
int broj_znakova = 0; // Broji unete znakove
// Unos niza
cout << "Unesite bilo koji broj znakova i na kraju pritisnite "
<< "'Enter'"
<< endl << endl;
znak = cin.get(); // Unos prvog znaka
// Petlja za sabiranje znakova unetih u jednom redu
while (znak != '\n') // While uslov se nalazi na pocetku
// petlje
{
++broj_znakova;
znak = cin.get(); // Unos sledeceg znaka
}
// Ispisivanje rezultata
cout << endl;
cout << "Uneli ste " << broj_znakova << " znakova" << endl
<< endl;
return 0;
}
```



19 Zadatak:

Napisati program koji crta pravougli trougao znacima ‘X’ korišćenjem ugneždene for petlje.
Kao i while petlja u prethodnom primeru, tako i for petlja može biti unutar spoljašnje for petlje.

```
// Program demonstrira ugnezdenu petlju
// prikazujuci redove koji se sastoje od znaka X.
// Broj iteracija unutrasnje for petlje zavisi od vrednosti brojaca
// koji kontrolise spoljasnju for petlju.
#include <iostream>
using namespace std;
int main()
{
// Deklarisanje promenljivih
const int BROJ_REDOVA = 22;
int trenutni_broj_koraka,
brojac_x;
// Ispisivanje rezultata
for (trenutni_broj_koraka = 1; trenutni_broj_koraka <=
BROJ_REDOVA; ++trenutni_broj_koraka)
{cout << endl;
for (brojac_x = 1; brojac_x <= trenutni_broj_koraka;
++brojac_x)
cout << "X";}
cout << endl << endl;
return 0;
}
```



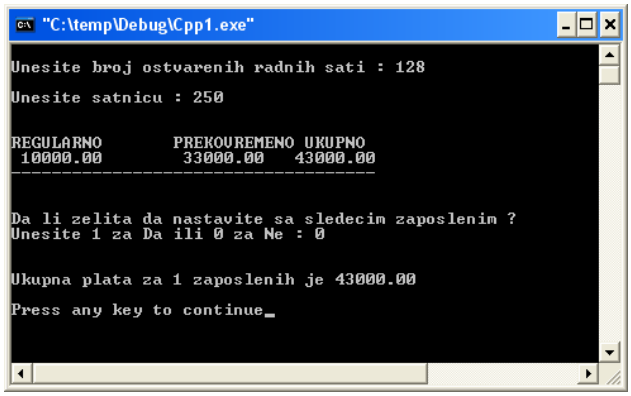
20 Zadatak:

Napisati program koji računa platu za zaposlene. Uzeti u obzir da se prekovremeni rad plaća 50% više od redovnog. Predvideti mogućnost obrade podataka za više zaposlenih, kao i sumiranje plata na kraju programa.

```
// Program izracunava ukupnu platu za svakog zaposlenog
// ukljucujuci i prekovremeni rad koji se placa 1.5 puta vise.
```

```
// Program takodje izracunava ukupne plate za sve obradjne zaposlene.
// Program ilustruje upotrebu if naredbe.

#include <iostream.h>
#include <iomanip.h>
int main()
{
// Deklarisanje promenljivih
const double FAKTOR_PREKOVREMENOG_RADA = 1.5;
const double REGULARAN_BROJ_SATI = 40.0;
int brojac_zaposlenih,
sledeci_zaposleni; // 1 ako postoji sledeci;
// 0 ako ne postoji
double radni_sati,
satnica,
regularna_plata,
prekovremena_plata,
ukupna_plata,
sve_plate;
// Inicijalizacija promenljivih
sve_plate = 0.00;
brojac_zaposlenih = 0;
//Podesavanje izlaznog formata za ispisivanja iznosa
cout << setprecision(2)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
do // Pocetak while petlje
{
// Unos podataka
cout << endl;
cout << "Unesite broj ostvarenih radnih sati : ";
cin >> radni_sati;
cout << "\nUnesite satnicu : ";
cin >> satnica;
// Izracunavanje plate
if (radni_sati > REGULARAN_BROJ_SATI)
{
regularna_plata = REGULARAN_BROJ_SATI * satnica;
prekovremena_plata = (radni_sati - REGULARAN_BROJ_SATI) *
FAKTOR_PREKOVREMENOG_RADA * satnica;
}
else
{regularna_plata = radni_sati * satnica;
prekovremena_plata = 0.00;}
ukupna_plata = regularna_plata + prekovremena_plata;
sve_plate += ukupna_plata;
++brojac_zaposlenih;
// Ispisivanje plate
cout << endl << endl;
cout << "REGULARNO PREKOVREMENO UKUPNO";
cout << endl << setw(9) << regularna_plata
<< setw(16) << prekovremena_plata
<< setw(11) << ukupna_plata << endl;
```

```
cout << "-----------------------------------" << endl;
// Pitanje korisniku da li zeli da nastavi sa sledecim
// zaposlenim
cout << endl << endl;
cout << "Da li zelita da nastavite sa sledecim zaposlenim ?"
<< endl;
cout << "Unesite 1 za Da ili 0 za Ne : ";
cin >> sledeci_zaposleni;
}
while (sledeci_zaposleni); // Uslov while petlje - na kraju
// bloka naredbi
// Ispisivanje zbira svih plata
cout << endl << endl;
cout << "Ukupna plata za " << brojac_zaposlenih
<< " zaposlenih je " << sve_plate << endl << endl;
return 0;
}
```

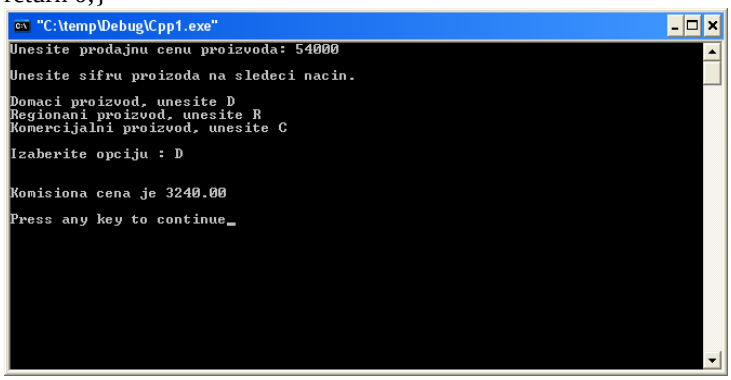


21 Zadatak:

Napisati program koji izračunava komisionu cenu u zavisnosti od tipa robe.

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
int main()
{
// Deklarisanje promenljivih
const double DOMACA_STOPA = 0.060;
const double REGIONALNA_STOPA = 0.050;
const double KOMERCIJALNA_STOPA = 0.045;
int kod_robe;
double prodajna_cena,
komisiona_stopa,
komisiona_cena;
```

```
//Podesavanje izlaznog formata za ispisivanja iznosa
cout << setprecision(2)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
// Unos podataka
cout << "Unesite prodajnu cenu proizvoda: ";
cin >> prodajna_cena;
cout << endl;
cout << "Unesite sifru proizoda na sledeci nacin."
<< endl << endl;
cout << "Domaci proizvod, unesite D" << endl;
cout << "Regionani proizvod, unesite R" << endl;
cout << "Komercijalni proizvod, unesite C" << endl << endl;
cout << "Izaberite opciju : ";
cin.get();
kod_robe = cin.get();
// Izracunavanja
switch (kod_robe)
{case 'D':
case 'd':
komisiona_stopa = DOMACA_STOPA;
break;
case 'R':
case 'r':
komisiona_stopa = REGIONALNA_STOPA;
break;
case 'C':
case 'c':
komisiona_stopa = KOMERCIJALNA_STOPA;
break;
default:
cout << endl << endl
<<"Neispravna sifra proizvoda! Pokusajte ponovo" << endl;
exit(1);
break;}
komisiona_cena = prodajna_cena * komisiona_stopa;
// Ispisivanje rezultata
cout << endl << endl;
cout << "Komisiona cena je " << komisiona_cena << endl << endl;
return 0;}
```

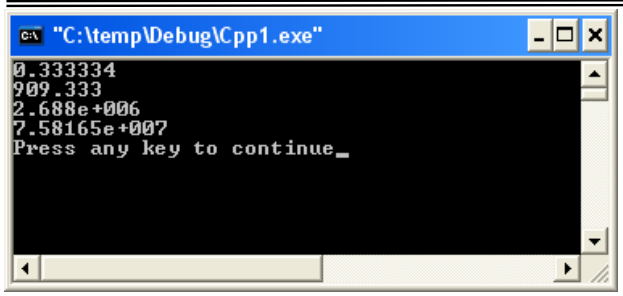


22 Zadatak:

Izračunati vrednost integrala

$\int_a^b x^2 dx$ za granice integraljenja a=0, b=1; a=-10, b=12; a=-200, b=40 i a=121, b=612 funkcijom **double integrate** sa brojem iteracija 500.

```
#include <iostream>
#include <math.h>
using namespace std;

double xSquared(double x)
{    // power funkcija, se mo\e yameniti sa x * x
   return pow(x, 2);}

double integrate(double (*pFunc)(double), double nDonjaGranica,
       double nGornjaGranica, long nIteracija = 500)
{
   // Formiranje totalnog broja intervala koji se koristi
   double nInterval = abs(nGornjaGranica - nDonjaGranica) * nIteracija;

   // Specifikacija sirine svakog intervala (delta x)
   double nIntervalWidth = (nGornjaGranica - nDonjaGranica) / nInterval;

   // variabla u kojoj se cuva totalna suma
   double nTotalSum = 0;

   for(double i = 0, j = nDonjaGranica; i < nInterval; i++)
   {
      // temporary variabla za cuvanje sledeceg subintervala
      double k = j + nIntervalWidth;

      // dodavanje  sumi
      nTotalSum += (nIntervalWidth / 2) * ( pFunc(j) + pFunc(k) );

      // setovanje j na sledeci subinterval
      j = k;
   }

   return nTotalSum;
}

int main(int argc, char *argv[])
{

   cout << integrate(xSquared, 0, 1) << endl;
   cout << integrate(xSquared, -10, 12) << endl;
   cout << integrate(xSquared, -200, 40) << endl;
   cout << integrate(xSquared, 121, 612) << endl;

   return 0;}
```



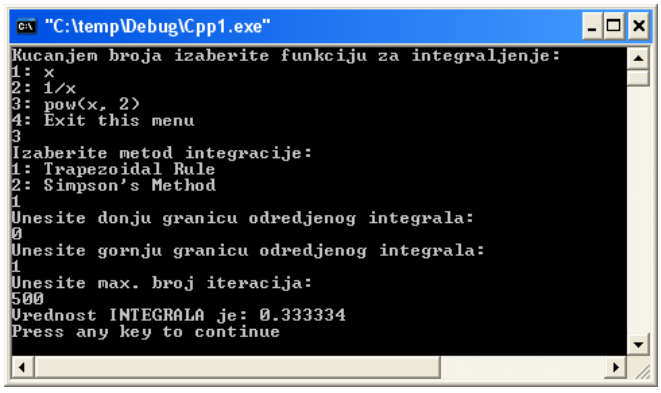
23 Zadatak:

Program nudi korisnicima 2 menija sa listom opcija koje moze da izabere. Prva opcija je u izboru jedne od podintegralnih funkcija koju resavamo a finalna opcija je u izboru metoda integracije. Rezultat takve numericke integracije se na izlazu pojavljuje u decimalnoj formi.

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <string>
#include <cmath>
using namespace std;

void getlimit(double& lower, double& upper, int& imax); // prototip funkcije ogranicenja


 double trapezoid(double(*f)(double), double a, double b, int imax)  // kodni blok za upotrebu trapezoidalnog metoda integracije
                                    {
                                       double r, dx, x;
                                       r = 0.0;
                                       dx = (b-a)/static_cast<float>(imax);

                                       for (int i = 1; i <= imax-1; i = i+1)
                                       {
                                        x = a + static_cast<float>(i)*dx;
                                        r = r + f(x);
                                       }
                                       r = (r + (f(a)+f(b))/2.0)* dx;
                                       return r;
                                    }

        double simpson(double(*f)(double), double a, double b, int imax) // kodni blok za upotrebu simpsonove metode integracije
                                    {
                                       double s, dx, x;
                                    // za neparno n dodaje se +1 da bi interval bio paran
                                       if((imax/2)*2 != imax) {imax=imax+1;}
```

```
                    s = 0.0;
                    dx = (b-a)/static_cast<float>(imax);
                    for (int i = 2; i <= imax-1; i = i+2)
                    {
                        x = a+static_cast<float>(i)*dx;
                        s = s + 2.0*f(x) + 4.0*f(x+dx);
                    }
                    s = (s + f(a)+f(b)+4.0*f(a+dx) )*dx/3.0;
                    return s;
                }

double fun1(double x) // definise funkciju 1
{
    return x;
}
double fun2(double x) // definise funkciju 2
{
    return 1 / x;
}
double fun3(double x) // definise funkciju 3
{
    return x * x;
}
int main()
{
    string str1 = "x";
    string str2 = "1/x";
    string str3 = "pow(x, 2)";
    double(*f)(double)  = 0;
    double a, b, lower, upper,  integralValue;
    int method, equation, imax;
    const int PICK_EQUATION = 4;
    const int MAXMETHODS = 2;

        cout << "Kucanjem broja izaberite funkciju za integraljenje:\n";
        cout << "1: " << str1 << " \n";
        cout << "2: " << str2 << " \n";
        cout << "3: " << str3 << " \n";
        cout << "4: Exit this menu\n";
        cin >> equation;
        if (equation == 1) // govori programu da ukljuci funkciju #1
            f = fun1;
        else if (equation == 2) // govori programu da ukljuci funkciju #2
            f = fun2;
        else            // govori programu da ukljuci funkciju #3
            f = fun3;

        cout << "Izaberite metod integracije:\n";
        cout << "1: Trapezoidal Rule" << endl; // govori programu da ukljuci
trapezoidalni metod integracije
        cout << "2: Simpson's Method" << endl; // govori programu da ukljuci
simpsonov metod integracije
        cin >> method;
```

```
        getlimit(lower, upper, imax);

        if(method <= MAXMETHODS)
                {
                switch(method)
                        {
                        case 1:
                           integralValue = trapezoid(f, lower,upper, imax);
                           break;
                        case 2:
                           integralValue = simpson(f, lower, upper, imax);
                           break;
                        }
                   cout << "Vrednost INTEGRALA je: " << integralValue << endl;
                }



        return 0;
}

void getlimit(double& lower, double& upper, int& imax)
{
        cout << "Unesite donju granicu odredjenog integrala: \n"; // takes the users input
for lower bound
        cin >> lower;
        cout << "Unesite gornju granicu odredjenog integrala: \n"; // takes the users input
for upper bound
        cin >> upper;
        cout << "Unesite max. broj iteracija: \n"; // takes the users input for maximum
number of iterations
        cin >> imax;

        return;
}
```



RAD SA STRINGOM

ASCII string

Do sada smo radili sa nizovima znakova u obliku tekstualnih konstanti koje smo zvali literalni string ("Hello World!"). Cilj nam je da nizove znakova tretiramo kao varijable. Na taj način moći ćemo vršiti obradu tekstualnih zapisa.
U C++ jeziku se koriste dva načina rada sa stringovima.

• Prvi način je da se string tretira kao niz znakova kojem poslednji znak mora biti jednak nuli. Ovakovi stringovi se nazivaji i ASCIIZ stringovi (ASCII +zero). U standardnoj biblioteci C jezika postoji više funkcija za rad s ovakovim stringovima, a deklarisane su u datoteci <string.h> odnosno <cstring>.

• Drugi je način, standardiziran u C++ jeziku, da se koristi klasa string koja je deklarisana u datoteci <string>.

Najpre ćemo upoznati kako se manipuliše stringom u C jeziku. Funkcije C- jezika tretiraju string kao za memorijski objekt koji sadrži niz znakova, uz uslov da poslednji znak u nizu mora biti jednak nuli. Nula na kraju stringa se koristi kao oznaka kraja stringa. Naprimer, u stringu koji sadrži tekst: Hello, World!, indeks nultog znaka je 13, što je jednako dužini stringa. Ovaj string u memoriji zauzima 14 bajta.

0	1	2	3	4	5	6	7	8	9	0	1	2	3
'H'	'e'	'l'	'l'	'o'	', '	' '	'W'	'o'	'r'	'l'	'd'	'!'	'\0'

Indeks nultog znaka je upravo jednak broju znakova u stringu.
Prema ovoj definiciji ASCIIZ string je svaki niz koji se deklariše kao:

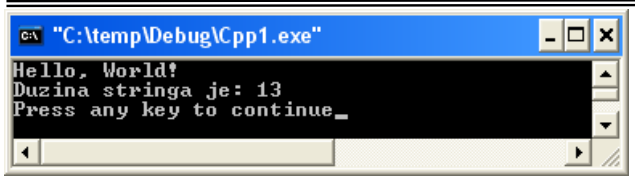
• niz znakova (npr. char str[10];) , ili kao
• pokazivač na znak (char *str;),

pod uslovom da se pri inicijalizaciji niza, i kasnije u radu s nizom uvek vodi računa o tome da polednji element niza mora biti jednak nuli.

Pogledajmo primer u kojem se string tretira kao obični niz znakova.

```
//Prvi C++ program - drugi put.
#include <iostream>
#include <cstring> // deklaracija funkcije strlen
using namespace std;
int main()
{
char hello[14] = { 'H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!', '\0' };
cout << hello << endl;
cout << "Duzina stringa je: " << strlen(hello) << endl;
return 0;
}
```

Posle izvršenja programa, dobije se poruka:



Ovaj program vrši istu funkciju kao i naš prvi C-program, štampa poruku: Hello, World!. Prvo je definisana i inicijalizovana varijabla hello. Ona je tipa znakovnog niza od 14 elemenata. Inicijalizacijom se u prvih 13 elemenata upisuju znakovi (Hello World), poslednji element se inicira na nultu vrednost.
Ispis ove varijable : Za ispis dužine stringa korišćena je standardna funkcija size_t strlen(char *s); koja vraća vrednost dužine stringa. Kao argument funkcija prihvata vrednost pokazivača na char, odnosno adresu početnog elementa stringa. (size_t je sinonim za unsigned).
Funkcija strlen se može implementirati na sledeći način:

1.verzija:

```
unsigned strlen(char s[])
{
unsigned i=0;
while (s[i] != '\0')          // petlja se prekida kada je s[i]==0, inače
i++;                          // inkrementira se brojač znakova, koji na kraju
return i;                     // sadrži duzinu stringa (bez nultog znaka)
}
```

2.verzija: pomoću pokazivača

```
unsigned strlen(char *s)
{
unsigned len = 0;
while (*s++ != '\0')          // petlja se prekida kada je *s==0, inače
len++;                        // inkrementira se brojač znakova i pokazivač
return len;                   // len sadrži duljinu stringa (bez nultog znaka)
}
```

String se može inicijalizirati navođenjem znakova od kojih se sastoji, kao u prethodnom primeru, ili pomoću literalne konstante:

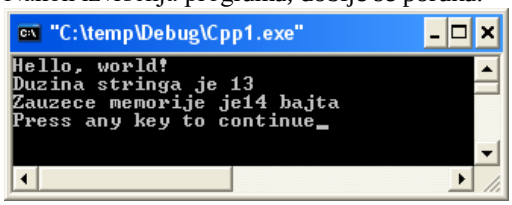
```
char hello[] = "Hello, World!"; // kompajler rezerviđe mesto u memoriji
```

Primer:

```
//Prvi C++ program - drugi put.
#include <iostream>
using namespace std;
unsigned strlen(char *s)
{
unsigned len = 0;
while (*s++ != '\0') // petlja se prekida kada je *s==0, inače
len++;               // inkrementira se brojač znakova i pokazivač
return len;          // len sadrži duzinu stringa (bez nultog znaka)
}
```

```
int main()
{
char hello[] = "Hello, world!";
cout << hello << endl;
cout << "Duzina stringa je " << strlen(hello) << endl;
cout << "Zauzece memorije je" << sizeof(hello)*sizeof(char)
<< " bajta" << endl;
return 0;
}
```

Nakon izvršenja programa, dobije se poruka:



Elementi stringa se mogu menjati naredbom dodele vrednosti, naprimer:

```
hello[0] = 'h';
hello[6] = 'w';
```
menjaju sadržaj stringa u "hello world";

Ako se procenjuje da će trebati više mesta za string, nego što se to navodi inicijalnim literalnim stringom, tada treba eksplicitno navesti dimenziju stringa. Deklaracijom,

```
char hello[50] = "Hello, World!";
```

kompajler rezerviše 50 mesta u memoriji i inicira prvih 14 elemenata na "Hello, World!", zaključno s nultim znakom.
Dozvoljeno je literalni string koristiti kao referencu niza:

```
cout << "0123456789ABCDEF"[n]; <=> char digits[] ="0123456789ABCDEF";
cout << digits[n];
```

Sledeća dva primera pokazuju rad s stringovima. Biće opisano kako se mogu napisati standardne funkcije za kopiranje stringa (strcpy) i upoređenje dva stringa (strcmp).

Funkcija strcpy kopira znakove stringa src u string dest, a vraća adresu od dest.

1. Verzija sa nizovima:

```
char *strcpy( char dst[], char src[])
{
int i;
for (i = 0; src[i] != '\0'; i++)
dst[i] = src[i];
dst[i] = '\0';
return dst;
}
```

2. Verzija sa pokazivačkim varijablama

```
char *strcpy(char *dest, const char *src)
{
char *dp = dest;
while((*dp++ = *src++) != '\0') // kopira se i '\0'
; /*prazna naredba*/
return dest;
}
```

Ova verzija sa inkrementiranjem pokazivača će se brže izvršavati od verzije koja je zapisana u indeksnoj notaciji.
Funkcija strcmp služi za upoređenje dva stringa s1 i s2. Deklarisana je s:

```
int strcmp(const char *s1, const char *s2)
```

Funkcija vraća vrednost 0 ako je sadržaj oba stringa isti, negativnu vrednost ako je s1 leksički manji od s2, ili pozitivnu vrednost ako je s1 leksički veći od s2. Leksičko poređenje se izvršava znak po znak prema kodnoj vrednosti ASCII standarda.

```
/*1. verzija*/
int strcmp( char s1[], char s2[])
{
int i = 0;
while(s1[i] == s2[i] && s1[i] != '\0')
i++;
if (s1[i] < s2[i])
return -1;
else if (s1[i] > s2[i])
return +1;
else
return 0;
}
```

Najprije se u for petlji poređuje da li su znakovi sa istim indeksom isti i da li je dostignut kraj niza (znak '\0'). Kad petlja završi, ako su oba znaka jednaka, to znači da su i svi prethodni znakovi jednaki. U tom slučaju funkcija vraća vrednost 0. U suprotnom funkcija vraća 1 ili –1 zavisno od numeričkog koda znakova koji se razlikuju.

```
/*2. verzija*/
int strcmp(char *s1, char *s2)
{
for(; *s1 == *s2; s1++, s2++)
{
if(*s1 == '\0')
return 0;
}
return *s1 - *s2;
}
```

Verzija s pokazivačima predstavlja optimiranu verziju prethodne funkcije. Uočite da se u **for** petlji ne koristi izraz inicijalizacije petlje. Najpre se poređuju znakovi na koje pokazuju s1 i s2. Ako su znakovi isti inkrementira se vrednost oba pokazivača. Telo petlje će

se izvršiti samo u slučaju ako su stringovi isti (tada s1 i s2 konačno pokazuju na znak '\0'). U suprotnom petlja se prekida, a funkcija vraća numeričku razliku ASCII koda znakova koji se razlikuju.

Napomena: verzije s inkrementiranjem pokazivača često rezultiraju bržim izvršenjem programa, ali predstavljaju programe koje je nešto teže razumeti od verzija sa indeksiranim nizovima.

Standardne funkcije za rad s ASCIIZ stringovima

U standardnoj biblioteci postoji niz funkcija za manipulisanje sa stringovima . One su deklarisane u datoteci <cstring> ili <string.h>. Funkcija im je:

size_t strlen(const char *s)
Vraća dužinu stringa s.

char *strcpy(char *s, const char *t)
Kopira string t u string s, uključujući '\0'; vraća s.

char *strncpy(char *s, const char *t, size_t n)
Kopira najviše n znakova stringa t u s; vraća s. Dopunjava string s sa '\0' znakovima ako t ima manje od n znakova.

char *strcat(char *s, const char *t)
Dodaje string t na kraj stringa s; vraća s.

char *strncat(char *s, const char *t, size_t n)
Dodaje najviše n znakova stringa t na string s, i znak '\0'; vraća s.

int strcmp(const char *s, const char *t)
Upoređuje string s sa stringom t, vraća <0 ako je s<t, 0 ako je s==t, ili >0 ako je s>t. Poređenje je leksikografsko, prema ASCII "abecedi".

int strncmp(const char *s, const char *t, size_t n)
Upoređuje najviše n znakova stringa s sa stringom t; vraća <0 ako je s<t, 0 ako je s==t, ili >0 ako je s>t.

char *strchr(const char *s, int c)
Vraća pokazivač na prvu pojavnost znaka c u stringu s, ili NULL znak ako c nije sadržan u stringu s.

char *strrchr(const char *s, int c)
Vraća pokazivač na zadnju pojavnost znaka c u stringu s, ili NULL znak ako c nije sadržan u stringu s.

char *strstr(const char *s, const char *t)
Vraća pokazivač na prvu pojavu stringa t u stringu s, ili NULL ako string s ne sadrži string t.

size_t strspn(const char *s, const char *t)
Vraća dužinu prefiksa stringa s koji sadrži znakove koji čine string t.

size_t strcspn(const char *s, const char *t)
Vraća dužinu prefiksa stringa s koji sadrži znakove koji nisu prisutni u stringu t.

char *strpbrk(const char *s, const char *t)
Vraća pokazivač na prvu pojavu bilo kojeg znaka iz string t u stringu s, ili NULL ako nije prisutan ni jedan znak iz string t u stringu s.

char *strerror(int n)
Vraća pokazivač na string kojeg interno generiše kompajler za dojavu greške u nekim sistemskim operacijama. Argument je obično globalna varijabla errno, čiju vrednost takođe postavlja kompajler pri sistemskim operacijama.

char *strtok(char *s, const char *sep)
strtok je funkcija kojom se može izvršiti razlaganje stringa na niz leksema koji su razdvojeni znakovima-separatorima. Skup znakova-separatora se zadaje u stringu sep. Funkcija vraća pokazivač na leksem ili NULL ako nema leksema.
Korištenje funkcije strtok je specifično jer u strigu može biti više leksema, a ona vraća pokazivač na jedan leksem. Da bi se dobili sedeći leksemi treba ponovo zvati istu funkciju, ali sa prvim argumentom jednakim NULL. Primeri, za string
char * s = "Prvi drugi,treci";
ako odaberemo znakove separatore: razmak, tab i zarez, tada sledeći iskazi daju ispis tri leksema (Prvi drugi i treci):

```
char *leksem = strtoken(s, " ,\t"); /* dobavlja prvi leksem */
while( leksem != NULL) {          /* ukoliko postoji */
printf("", leksem);               /* ispiši ga i */
lexem = strtok(NULL, " ,\t");     /* dobavi sledeći leksem */
}                                 /* pa ponovi postupak */
```

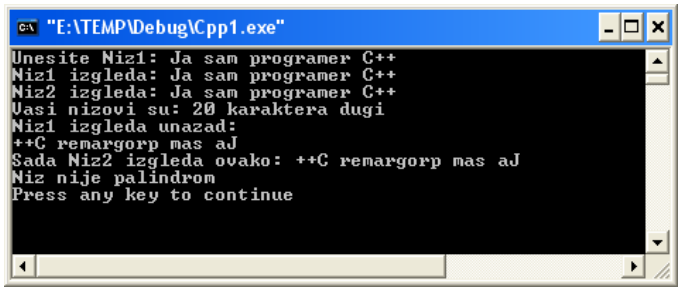
1 Zadatak:

Sastaviti program na C++ jeziku za unos skupa karaktera ”anavolimilovana” u promenljivu Niz1, pa ga onda iskopirati u promenljivu Niz2 korišćenjem funkcije strcpu. Odštampati sadržaj unete promenljive Niz 1, programski realizovane promenljive Niz 2 i odrediti njihovu dužinu primenom funkcije strlen. Dalje odštampati kako izgleda Niz1 unazad (od zadnjeg do prvog karaktera) pa takav niz smestiti u promenljivu Niz2. Odštampati kako sada izgleda promenljiva Niz2.

Opciono - primenom funkcije strcmp(Niz1,Niz2) utvrditi da li su sada ovako dobijeni nizovi palindromi?

```
#include<iostream.h>
#include<string.h>
int main()
{
int i;
char Niz1[40];
char Niz2[40];
cout << "Unesite Niz1: ";
cin.get(Niz1,39); //uzima do 39 karaktera ili novog reda - return
strcpy(Niz2,Niz1);
```

```
cout << "Niz1 izgleda: " << Niz1 << endl;
cout << "Niz2 izgleda: " << Niz2 << endl;
cout << "Vasi nizovi su: " << strlen(Niz1) << " karaktera dugi " << endl;
cout << "Niz1 izgleda unazad: " << endl;
for(i=strlen(Niz1)-1;i>=0;i--)
{cout<<Niz1[i];
Niz2[strlen(Niz1)-i-1]=Niz1[i];
Niz2[strlen(Niz1)]='\0';}
cout<<endl;
cout << "Sada Niz2 izgleda ovako: " << Niz2 << endl;
if(!strcmp(Niz1, Niz2)) cout<<"Niz2 je palindrom sa Niz1" <<endl;
else cout<<"Niz nije palindrom\n";
return 0;
}
```



Mnogi programi koriste jedinstvena imena za sve entitiete što može da izazove velike probleme. To je istina, kada se aplikacija paralelno razvijaa sa strane nekoliko programera, ili kada distributer hoće da produkuje biblioteku koja se odnosi na te programe. Mehanizam koji može pomoći da se ovi problemi reše sastoji se u upotrebi funkcije **nemespace**, u obliku:

using namespace std, ili sa drugim atributima;

NAPRIMER:

```
// Using a namespace
namespace myRegion
{
.....
.....
}

// Using a namespace
namespace data
{
  extern const double pi = 3.14159265;
  extern const int primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31};
}
```

Ovde smo definisali pi i primes[] u okviru namespace data. Možemo koristiti ove promenljive i u drugoj programskoj jedinici koja sada sadrži:

```
// Using a namespace
#include <iostream>
```

```
namespace data
{
 extern const double pi; // Variable su definisane u drugom fajlu
 extern const int primes[]; // Array je definisano u drugom fajlu
}

int main()
{
 std::cout << std::endl
        << "pi ima vrednost "
        << data::pi << std::endl;  //Posmatra se kao konstruktor
 std::cout << "Četvrti prime broj je "
        << data::primes[3] << std::endl;
 return 0;
}
```

Prvo moramo postaviti relaciju imezmeđu naredbe ***namespaces*** i načina na koji će se uključivati hederski fajlovi. Pre nego što je standardizovan novi stil hederskog fajla **<iostream>** (gde nema ekstenzije '**.h**'), tipični način uključivanja hederskog fajla bio je sa ekstenzijom '**.h**', kao naprimer **<iostream.h>**. U tom slučaju, ***namespaces*** nije deo predeprocesorskih direktiva. Na taj način se može utvrditi analogija između ova dva vida pisanja koda, pa ako napišemo:

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

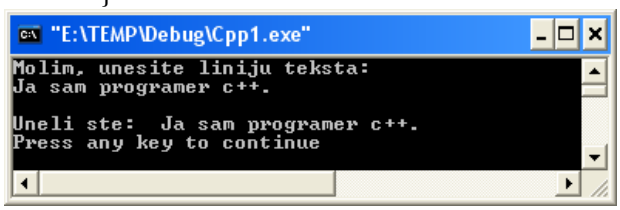
to znači

```
#include <iostream>
using namespace std;
```

Dakle da bi koristili deklaraciju promenljive strings uključićemo hedersku datoteku <string>. Klasa string je definisana sa namespace std tako da je korišćenje ove direktive neophodno.

2 Zadatak:

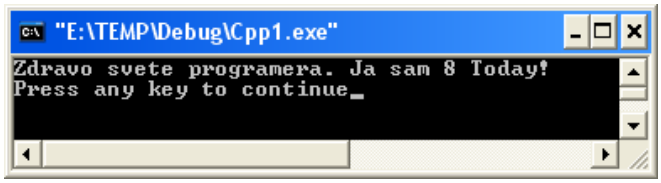
```
#include<iostream>
#include<string>
using namespace std;
        int main()
{       cout << "Molim, unesite liniju teksta:\n";
        string s;
        getline(cin,s);
        cout << "Uneli ste:  " << s << '\n';
        return 0;
}
```



3 Zadatak:

Prva dva stringa, s1 i s2, startuju prazni, dok s3 i s4 prikazuju dva ekvivalentna nacina da inicijalizuju objekat stringa iz polja karaktera.

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string s1, s2; // Empty strings
string s3 = "Zdravo svete programera."; // Initialized
string s4("Ja sam"); // Also initialized
s2 = "Today"; // Assigning to a string
s1 = s3 + " " + s4; // Combining strings
s1 += " 8 "; // Appending to a string
cout << s1 + s2 + "!" << endl;
}
```



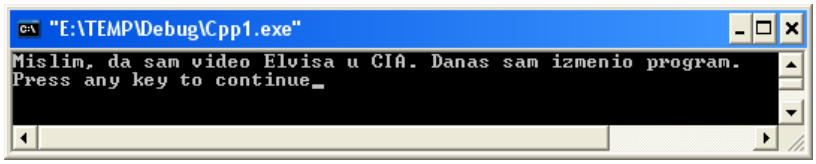
4 Zadatak:

Primer zamene karaktera u stringu - complete demonstration replace( )

```
#include <string>
#include <iostream>
using namespace std;
void replaceChars(string& modifyMe,string findMe, string newChars)
{
// Look in modifyMe for the "find string"
// starting at position 0
int i = modifyMe.find(findMe, 0);
// Did we find the string to replace?
if(i != string::npos)
// Replace the find string with newChars
modifyMe.replace(i,newChars.size(),newChars);
}

int main()
{
string bigNews =
"Mislim, da sam video Elvisa u UFO. "
"Danas sam izmenio program.";
string replacement("CIA");
string findMe("UFO");
// Find "UFO" in bigNews and overwrite it:
```

```
replaceChars(bigNews, findMe, replacement);
cout << bigNews << endl;
}
```

IZLAZ:

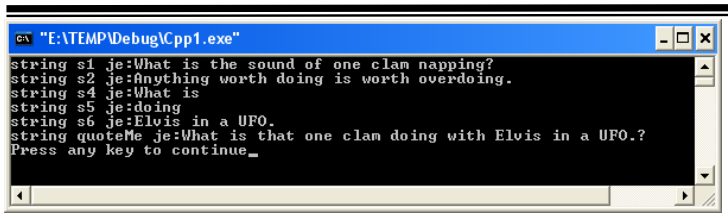


5 Zadatak:

Najprostija forma inicijalizacije stringa, koja nudi varijacije koje nude fleksibilnost i bolju kontrolu.. Mozemo :

- » Koristiti ma koji deo C char polja ili C++ stringa
- » Kombinovati razlicite delove inicijalizacije podataka operatorom +
- » Koristiti string objekta substr( ) funkciju clanicu za kreiranjea substringa

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string s1
("What is the sound of one clam napping?");
cout<<"string s1 je:"<<s1<<endl;
string s2
("Anything worth doing is worth overdoing.");
string s3("I saw Elvis in a UFO.");
cout<<"string s2 je:"<<s2<<endl;
// Copy the first 8 chars
string s4(s1, 0, 8);
cout<<"string s4 je:"<<s4<<endl;
// Copy 6 chars from the middle of the source
string s5(s2, 15, 6);
cout<<"string s5 je:"<<s5<<endl;
// Copy from middle to end
string s6(s3, 6, 15);
cout<<"string s6 je:"<<s6<<endl;
// Copy all sorts of stuff
string quoteMe = s4 + "that" +
// substr() copies 10 chars at element 20
s1.substr(20, 10) + s5 +
// substr() copies up to either 100 char
// or eos starting at element 5
"with" + s3.substr(5, 100) +
// OK to copy a single char this way
s1.substr(37, 1);
cout<<"string quoteMe je:"<<quoteMe<<endl;
}
```

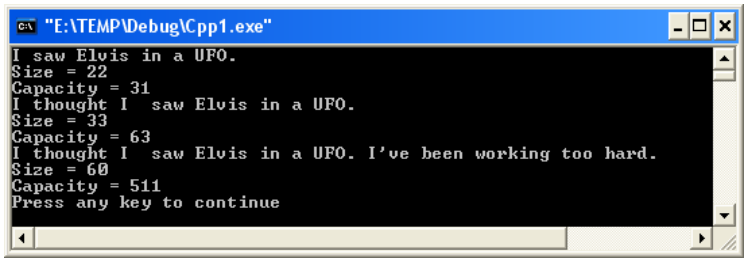


6 Zadatak:

Slično kao predhodni primer: C++ za string obezbedjuje nekoliko alata za monitoring i upravljanje njegove veličine. Tako size( ), resize( ),capacity( ), i reserve( ) funkcije članice mogu biti vrlo korisne za rad sa ovakvim podacima.

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string bigNews("I saw Elvis in a UFO. ");
cout << bigNews << endl;
// How much data have we actually got?
cout << "Size = " << bigNews.size() << endl;
// How much can we store without reallocating
cout << "Capacity = "<< bigNews.capacity() << endl;
// Insert this string in bigNews immediately
// following bigNews[1]
bigNews.insert(1, " thought I ");
cout << bigNews << endl;
cout << "Size = " << bigNews.size() << endl;
cout << "Capacity = "<< bigNews.capacity() << endl;
// Make sure that there will be this much space
bigNews.reserve(500);
// Add this to the end of the string
bigNews.append("I've been working too hard.");
cout << bigNews << endl;
cout << "Size = " << bigNews.size() << endl;
cout << "Capacity = "<< bigNews.capacity() << endl;
}
```

IZLAZ:

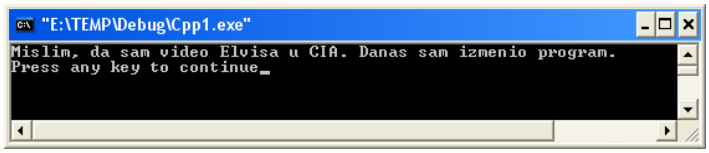


7 Zadatak:

Ovom prilikom proveravamo da li nesto postoji, pre nego izvrsimo replace( ).

```
#include <string>
#include <iostream>
using namespace std;
void replaceChars(string& modifyMe,string findMe, string newChars)
{// Look in modifyMe for the "find string"
// starting at position 0
int i = modifyMe.find(findMe, 0);
// Did we find the string to replace?
if(i != string::npos)
// Replace the find string with newChars
modifyMe.replace(i,newChars.size(),newChars);}

int main()
{
string bigNews =
"Mislim, da sam video Elvisa u UFO. "
"Danas sam izmenio program.";
string replacement("CIA");
string findMe("UFO");
// Find "UFO" in bigNews and overwrite it:
replaceChars(bigNews, findMe, replacement);
cout << bigNews << endl;}
```

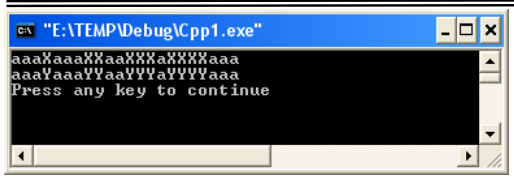
IZLAZ:



8 Zadatak:

Jednostavna zamena karaktera koriscenjem STL replace( ) algoritma

```
#include <string>
#include <algorithm>
#include <iostream>
using namespace std;
int main() {
string s("aaaXaaaXXaaXXXaXXXXaaa");
cout << s << endl;
replace(s.begin(), s.end(), 'X', 'Y');
cout << s << endl;
}
```

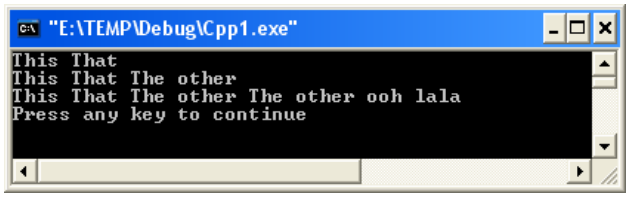


9 Zadatak:

Koriscenje operatora + i += za sabiranje stringova.

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string s1("This ");
string s2("That ");
string s3("The other ");
// operator+ concatenates strings
s1 = s1 + s2;
cout << s1 << endl;
// Another way to concatenates strings
s1 += s3;
cout << s1 << endl;
// You can index the string on the right
s1 += s3 + s3[4] + "oh lala";
cout << s1 << endl;
}
```

IZLAZ:



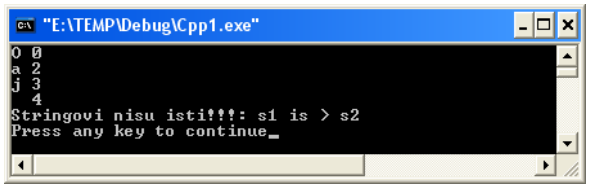
10 Zadatak:

C++ obezbedjuje nekoliko nacina za uporedjenje stringova. Najjednostavnije je koristiti funkcijeske operatore ==, operator != operator >, operator <, operator >=, i operator <=.

```
#include <string>
#include <iostream>
using namespace std;
int main() {
// Strings to compare
string s1("Ovaj ");
string s2("Onaj ");
```

```
for(int i = 0; i< s1.size() && i < s2.size(); i++)
// See if the string elements are the same:
if(s1[i] == s2[i])
cout << s1[i] << " " << i << endl;
// Use the string inequality operators
if(s1 != s2)
        {
cout << "Stringovi nisu isti!!!:" << " ";
if(s1 > s2)
cout << "s1 is > s2" << endl;
else
cout << "s2 is > s1" << endl;
        }
}
```

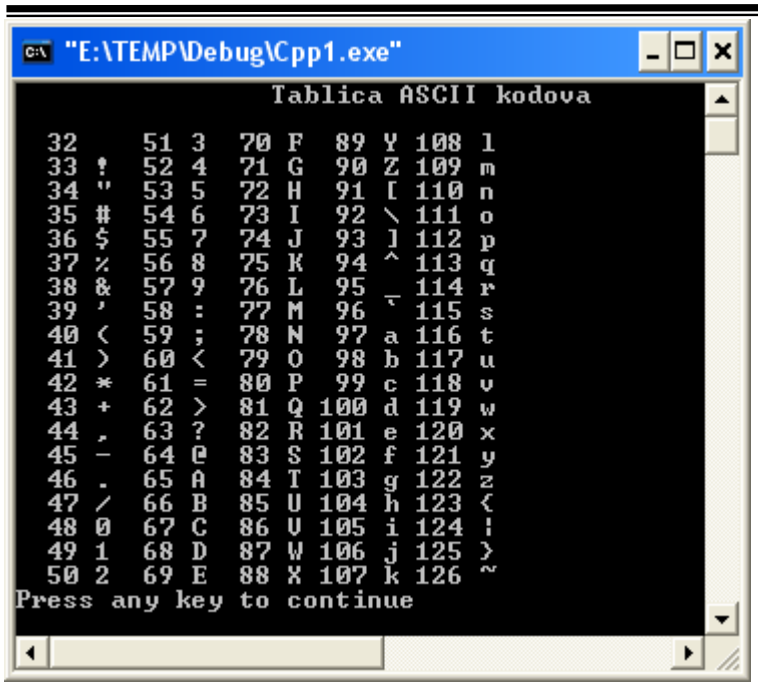
IZLAZ:



11 Zadatak:

Program za prikazivanje tablice ASCII kodova:

```
#include <iostream.h>
#include <iomanip.h>
main()
{
        char c=' ';
        int i;

        cout<<"\t\tTablica ASCII kodova \n \n";
linija:
        i=0;
znak:
        cout<<setw(4)<<(int)(c+i)<<" "<<setw(1)<<(char)(c+i);
        i=i+19;
        if (i<95) goto znak;
        cout<<"\n";
        c=c+1;
        if (c<' '+19) goto linija;
}
```

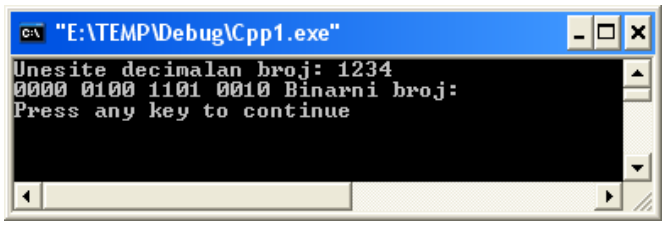


12 Zadatak:

Sastaviti program na programskom jeziku C++ koji učitava decimalan pozitivan celi broj u obliku niza znakova i ispisuje njegovu vrednost u binarnom obliku. Pretpostaviti da se za interno predstavljanje celih brojeva koristi 16 bitova.

```
#include <iostream.h>
#include <iomanip.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
main()
{
        char dec[10];
        short int bin,i;
        cout<<"Unesite decimalan broj: ";
cin>>dec;       /* ucitava string sa standardnog ulaza (dec=&dec[0]) */
                /*      atoi vraca 0 ukoliko nije uspela konverzija
                        Ukoliko je strlen(dec)=0 onda se drugi deo USLOVA (iza &&
        operatora)              nece ni proveravati. Po postavci zadatka ocekuje se
        pozitivan broj,  i takav uslov se jednostavno, ovim putem, moze proveriti.      */
if(strlen(dec) && (bin=atoi(dec)))
        {
                cout<<"Binarni broj: ";
                i=-1;
                while (++i<16)
                {
                        putchar((bin & 0x8000) ? '1' : '0');
```

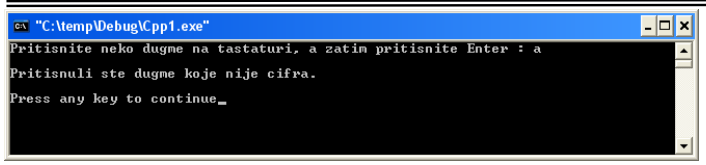
```
                    /*      0x8000 ima jedinicu na najvisem 15-om bitu
                            Gornja naredba ce ispisati 15-i bit broja bin!
                            Najpre ispisujemo najvise bitove, jer takav prikaz zelimo na
                            ekranu bit15 bit14 ... bit2 bit1 bit0   */
                    bin <<= 1 ;
/* bin = bin shl 1 ; pomeramo bin ulevo za 1 bit */
                            if (i%4 == 3)
                            putchar(' ');
/* prikaz razmaka izmedju svake polovine bajta */
                    }
          cout<<"\n";
          }
          else
                    cout<<"Neispravan broj ili nula\n";
}
```



13 Zadatak:

Napisati program koji odlučuje da li je pritisnuto dugme cifra.

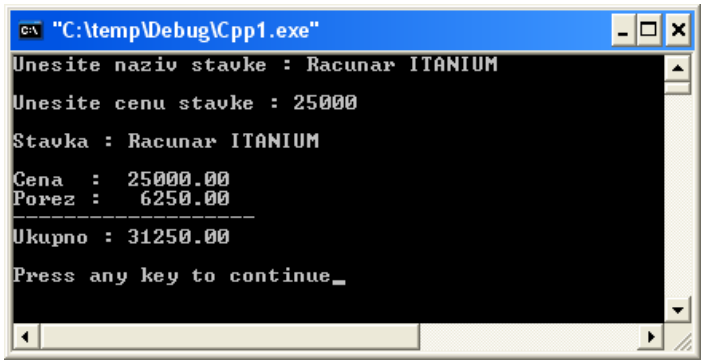
```
// Program prikazuje koriscenje slozene if naredbe
// koja odlucuje da li je pritisnutuo dugme broj.
// Program koristi cin.get() za unos znaka.
#include <iostream.h>

int main()
{
// Deklarisanje promenljivih
char znak;
// Unos znaka sa tastature
cout << "Pritisnite neko dugme na tastaturi, a zatim pritisnite "
<< "Enter : ";
znak = cin.get();
cout << endl;
// Provera unetog znaka i ispisivanje rezultata
if ( (znak >= '0') && (znak <= '9'))
cout << "Pritisnuli ste dugme koje je cifra." << endl;
else
cout << "Pritisnuli ste dugme koje nije cifra." << endl
<< endl;
return 0;
}
```



14 Zadatak:

Napisati program koji za unos niza znakova koristeći naredbu cin.getline().

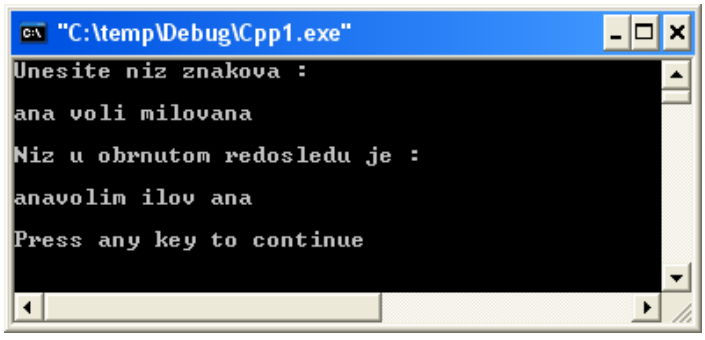
```
// program demonstrira koriscenje naredbe cin.getline()
// za unos stringa
#include <iostream.h>
#include <iomanip.h>
int main()
{
double const STOPA_POREZA = 0.25;
char naziv_stavke[51];
double cena,
porez,
ukupno;
cout << setprecision(2)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
cout << "Unesite naziv stavke : ";
cin.getline(naziv_stavke, 51);
cout << endl;
cout << "Unesite cenu stavke : ";
cin >> cena;
porez = cena * STOPA_POREZA;
ukupno = cena + porez;
cout << endl;
cout << "Stavka : " << naziv_stavke << endl << endl;;
cout << "Cena  : " << setw(9) << cena << endl;
cout << "Porez : " << setw(9) << porez << endl;
cout << "--------------------" << endl;
cout << "Ukupno :" << setw(9) << ukupno << endl << endl;
return 0;
}
```



15 Zadatak:

Napisati program koji zahteva unos znakova u jednom redu korišćenjem klase Cstring i pokazivača. Program nakon toga ispisuje uneti niz u obrnutom redosledu od redosleda unosa.

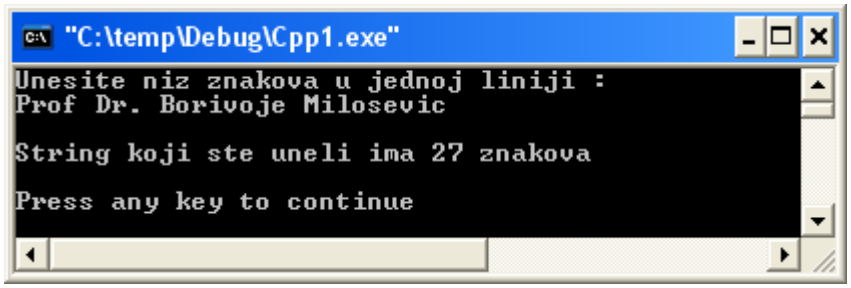
```
// Ovaj program zahteva unos stringa
// a zatim ga ispisuje i obrnutom redosledu.
#include <iostream.h>
int main()
{
char linija[81];
char* pokazivac_na_znak = linija;
cout << "Unesite niz znakova :" << endl << endl;
cin.getline(linija, 81);
// Pronalazenje kraja stringa
while ( *pokazivac_na_znak != '\0' )
++ pokazivac_na_znak;
// ch_ptr sada pokazuje na null
-- pokazivac_na_znak;
// pokazivac_na_znak sada pokazuje na poslednji znak u stringu
cout << endl;
cout << "Niz u obrnutom redosledu je :" << endl << endl;
// while petlja ispisuje sve znakove osim prvog
while (pokazivac_na_znak != linija )
{
cout << *pokazivac_na_znak;
--pokazivac_na_znak;
}
// Ispisivanje prvog znaka
cout << *pokazivac_na_znak;
cout << endl << endl;
return 0;
}
```



16 Zadatak:

Napisati program koji izračunava broj unetih znakova u jednoj liniji. Koristiti funkciju kojoj se string prosleđuje po adresi.

```
// Program izracunava broj unetih znakova u jednoj liniji
// Koristi funkciju Duzina_Niza() za brojanje unetih znakova
#include <iostream.h>
int Duzina_Niza(char*);
int main()
{
char linija[81];
cout << "Unesite niz znakova u jednoj liniji : " << endl;
cin.getline(linija,81);
cout << endl;
cout << "String koji ste uneli ima " << Duzina_Niza(linija)
<< " znakova" << endl << endl;
return 0;
}
int Duzina_Niza(char* pokazivac_na_znak)
{
int duzina_niza = 0;
while (*pokazivac_na_znak != '\0')
{
++duzina_niza;
++pokazivac_na_znak;
}
return duzina_niza;
}
```

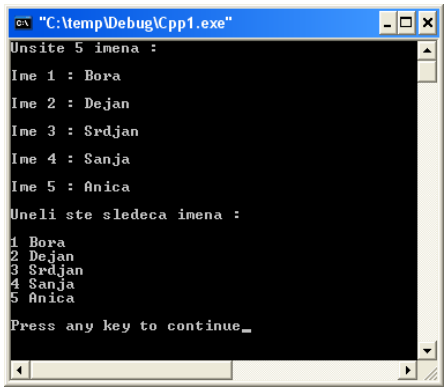


17 Zadatak:

Napisati program koji unetom nizu dinamički dodeljuje memoriju.

```
// Program ilustruje upotrebu naredbi new i delete
// za dinamicku dodelu memorije nizu.
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
int main()
```

```
{
const int BROJ_OSOBA = 5;
char buffer[81];
char* osoba[BROJ_OSOBA];
int i;
cout << "Unsite " << BROJ_OSOBA << " imena :" << endl;
for (i = 0; i < BROJ_OSOBA; ++i)
{
cout << endl;
cout << "Ime " << i + 1 << " : ";
cin.getline(buffer, 81);
osoba[i] = new char [strlen(buffer) + 1];
strcpy(osoba[i], buffer);
}
cout << endl;
cout << "Uneli ste sledeca imena :" << endl;
for (i = 0; i < BROJ_OSOBA; ++i)
cout << endl << i+1 << " " << osoba[i];
for (i = 0; i < BROJ_OSOBA; ++i)
delete [] osoba[i];
cout << endl << endl;
return 0;
}
```



Prilikom statičog dodeljivanja memorije, memorijski prostor se rezerviše za sve moguće članove niza, bez obzira da li će rezervisani prostor biti upotrebljen ili ne. Za razliku od statičke dodele memorije, dinamička dodela memorije nizu može znatno da uštedi memorijski prostor i ubrza rad aplikacije. Dinamičkom dodelom, u memoriju se upisuju samo postojeći elementi niza.

F U N K C I J E

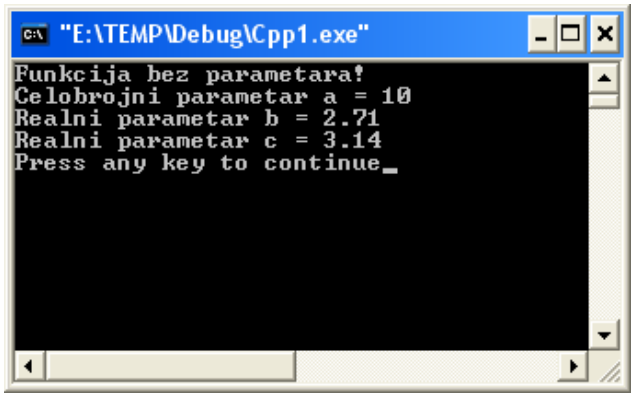
Ima mnogo razloga za upotrebu funkcija:

a.) Deo koda može biti iskorišćen mnogo puta u različitim delovima programa.
b.) Program se deli na niz blokova. Svaki blok će raditi specijalni posao. Razumevanje i dizajn programa biće na ovaj način pojednostavljeno.
c.) Blok koda se može izvršavati sa različitim brojem inicijalnih parametara. Ti parametri su postavljeni u funkciji sa argumentima.
d.) C++ omogucuje da više funkcija ima isti naziv, pod uslovom da su im liste parametara različite.
Prema listi parametara kod poziva funkcije odreduje se koja će funkcija biti pozvana.

1 Zadatak:

Kako radi funkcija bez parametara?

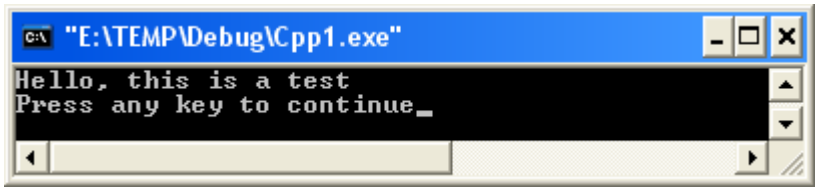
```
#include <iostream.h>
void funkcija()
{cout << "Funkcija bez parametara!" << endl;
}
void funkcija(int a)
{cout << "Celobrojni parametar a = " << a << endl;
}
void funkcija(float b, float c)
{cout << "Realni parametar b = " << b << endl;
cout << "Realni parametar c = " << c << endl;
}

void main()
{
funkcija();
funkcija(10);
funkcija(2.71,3.14);
}
```



2 Zadatak:

Pokazati kako funkcioe prototip funkcije.

```
#include<iostream.h>
myfunc();/*inicijalizacija funkcije bez argumenata - prototip*/
main()
{
  myfunc();/*pozivanje funkcije bez argumenata*/
}
                        myfunc()/*definisanje funkcije bez argumenata*/
{
  cout<<"Hello, this is a test\n";
}
```



3 Zadatak:

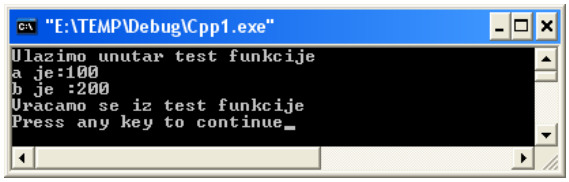
Korišćenje prototipa funkcije sa argumentom.

```
#include <iostream.h>
sayhello(int count);
main()
{sayhello(4);}
sayhello(int count)
{int c;
for(c=0;c<count;c++)
  cout<<"\nHello";}
```

4 Zadatak:

Test funkcije

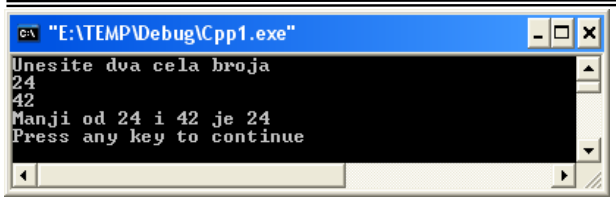
```
#include<iostream.h>
void test();
int main()
{
cout<<"Ulazimo unutar test funkcije\n";
test();
cout<<"Vracamo se iz test funkcije\n";
return 0;
}

void test()
{int a,b;a=100;
b=a+100;
cout<<"a je:"<<a<<"\nb je :"<<b<<"\n";}
```



5 Zadatak:

Naći minimum od dva cela broja. U glavnom programu obezbediti štampanje, a u funkciji njihovo poređenje.

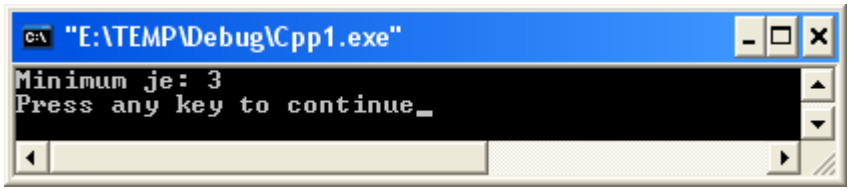
```
#include <iostream.h>
int imin(int n,int m);
int main()/* glavni program */
{
        int broj1,broj2;
        cout<<"Unesite dva cela broja\n";
        cin>>broj1>>broj2;
        cout<<"Manji od "<< broj1<<" i "<< broj2<<" je "<< imin(broj1,broj2)<<"\n";
        return 0;}

int imin(int n,int m)/* potprogram */
{
        int min;
        if (n<m)
                min=n;
        else
                min=m;
        return min;}
```



6 Zadatak:

Isti kao 4. samo na drugi način.

```
#include <iostream.h>
int min(int a,int b);

main()
{
int m;
m=min(3,6);
cout<<"Minimum je: "<<m<<"\n";
return 0;
}

int min(int a, int b)
{
if(a<b)
 return a;
else
 return b;}
```

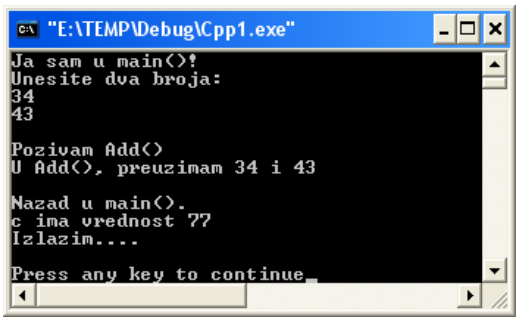


7 Zadatak:

Formiranje zbira preko funkcije

```
#include <iostream.h>
int Add (int x, int y);
int main()
{
cout << "Ja sam u main()!\n";
int a, b, c;
cout << "Unesite dva broja:\n";
cin >> a;
cin >> b;
cout << "\nPozivam Add()\n";
```

```
c=Add(a,b);
cout << "\nNazad u main().\n";
cout << "c ima vrednost " << c;
cout << "\nIzlazim....\n\n";
return 0;
}

int Add (int x, int y)
{
cout << "U Add(), preuzimam " << x << " i " << y << "\n";
return (x+y);
}
```

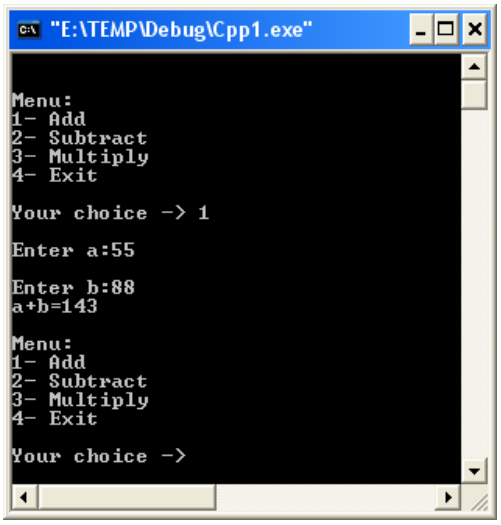


8 Zadatak:

Uraditi meni sa izborom operacija za sabiranje, oduzimanje i množenje dva cela realna broja. U funkciji izvesti opisane računske radnje.

```
#include <iostream.h>
#include <stdlib.h>
add();
subtract();
multiply();
main()
{
int choice;
while(1)
 {
 cout<<"\n\nMenu:\n";
 cout<<"1- Add\n2- Subtract\n";
 cout<<"3- Multiply\n4- Exit";
 cout<<"\n\nYour choice -> ";
 cin>>choice;
            switch(choice)
                    {    case 1 : add();
          break;
                        case 2 : subtract();
          break;
                        case 3 : multiply();
```

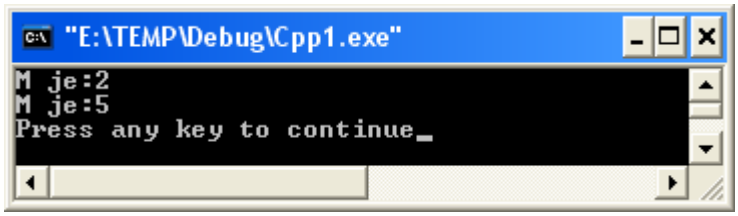
```
        break;
                case 4 : cout<<"\nProgram Ends. !";
        exit(0);

default:
        cout<<"\nInvalid choice";     }  }}

add()
{float a,b;
cout<<"\nEnter a:";
cin>>a;
cout<<"\nEnter b:";
cin>>b;
cout<<"a+b="<<a+b;}

subtract()
{float a,b;
cout<<"\nEnter a:";
cin>>a;
cout<<"\nEnter b:";
cin>>b;
cout<<"a-b="<<a-b;}

multiply()
{float a,b;
cout<<"\nEnter a:";
cin>>a;
cout<<"\nEnter b:";
cin>>b;
cout<<"a*b="<<a*b;}
```



8 Zadatak:

Pozivanje funkcije po vrednosti

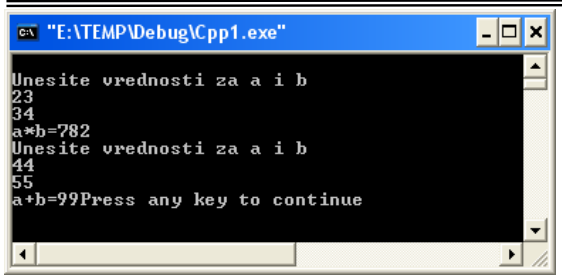
```
/*Pozivanje po vrednosti*/
#include <iostream.h>
int test(int a);
int main()
{int m,y;
m=2;
cout<<"M je:"<<m;
y=test(m);
cout<<"\nM je:"<<y<<endl;
return 0;}
int test(int a)
{a=5;
return a ;}
```



9 Zadatak:

MNOŽENJE I SABIRANJE DVA REALNA BROJA PREKO FUNKCIJE

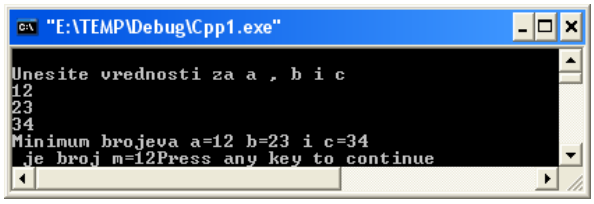
```
#include <iostream.h>
#include <math.h>
mno();
sab();
main()
{float a,b;
mno(); sab();}
mno()
{float a,b;
cout<<"\nUnesite vrednosti za a i b\n";
cin>>a>>b;
cout<<"a*b="<<a*b;}
sab()
{float a,b;
cout<<"\nUnesite vrednosti za a i b\n";
cin>>a>>b;
cout<<"a+b="<<a+b;}
```



10 Zadatak:

MINIMUM TRI CELA BROJA PREKO FUNKCIJE

```
#include <iostream.h>
#include <math.h>
min();
main()
{int a,b,c,m;
min();}
min()
{int a,b,c,m;
cout<<"\nUnesite vrednosti za a , b i c\n";
cin>>a>>b>>c;
m=a;
if(b<m)  m=b;
if(c<m)  m=c;
cout<<"Minimum brojeva a="<<a<<" b="<<b<<" i c="<<c<<"\n"
<<" je broj m="<<m;}
```

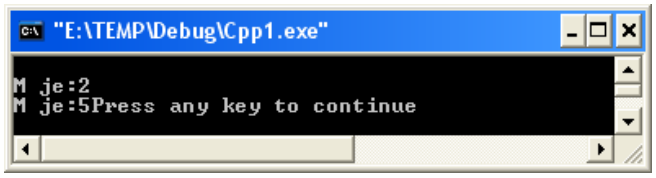


11 Zadatak:

Poziv funkcije salje vrednost promenljive m funkciji a ne salje promenljivu samu sebi: m=2, M=2

```
/*Pozivanje po referenci*/
#include <iostream.h>
void test(int *ptr);
main()
{
int m;
m=2;
cout<<"\nM je:"<<m;
```

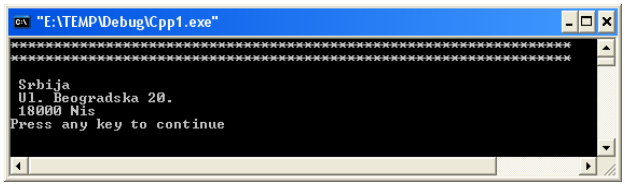
```
test(&m);
cout<<"\nM je:"<<m;
return 0;
}
void test(int *ptr)
{
*ptr=5;
}
/*Poziv funkcije salje vrednost promenljive m po referenci: m=2, M=5*/
```



12 Zadatak:

Poziv funkcije bez argumenta

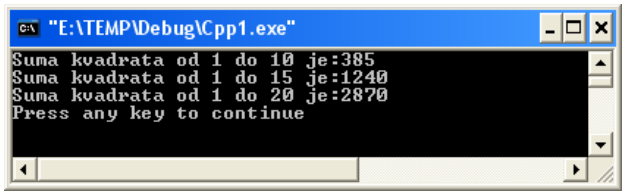
```
#include <iostream.h>
#include <stdio.h>
#define IME "Srbija"
#define ADRESA "Ul. Beogradska 20."
#define MESTO "18000 Nis"
#define LIMIT 65
void main()
{void zvezde(void);/*deklaracija funkcije bez argumenata*/
zvezde(); /*poziv korisnicke funkcije */
cout<<"\n "<<IME ; /*poziv funkcije iz standardne biblioteke*/
cout<<"\n "<<ADRESA;
cout<<"\n "<<MESTO<<"\n";
zvezde() ;} /*definicija korisnicke funkcije */
void zvezde() /*funkcija nema argumenata*/
{ int brojac;
for(brojac=1;brojac<=LIMIT;brojac++)
putchar ('*');
putchar ('\n') ; }
```



13 Zadatak:

Suma kvadrata preko funkcije

```
/*Program za izracunavanje sume
kvadrata celih brojeva od 1 do n*/
#include <iostream.h>
suma_kvadrata(int p);
main()
        {suma_kvadrata(10);
        suma_kvadrata(15);
        suma_kvadrata(20);
    return 0;}
suma_kvadrata(n)                        /*f ja za izracunavanje*/
        int n;                          /*sume kvadrata*/
        {int i;
        long suma=0;
        for (i=1; i<=n; suma+=(long)i*i, ++i);
        cout<<"Suma kvadrata od 1 do "<<n<<" je:"<<suma<<"\n";}
```

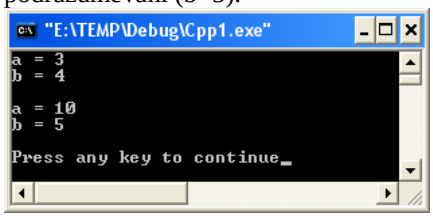


14 Zadatak:

Prilikom poziva funkcije potrebno je navesti listu argumenata koja odgovara listi argumenata u zaglavlju funkcije. C++ dopušta da se neki od argumenata u pozivu funkcije izostave, tako da se umesto njih koriste podrazumevane vrednosti.

```
#include <iostream.h>
void funkcija(int a, int b=5){
cout << "a = " << a << endl;
cout << "b = " << b << endl << endl;}

void main(){
funkcija (3,4);
funkcija (10);}
```

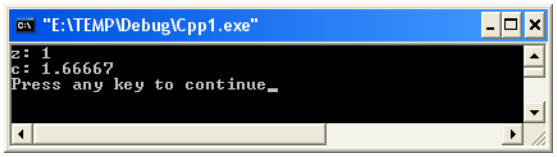
U drugom pozivu funkcije izostavljen je drugi argument, umesto kojeg se koristi podrazumevani (b=5).



15 Zadatak:

Preko dve funkcije prikazati operaciju deljenja i njen rezultat kada se u glavni program poziva kao INT tip i FLOAT tip.

```
#include <iostream.h>

 void intDiv(int x, int y)
 {
      int z = x / y;
      cout << "z: " << z << endl;
 }

 void floatDiv(int x, int y)
{
    float a = (float)x;                  // stari stil
    float b = static_cast<float>(y);    // napredni stil
    float c = a / b;

    cout << "c: " << c << endl;
}

 int main()
 {
     int x = 5, y = 3;
     intDiv(x,y);
     floatDiv(x,y);
     return 0;
 }
```

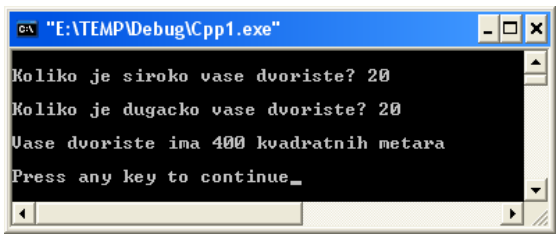


16 Zadatak:

Sastaviti program za izračunavanje površine dvorišta, kada se sirina i dužina dvorišta unose u glavnomprogramu a površina izračunava u funkciji.

```
#include <iostream.h>
   int Povrsina(int length, int width); //function prototype

   int main()
   {
     int lengthOfYard;
    int widthOfYard;
    int areaOfYard;
```

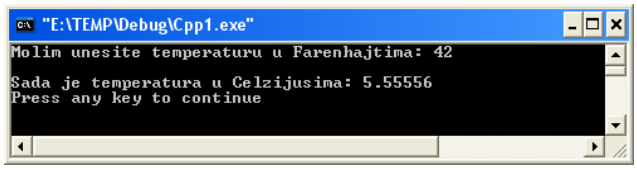
```
   cout << "\nKoliko je siroko vase dvoriste? ";
   cin >> widthOfYard;
   cout << "\nKoliko je dugacko vase dvoriste? ";
   cin >> lengthOfYard;

   areaOfYard= Povrsina (lengthOfYard,widthOfYard);

   cout << "\nVase dvoriste ima ";
   cout << areaOfYard;
   cout << " kvadratnih metara\n\n";
          return 0;
 }

 int Povrsina (int yardLength, int yardWidth)
 {
    return yardLength * yardWidth;
 }
```



17 Zadatak:

Sastaviti program za konverziju stepeni Farenhajta u Celzijus. Konverziju organizovati u funkciji.

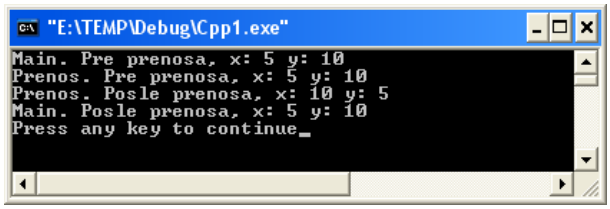
```
#include <iostream.h>

    float Konverzija(float);
    int main()
    {
      float TempFer;//Temperatura u Farenhajtima
      float TempCel;//Temperatura u Stepenima

       cout << "Molim unesite temperaturu u Farenhajtima: ";
      cin >> TempFer;
      TempCel = Konverzija(TempFer);
      cout << "\nSada je temperatura u Celzijusima: ";
      cout << TempCel << endl;
             return 0;
    }
    float Konverzija(float TempFer)
    {
      float TempCel;
      TempCel = ((TempFer - 32) * 5) / 9;
      return TempCel;    }
```



18 Zadatak:

Demonstracija prenosa promenljivih po vrednosti.

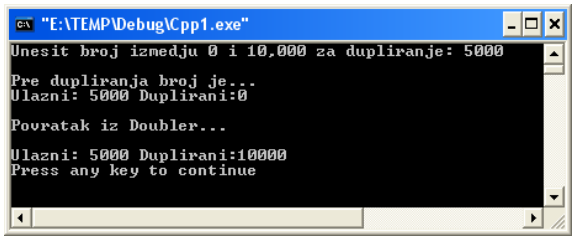
```
#include <iostream.h>

    void prenos(int x, int y);//Funkcijski prototip

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

cout << "Main. Pre prenosa, x: " << x << " y: " << y << "\n";
      prenos(x,y);//Poziv funkcije za prenos vrednosti
cout << "Main. Posle prenosa, x: " << x << " y: " << y << "\n";
        return 0;
    }

    void prenos (int x, int y)//Funkcija
    {
      int temp;

cout << "Prenos. Pre prenosa, x: " << x << " y: " << y << "\n";

      temp = x;
      x = y;
      y = temp;

cout << "Prenos. Posle prenosa, x: " << x << " y: " << y << "\n";

    }
```



19 Zadatak:

Demonstraciju višeznačne naredbe return izvesti programom za dupliranje vrednosti promenljive.

```
#include <iostream.h>
    int Doubler(int IznosZaDupliranje);
    int main()
    {   int rezultat = 0;
        int input;
  cout << "Unesit broj izmedju 0 i 10,000 za dupliranje: ";
  cin >> input;
  cout << "\nPre dupliranja broj je... ";
cout << "\nUlazni: " << input << " Duplirani:" << rezultat << "\n";
        rezultat = Doubler(input);
        cout << "\nPovratak iz Doubler...\n";
cout << "\nUlazni: " << input << " Duplirani:" <<rezultat<< "\n";
        return 0; }

   int Doubler(int original)
   {
        if (original <= 10000)
            return original * 2;
        else
                        return -1;
        cout << "Ovo stampanje je nemoguce!\n";
   }
```

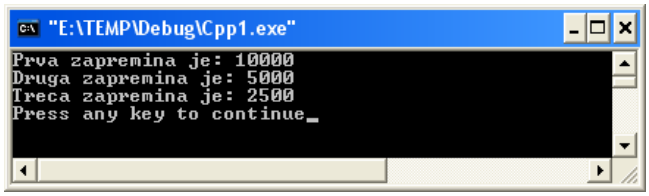


20 Zadatak:

Demonstracija korišćenja parametara po difoltu kada se rešava zapremina kocke u funkciji.

```
#include <iostream.h>

  int ZapreminaKocke(int length, int width = 25, int height = 1);
          /*Kada u programu nisu definisane vrednosti promenljivih se uzimaju po difoltu,
          onako kako su ovde specificirane*/
  int main()
  {
     int length = 100;
     int width = 50;
     int height = 2;
     int zapremina;
```

```
    zapremina = ZapreminaKocke(length, width, height);
    cout << "Prva zapremina je: " << zapremina << "\n";

    zapremina = ZapreminaKocke(length, width);
    cout << "Druga zapremina je: " << zapremina << "\n";

    zapremina = ZapreminaKocke(length);
    cout << "Treca zapremina je: " << zapremina << "\n";
     return 0;
}

ZapreminaKocke(int length, int width, int height)
{
   return (length * width * height);
 }
```

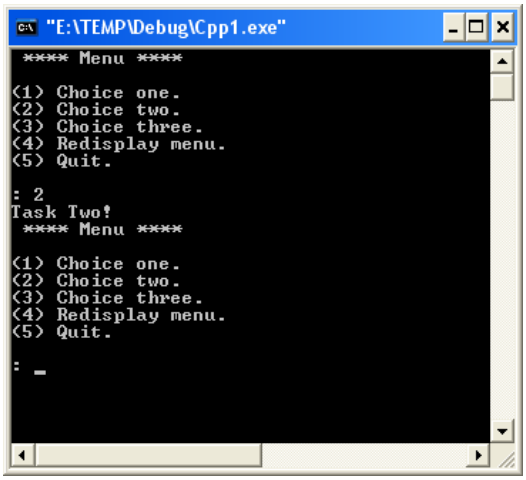


21 Zadatak:

Korišćenje beskonačne petlje za menadžment korisničke interakcije.

```
#include <iostream.h>
   // prototipovi
   int menu();
   void DoTaskOne();
   void DoTaskMany(int);
   int main()
   {   bool exit = false;
      for (;;)
      {    int choice = menu();
          switch(choice)
          {  case (1):
                DoTaskOne();
                break;
             case (2):
                DoTaskMany(2);
                break;
             case (3):
                DoTaskMany(3);
                break;
             case (4):
                continue;  // redundant!
                break;
```

```
            case (5):
                exit=true;
                break;
            default:
                cout << "Please select again!\n";
                break;
        }         // end switch

        if (exit)
            break;
    }             // end forever
  return 0;
}                 // end main()

int menu()
{int choice;
   cout << " **** Menu ****\n\n";
   cout << "(1) Choice one.\n";
   cout << "(2) Choice two.\n";
   cout << "(3) Choice three.\n";
   cout << "(4) Redisplay menu.\n";
   cout << "(5) Quit.\n\n";
   cout << ": ";
   cin >> choice;
   return choice;}

void DoTaskOne()
{       cout << "Task One!\n";    }

void DoTaskMany(int which)
{
   if (which == 2)
       cout << "Task Two!\n";
   else
       cout << "Task Three!\n";
}
```



22 Zadatak:

Prilikom poziva funkcije potrebno je navesti listu argumenata koja odgovara listi argumenata u zaglavlju funkcije. C++ dopušta da se neki od argumenata u pozivu funkcije izostave, tako da se umesto njih koriste podrazumevane vrednosti.

Primer:

```
#include <iostream.h>
void funkcija(int a, int b=5){
cout << "a = " << a << endl;
cout << "b = " << b << endl << endl;
}
void main(){
funkcija (3,4);
funkcija (10);}
```

23 Zadatak:

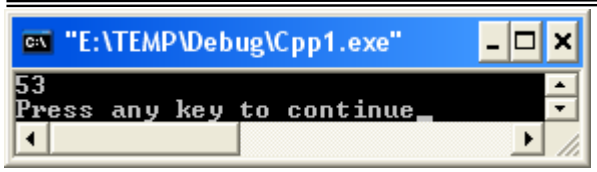
Deklaracije referenci
U C-u se često koriste pokazivači za prosledivanje argumenata funkcijama. U C++ može se koristiti referentni operator (&) u listi argumenata, što čini programski kod čišcim.

C:

```
void zamena (int *a, int *b){
int t = *a;
*a = *b;
*b = t;
}
void main()
{int x = 3, y = 5;
zamena (&x, &y);
printf ("%i %i\n",x,y);}
```

C++:

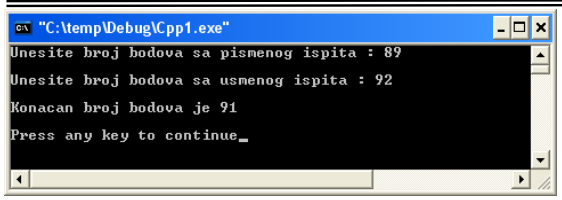
```
#include <iostream.h>
void zamena (int &a, int &b)
{int t = a;
a = b;
b = t;}
void main()
{int x = 3, y = 5;
zamena (x,y);
cout << x << y << endl;}
```



24 Zadatak:

Napisati program koji izračunava ocenu na ispitu na osnovu broja bodova sa usmenog i pismenog dela ispita. Koristiti funkciju definisanu izvan funkcije main().

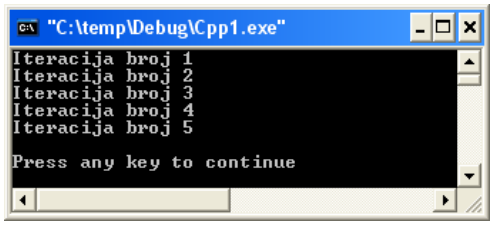
```
// Program izracunava krajnju ocenu na ispitu na osnovu rezultata
// pismenog i usmenog ispita. Koisti funkciju
// Izracunaj_Konacnu_Ocenu() za izracunavanje konacne ocene.
// Funkcija vraca vrednost u main().
#include <iostream.h>
int Izracunaj_Konacnan_Broj_Bodova(int, int); // Deklarisanje funkcije
int main()
{// Deklarisanje promenljivih
int bodovi_sa_pismenog_ispita,
bodovi_sa_usmenog_ispita,
konacan_broj_bodova;
// Unos podataka
cout << "Unesite broj bodova sa pismenog ispita : ";
cin >> bodovi_sa_pismenog_ispita;
cout << endl;
cout << "Unesite broj bodova sa usmenog ispita : ";
cin >> bodovi_sa_usmenog_ispita;
// izracunavanja pozivom funkcije
konacan_broj_bodova =
Izracunaj_Konacnan_Broj_Bodova(bodovi_sa_pismenog_ispita,
bodovi_sa_usmenog_ispita);
// Ispisivanje rezultata
cout << endl;
cout << "Konacan broj bodova je " << konacan_broj_bodova << endl;
cout << endl;
return 0;}
int Izracunaj_Konacnan_Broj_Bodova(int pismeni, int usmeni)
// Definisanje funkcije
{
// Deklarisanje promenljivih
const double UTICAJ_PISMENOG_ISPITA = 0.40;
const double UTICAJ_USMENOG_ISPITA = 0.60;
double bodovi;
int zaokruzeni_bodovi;
// Izracunavanja
bodovi = UTICAJ_PISMENOG_ISPITA * pismeni + UTICAJ_USMENOG_ISPITA
* usmeni;
zaokruzeni_bodovi = bodovi + 0.5;
return zaokruzeni_bodovi;}
```



25 Zadatak:

Napisati program koji broji iteracije u petlji primenom static promenljive.

```
#include <iostream.h>
void Brojac_Iteracija(); // Deklarisanje funkcije
int main()
{
// Deklarisanje promenljivih
int i;
// for petlja koja poziva funkciju Brojac_Iteracija()
// definisani broj puta
for (i = 1; i <= 5; ++i)
Brojac_Iteracija();

cout << endl;
return 0;
}
void Brojac_Iteracija() // Definicija funkcije
{
// Deklarisanje promenljivih
static int brojac = 0; // static oznacava da je memorijska
// lokacija u koju se smesta promenljiva brojac nepromenljiva
++brojac;
// Ispisivanje poruke
cout << "Iteracija broj " << brojac << endl;
return;
}
```



Trajanje promenljive je vreme za koje program alocira memoriju promenljivoj. Automatic variable je lokalna varijabla, koja po default-u ima auto storage class, automatski pozivom funkcije dodeljuje joj se memorija .
static variable se dodeljuje memorija kada program počne. Ostaje da važi sve vreme dok se program izvršava, nezavisno koja je funkcija aktivna. Globalna variabla ima static storage class po default-u.

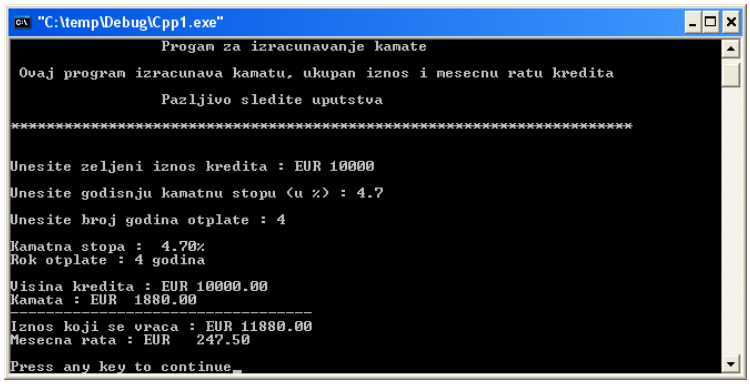
26 Zadatak:

Napisati program koji izračunava mesečnu ratu prilikom otplate kredita upotrebom funkcija.

```
// Program za izracunavanje kamate
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
void Ispisivanje_Pozdravne_Poruke();
double Unos_Visine_Kredita();
double Unos_Kamatne_Stope();
int Unos_Roka_Otplate();
int main()
{
double kamata,
kredit,
kamatna_stopa,
ukupno,
mesecna_rata;
int rok_otplate;
cout << setprecision(2)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
// Ispisivanje pozdravne poruke
Ispisivanje_Pozdravne_Poruke();
// Unos podataka
kredit = Unos_Visine_Kredita();
kamatna_stopa = Unos_Kamatne_Stope();
rok_otplate = Unos_Roka_Otplate();
// Izracunavanja
kamata = kredit * (kamatna_stopa/100) * rok_otplate;
ukupno = kredit + kamata;
mesecna_rata = ukupno / (12 * rok_otplate);
// Ispisivanje rezultata
cout << endl;
cout << "Kamatna stopa : " << setw(5) << kamatna_stopa
<< "%" << endl;
cout << "Rok otplate : " << rok_otplate << " godina"
<< endl << endl;
cout << "Visina kredita : EUR" << setw(9) << kredit << endl;
cout << "Kamata : EUR" << setw(9) << kamata << endl;
cout << "----------------------------------" << endl;
cout << "Iznos koji se vraca : EUR" << setw(9) << ukupno << endl;
cout << "Mesecna rata : EUR" << setw(9) << mesecna_rata
<< endl;
cout << endl;
return 0;
} // kraj funkcije main()
void Ispisivanje_Pozdravne_Poruke()
{
int broj_znakova;
```

```
cout << endl << endl;
for (broj_znakova = 1; broj_znakova <= 70; ++broj_znakova)
cout << "*";
cout << endl << endl;
cout << "\t\t Progam za izracunavanje kamate" << endl << endl;
cout << " Ovaj program izracunava kamatu, ukupan iznos i ";
cout << "mesecnu ratu kredita" << endl << endl;
cout << "\t\t Pazljivo sledite uputstva" << endl << endl;
for (broj_znakova = 1; broj_znakova <= 70; ++broj_znakova)
cout << "*";
cout << endl << endl << endl;
} // Kraj funkcije Ispisivanje_Pozdravne_Poruke()
double Unos_Visine_Kredita()
{
const double MIN_KREDIT = 1000.00;
const double MAX_KREDIT = 20000.00;
double kredit;
cout << "Unesite zeljeni iznos kredita : EUR ";
cin >> kredit;
if (kredit < MIN_KREDIT)
{
cout << endl;
cout << "Zao nam je, ne odobravamo kredite manje od 1,000.00 "
<< "EUR" << endl;
exit(1);
}
else
if (kredit > MAX_KREDIT)
{
cout << endl;
cout << "Zao nam je, ne odobravamo kredite";
cout << " vece od 20,000.00 EUR" << endl << endl;
cout << "Pokusajte sa drugom vrstom kredita\n";
exit(1);
}
else
return kredit;
} // Kraj funkcije Unos_Visine_Kredita()
double Unos_Kamatne_Stope()
{
const double MAX_KAMATNA_STOPA = 18.70;
double kamatna_stopa;
cout << endl;
cout << "Unesite godisnju kamatnu stopu (u %) : ";
cin >> kamatna_stopa;
if (kamatna_stopa > MAX_KAMATNA_STOPA)
{
cout << endl;
cout << "Zao nam je, kamatna stopa prevazilazi zakonski "
<< "maksimum od " << MAX_KAMATNA_STOPA << "%" << endl;
exit(1);
}
else
```

```
return kamatna_stopa;
} // Kraj funkcije Unos_Kamatne_Stope()
int Unos_Roka_Otplate()
{
const int MAX_ROK = 5;
int rok;
cout << endl;
cout <<"Unesite broj godina otplate : ";
cin >> rok;
if (rok > MAX_ROK)
{
cout << endl;
cout << "Zao nam je, rok otplate je veci od"
<< MAX_ROK << "godina" << endl;
exit(1);
}
else
return rok;
}
```

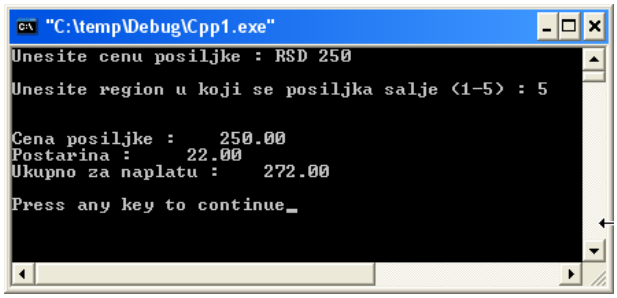


27 Zadatak:

Napisati program koji računa iznos koji se plaća prilikom preuzimanja poštanske pošiljke. Koristiti jednodimenzioni niz.

```
// Program racuna iznos koji je potrebno platiti prilikom prijema posiljke.
// U cenu ulazi cena posiljke kao i postarina, koja zavisi od regiona
// u kome se isporucuje posiljka.
// U programu se demonstrira upotreba promenljive tipa niz - array.
#include <iostream.h>
#include <iomanip.h>
int Unesi_Korektan_Region();
int main()
{
// Sledeca deklaracija definise niz celih brojeva stopa[]
// koji ima 5 elemenata.
double stopa[5] = {0.075, 0.080, 0.082, 0.085, 0.088};
```

```
int region;
double cena_posiljke,
postarina,
ukupno_za_naplatu;
cout << setprecision(2)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
cout << "Unesite cenu posiljke : RSD ";
cin >> cena_posiljke;
region = Unesi_Korektan_Region();
postarina = cena_posiljke * stopa[region - 1];
ukupno_za_naplatu = cena_posiljke + postarina;
cout << endl << endl;
cout << "Cena posiljke : " << setw(9) << cena_posiljke
<< endl;
cout << "Postarina : " << setw(9) << postarina << endl;
cout << "Ukupno za naplatu : " << setw(9) << ukupno_za_naplatu
<< endl << endl;
return 0;
} //Kraj funkcije main()
int Unesi_Korektan_Region()
{
bool pogresan_unos;
int region;
do
{
cout << endl;
cout << "Unesite region u koji se posiljka salje (1-5) : ";
cin >> region;
if (region < 1 || region > 5)
{
cerr << endl;
cerr << "Pogresan unos regiona - Pokusajte ponovo.";
pogresan_unos = true;
}
else
pogresan_unos = false;
}
while(pogresan_unos);
return region;
}
```



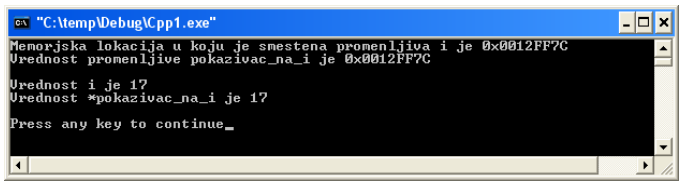
POINTERI

Pokazivač je promenljiva čija je vrednost adresa druge promenljive.
Operator indirekcije * je unarni operator. Sledi, ima isti prioritet i (sa desna-u-levo) asocijativnost, kao i drugi unarni operatori.
Čita se kao "the target of." Cilj pokazivača je promenljiva na koju on pokazuje.

1 Zadatak:

Napisati program koji dodeljuje vrednost promenljivoj primenom pokazivača.

```
// Program ilustruje upotrebu pokazivaca - pointera
// i indirektnog operatora.
#include <iostream.h>
int main()
{
int i;
int* pokazivac_na_i = &i;
cout << "Memorjska lokacija u koju je smestena promenljiva i je "
<< &i << endl;
cout << "Vrednost promenljive pokazivac_na_i je "
<< pokazivac_na_i << endl << endl;
// Dodeljivanje vrednosti koriscenjem indirekcije
*pokazivac_na_i = 17;
cout << "Vrednost i je " << i << endl;
cout << "Vrednost *pokazivac_na_i je " << *pokazivac_na_i
<< endl << endl;
return 0;
}
```

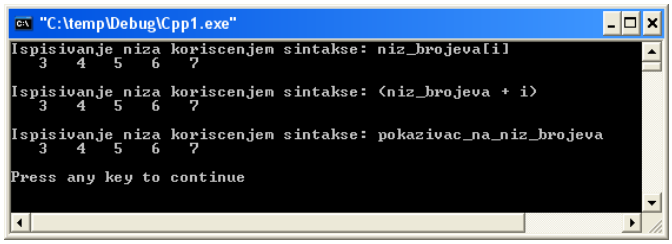


2 Zadatak:

Napisati program koji vrši operacije sa pokazivačima na nizove, kao i samim nizovima.

```
// Program demonstrira aritmeticke operacije sa pokazivacima
// i redovima.
#include <iostream.h>
#include <iomanip.h>
int main()
{
int niz_brojeva[5] = {3, 4, 5, 6, 7};
int* pokazivac_na_niz_brojeva;
```

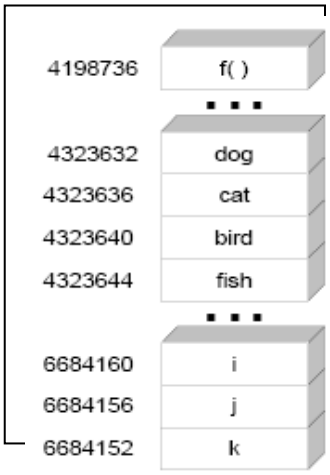
```
int i;
cout << "Ispisivanje niza koriscenjem sintakse: niz_brojeva[i]"<< endl;
for (i = 0; i < 5; ++i)
cout << setw(4) << niz_brojeva[i];
cout << endl << endl;
cout << "Ispisivanje niza koriscenjem sintakse: "<< "(niz_brojeva + i)" << endl;
for (i = 0; i < 5 ; ++i)
cout << setw(4) << *(niz_brojeva + i);
cout << endl << endl;
cout << "Ispisivanje niza koriscenjem sintakse: "
<< "pokazivac_na_niz_brojeva" << endl;
for (pokazivac_na_niz_brojeva = niz_brojeva;
pokazivac_na_niz_brojeva < niz_brojeva + 5;
++pokazivac_na_niz_brojeva)
cout << setw(4) << *pokazivac_na_niz_brojeva;
cout << endl << endl;
return 0;}
```

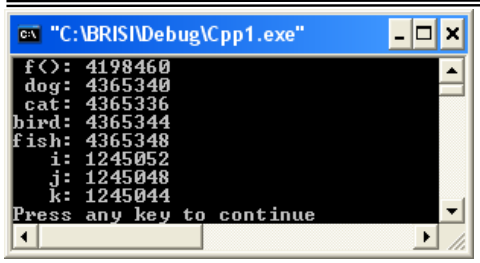


3 Zadatak:

Napraviti program za prikaz adresa funkcije i promenljivih u operativnoj memoriji.

```
#include <iostream.h>
#include<iomanip.h>
int dog, cat, bird, fish;
void f(int pet);

int main()
{int i, j, k;
cout << setw(6)<<"f(): " << (long)&f << endl;
cout << setw(6)<<"dog: " << (long)&dog << endl;
cout << setw(6)<<"cat: " << (long)&cat << endl;
cout << setw(6)<<"bird: " << (long)&bird << endl;
cout << setw(6)<<"fish: " << (long)&fish << endl;
cout << setw(6)<<"i: " << (long)&i << endl;
cout << setw(6)<<"j: " << (long)&j << endl;
cout << setw(6)<<"k: " << (long)&k << endl;
return 0;}

void f(int pet)
{cout << "pet id number: " << pet << endl;}
IZLAZ:
```



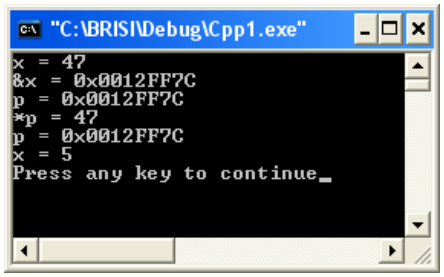


4 Zadatak:

Korišćenje ukazivača na adresu i indirekcije:

```
#include <iostream.h>
void f(int* p) ;
int main() {
int x = 47;
cout << "x = " << x << endl;
cout << "&x = " << &x << endl;
f(&x);     //poziv funkcije
cout << "x = " << x << endl;
return 0;}
void f(int* p) {
cout << "p = " << p << endl;
cout << "*p = " << *p << endl;
*p = 5;
cout << "p = " << p << endl;}
```

IZLAZ:



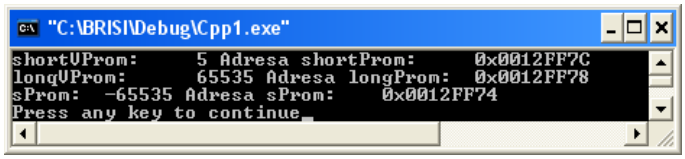
5 Zadatak:

Demonstriranje operatora adresa od, i adresa lokalnih promenljivih

```
 #include <iostream.h>
      int main()
      {unsigned short shortProm=5;
      unsigned long longProm=65535;
      long sProm = -65535;
       cout <<"shortVProm:\t" << shortProm;
      cout << " Adresa shortProm:\t";
      cout << &shortProm<< "\n";
      cout << "lonqVProm:\t" << longProm;
      cout << " Adresa longProm:\t" ;
      cout << &longProm << "\n";
```

```
    cout << "sProm:\t"<< sProm;
    cout << " Adresa sProm:\t" ;
    cout << &sProm << "\n";
   return 0;}
```

IZLAZ:

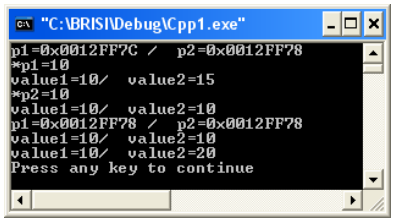


6 Zadatak:

Primena operatora adresa i ukazatelja:

```
#include <iostream.h>
main ()
{ int value1 = 5, value2 = 15;
int *p1, *p2;
p1 = &value1; // p1 = adresa value1
p2 = &value2; // p2 = adresa value2
cout << "p1=" << p1 << " /  p2=" << p2<<"\n" ;
*p1 = 10; // ukazatelj na p1=10
cout<<"*p1="<<*p1<<endl;
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
*p2 = *p1; // ukazatelj p2 jednak je ukazatelju p1
cout<<"*p2="<<*p2<<endl;
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
p1 = p2; // p1 = p2
cout << "p1=" << p1 << " /  p2=" << p2<<"\n" ;
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
*p1 = 20; // ukazatelj na p1=20
cout << "value1=" << value1 << "/  value2=" << value2<<endl;
return 0; }
```

IZLAZ:



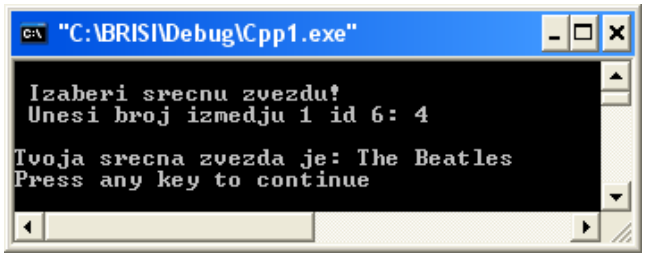
7 Zadatak:

Inicijalizacija pointera sa stringovima

```
#include <iostream.h>
int main() { // Inicijalizacija pointerskog polja
char *pstr[ ]= { "Robert Redford",
                 "Sting",
                 "Lassie",
                 "The Beatles",
```

```
                    "Boris Karloff",
                    "Oliver Hardy" };
    char *pstart = "Tvoja srecna zvezda je: ";
    int izbor = 0;
    cout << endl << " Izaberi srecnu zvezdu!"<<endl;
    cout << " Unesi broj izmedju 1 id 6: ";
    cin >> izbor;
    cout << endl;
    if(izbor >= 1 && izbor <= 6) // Provera ulaznih podataka
    cout << pstart << pstr[izbor-1]; // Izabrano ime zvezde
    else
    // Invalid input
    cout << "Zalim, nisi izabrao srecnu zvezdu.";
    cout << endl;
    return 0; }
```

IZLAZ:

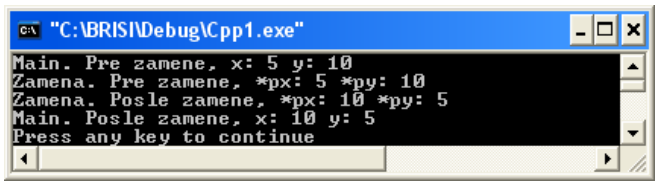


8 Zadatak:

Demonstrcija prelaza po referenci korišćenjem Pointera

```
#include <iostream.h>
    void zamena(int *x, int *y);
    int main()
    {       int x = 5, y = 10;
     cout << "Main. Pre zamene, x: " << x << " y: " << y << "\n";
     zamena(&x,&y);
     cout << "Main. Posle zamene, x: " << x << " y: " << y << "\n";
   return 0;     }
    void zamena (int *px, int *py)
    {       int temp;
     cout << "Zamena. Pre zamene, *px: " << *px << " *py: " << *py << "\n";
     temp = *px;
     *px = *py;
     *py = temp;
     cout << "Zamena. Posle zamene, *px: " << *px << " *py: " << *py << "\n";     }
```

IZLAZ:



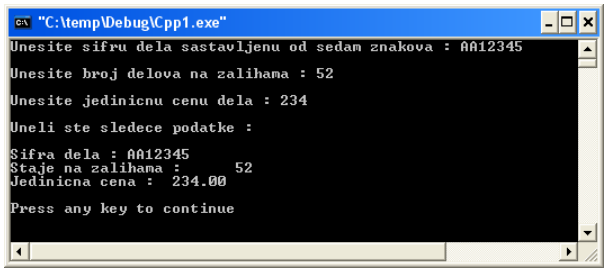
STRUKTURE

Primetimo da je struktura podataka definisana izvan funkcije main(), i da se kasnije tretira kao tip promenljive. Pristup pojedinom članu strukure se vrši sintaksom: ime_strukture.ime_clana.

1 Zadatak:

Napisati program koji formira strukturu podataka radi evidentiranja podataka u prodavnici delova. Zatim program zahteva unos podataka koji su definisani u strukturi i vrši ispisivanje unetih podataka na ekranu.

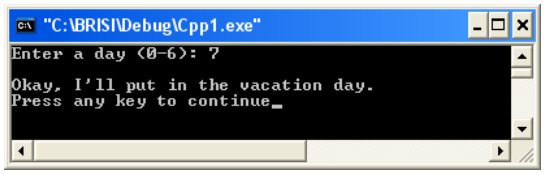
```
// Program ilustruje deklarisanje i upotrebu strukture promenljivih
// kao i opratora clanica za pristupanje promenljivima clanicama.
#include <iostream.h>
#include <iomanip.h>
struct OPIS_DELA
{char sifra_dela[8];
int kolicina_na_stanju;
double jedinicna_cena;};
int main()
{OPIS_DELA deo;
cout << setprecision(2)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
cout << "Unesite sifru dela sastavljenu od sedam znakova : ";
cin.getline(deo. sifra_dela, 8);
cout << endl;
cout << "Unesite broj delova na zalihama : ";
cin >> deo. kolicina_na_stanju;
cout << endl;
cout << "Unesite jedinicnu cenu dela : ";
cin >> deo. jedinicna_cena;
cout << endl;
cout << "Uneli ste sledece podatke :" << endl << endl;
cout << "Sifra dela : " << deo. sifra_dela << endl;
cout << "Staje na zalihama : " << setw(7)
<< deo. kolicina_na_stanju << endl;
cout << "Jedinicna cena : " << setw(7)
<< deo. jedinicna_cena << endl << endl;
return 0;}
```



2 Zadatak:

Primena izvedenog tipa podataka enum kao uvod u strukture:

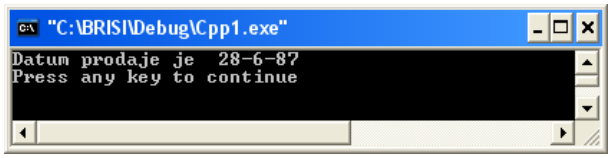
```
#include <iostream.h>
 int main()
 {
    enum Days { Sunday, Monday, Tuesday,
       Wednesday, Thursday, Friday, Saturday };
    int choice;
    cout << "Enter a day (0-6): ";
    cin >> choice;
    if (choice == Sunday || choice == Saturday)
      cout << "\nYou're already off on weekends!\n";
   else
     cout << "\nOkay, I'll put in the vacation day.\n";
   return 0;
 }
```



2 Zadatak:

Program za ilustraciju struktura

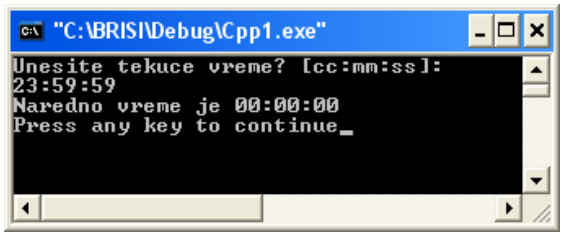
```
#include<iostream.h>
main()
        {
        struct datum {
                int dan;
                int mesec;
                int godina;
                   };
        struct datum prodaja;
        prodaja.dan=28;
        prodaja.mesec=6;
        prodaja.godina=1987;
        cout<< ''Datum prodaje je  ''<< prodaja.dan<<''-''<<prodaja.mesec<<''-''
<< prodaja.godina%100<<endl;,
        }
```



3 Zadatak:

Program za korekciju tekuceg vremena

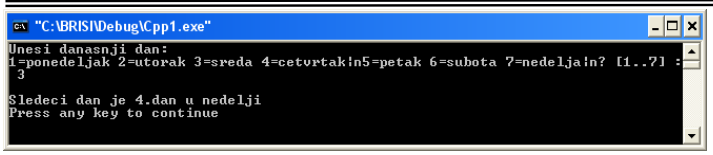
```
#include<iostream.h>
#include<iomanip.h>
void main()
        {
        struct vreme {      int sat;    int minut;          int sekund;          }_vreme[2];
        cout<<"Unesite tekuce vreme? [cc:mm:ss]:\n";
        cin>>_vreme[0].sat;
        cin>>_vreme[0].minut;
        cin>>_vreme[0].sekund;
        _vreme[1]=_vreme[0];
        if(++_vreme[1].sekund==60)
                {           _vreme[1].sekund=0;
                    if(++_vreme[1].minut==60)
                        {  _vreme[1].minut=0;
                            if(++_vreme[1].sat==24)
                                        _vreme[1].sat=0; }                      }
                cout << setiosflags( ios::fixed );
                cout.precision(2);
                cout<<"Naredno vreme je    "<<_vreme[1].sat<<":"<<_vreme[1].minut
                        <<":"<<_vreme[1].sekund<<endl;
        }
```



4 Zadatak:

Program za odredjivanje sledeceg dana

```
main()
        {           int k;
        enum dani {ponedeljak, utorak, sreda, cetvrtak, petak, subota, nedelja} sledeci_dan;
        printf("Unesi danasnji dan:\n");
        printf("1=ponedeljak 2=utorak 3=sreda 4=cetvrtak|n");
        printf("5=petak 6=subota 7=nedelja|n? [1..7] : ");
        scanf("%d", &k);
        k+=1; k%=7;
        sledeci_dan=(enum dani) k;
        printf("\nSledeci dan je %d.dan u nedelji\n", (int)sledeci_dan); }
```



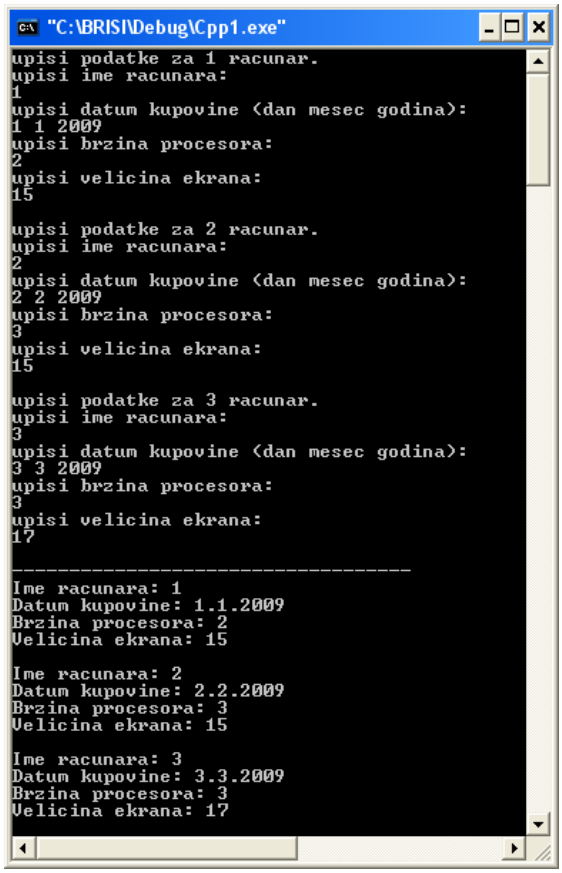
5 Zadatak:

Sastaviti program za unos karakteristika računara primenom struktura tipa Datum {int dan, mesec, godina;} i predmet { char ime[32];Datum d_kupovine ;int brzina_procesora; int velicina_ekrana; }. Broj računara za unos ograničiti.

```
#include<iostream.h>
struct Datum //Struktura datum
{int dan, mesec, godina;};

struct predmet //struktura tipa predmet
{char ime[32];
Datum d_kupovine;//ugnjezdena struktura tipa Datum
int brzina_procesora;
int velicina_ekrana; };

void upis(predmet racunar[]) //funkcija za upis podataka
{
for (int i=0; i<3; i++)
{
cout<< "upisi podatke za "<< i+1<< " racunar." <<endl;
cout <<"upisi ime racunara: "<<endl;
cin>> racunar[i].ime;
cout<< "upisi datum kupovine (dan mesec godina): "<<endl;
cin>> racunar[i].d_kupovine.dan >> racunar[i].d_kupovine.mesec>>
racunar[i].d_kupovine.godina;
cout<< "upisi brzina procesora: "<<endl;
cin>> racunar[i].brzina_procesora;
cout<< "upisi velicina ekrana: "<<endl;
cin>> racunar[i].velicina_ekrana;
cout<< endl;
}}

void izpis(predmet racunar[]) //funkcija za izpis podataka
{
for (int i=0; i<3; i++)
{
cout<< "Ime racunara: " <<racunar[i].ime<< endl;
cout<< "Datum kupovine: "<< racunar[i].d_kupovine.dan <<"."<<
racunar[i].d_kupovine.mesec<< "."
<<racunar[i].d_kupovine.godina<< endl;
cout<< "Brzina procesora: "<< racunar[i].brzina_procesora<< endl;
cout<< "Velicina ekrana: " <<racunar[i].velicina_ekrana<< endl;
cout<< endl;
}}
```

```
int main()
{predmet racunar[10]; //definicija polja racunar tipa predmet
upis (racunar); //poziv funkcije upis
cout<< "-----------------------------------"<<endl;
izpis(racunar); //poziv funkcije izpis
return 0;}
```

Izlaz :



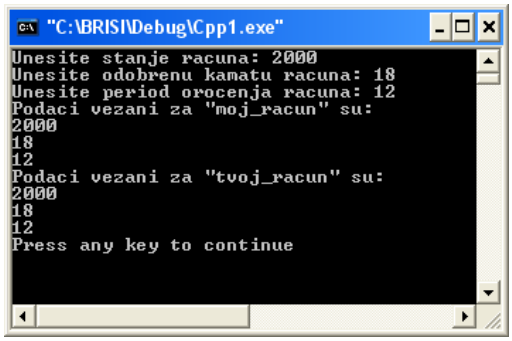
6 Zadatak:

Da vidimo kako bi sve to izgledalo u programu, koji će samo učitati od korisnika podatke o nekom računu, a zatim ih odštampati na ekran ali preuzete i u drugoj strukturi..

```
#include<iostream.h>
struct Racun{double stanje;double kamata;int period;};
int main ()
{
Racun moj_racun, tvoj_racun;//dve strukture tipa Racun
cout<<"Unesite stanje racuna: ";
cin>>moj_racun.stanje;
cout<<"Unesite odobrenu kamatu racuna: ";
```

```
cin>>moj_racun.kamata;
cout<<"Unesite period orocenja racuna: ";
cin>>moj_racun.period;
tvoj_racun=moj_racun;
cout<<"Podaci vezani za \"moj_racun\" su: \n";
cout<<moj_racun.stanje<<endl;
cout<<moj_racun.kamata<<endl;
cout<<moj_racun.period<<endl;
cout<<"Podaci vezani za \"tvoj_racun\" su: \n";
cout<<tvoj_racun.stanje<<endl;
cout<<tvoj_racun.kamata<<endl;
cout<<tvoj_racun.period<<endl;
return 0;}
```

Izlaz:



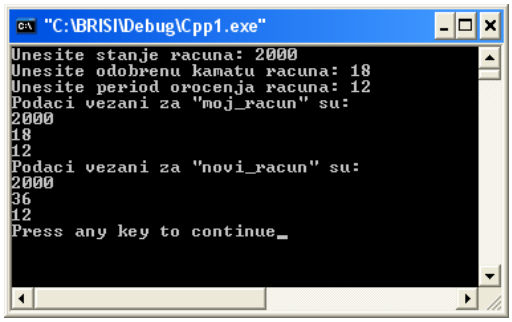
7 Zadatak:

Program demonstrira upotrebu funkcija u radu sa strukturama

```
#include <iostream.h>
struct Racun{ double stanje;double kamata;int period; };
Racun udvostruciKamatu( Racun );  //funkcija
int main ( )
{Racun moj_racun, novi_racun;//dve strukture tipa Racun
cout<<"Unesite stanje racuna: ";
cin>>moj_racun.stanje;
cout<<"Unesite odobrenu kamatu racuna: ";
cin>>moj_racun.kamata;
cout<<"Unesite period orocenja racuna: ";
cin>>moj_racun.period;
novi_racun=udvostruciKamatu(moj_racun);
cout<<"Podaci vezani za \"moj_racun\" su: \n";
cout<<moj_racun.stanje<<endl;
cout<<moj_racun.kamata<<endl;
cout<<moj_racun.period<<endl;
cout<<"Podaci vezani za \"novi_racun\" su: \n";
cout<<novi_racun.stanje<<endl;
cout<<novi_racun.kamata<<endl;
cout<<novi_racun.period<<endl;
```

```
return 0;}

Racun udvostruciKamatu(Racun stari_racun)    //funkcija
{ Racun temp=stari_racun;
temp.kamata=2*stari_racun.kamata;
return temp; }
```

Izlaz:

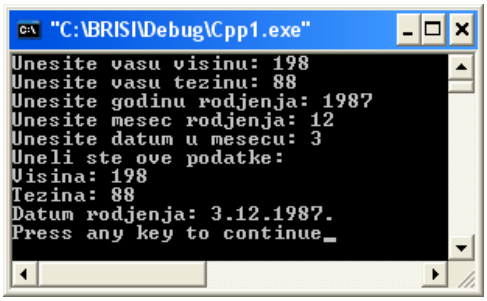


8 Zadatak:

Program demonstrira upotrebu struktura unutar struktura

```
#include <iostream.h>
struct Datum
{ int mesec;
int dan;
int godina;};
struct Osoba
{ double visina;
int tezina;
Datum rodjendan; };
int main ( )
{ Osoba osoba1;
cout<<"Unesite vasu visinu: ";
cin>>osoba1.visina;
cout<<"Unesite vasu tezinu: ";
cin>>osoba1.tezina;
cout<<"Unesite godinu rodjenja: ";
cin>>osoba1.rodjendan.godina;
cout<<"Unesite mesec rodjenja: ";
cin>>osoba1.rodjendan.mesec;
cout<<"Unesite datum u mesecu: ";
cin>>osoba1.rodjendan.dan;
cout<<"Uneli ste ove podatke: \n";
cout<<"Visina: "<<osoba1.visina<<endl;
cout<<"Tezina: "<<osoba1.tezina<<endl;
cout<<"Datum rodjenja: "<<osoba1.rodjendan.dan;
cout<<"."<<osoba1.rodjendan.mesec;
cout<<"."<<osoba1.rodjendan.godina<<"."<<endl;
return 0;}
```

Izlaz:

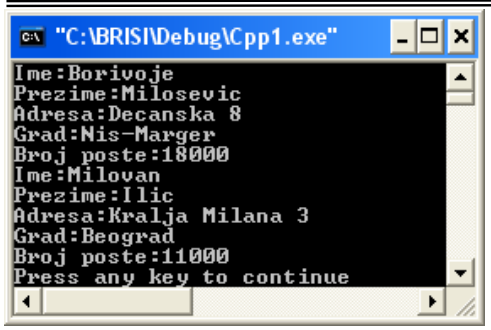


9 Zadatak:

Napraviti strukturu zaposleni i inicijalizovati je primenom komande za kopiranje stringa.

```
#include <string.h>
#include <iostream.h>
struct Zaposleni // Deklaracija strukture
{char ime[20]; //Deklaracija polja char s dvadeset znakova
char prezime[20];
char adresa[50];
char grad[20];
int BrojPoste;};
void main()
{
struct Zaposleni slog1; //Instanca strukture Zaposleni, a njen naziv je slog1
strcpy(slog1.ime, "Borivoje"); //Kopirati odgovarajuce vrednosti
strcpy(slog1.prezime, "Milosevic");
strcpy(slog1.adresa, "Decanska 8");
strcpy(slog1.grad, "Nis-Marger");
slog1.BrojPoste = 18000;
struct Zaposleni slog2 ={"Milovan","Ilic","Kralja Milana 3","Beograd",11000,};
 /* Kraci oblik kreiranja i  instanciranja strukture*/
cout<<"Ime:" <<slog1.ime<<endl;
cout<<"Prezime:" <<slog1.prezime<<endl;
cout<<"Adresa:" <<slog1.adresa<<endl;
cout<<"Grad:" <<slog1.grad<<endl;
cout<<"Broj poste:" <<slog1.BrojPoste<<endl;
cout<<"Ime:" <<slog2.ime<<endl;
cout<<"Prezime:" <<slog2.prezime<<endl;
cout<<"Adresa:" <<slog2.adresa<<endl;
cout<<"Grad:" <<slog2.grad<<endl;
cout<<"Broj poste:" <<slog2.BrojPoste<<endl;
}
```

Izlaz :



10 Zadatak:

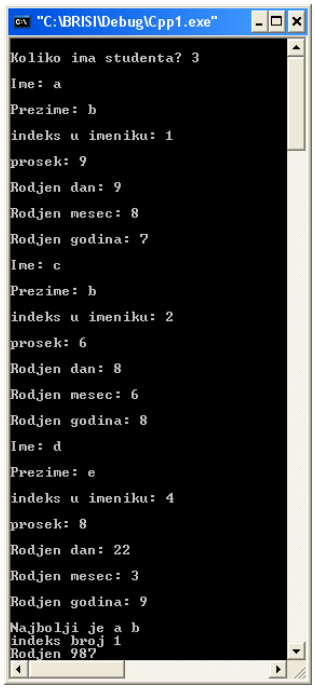
Primer programa u kojem se unose podaci za: ime, prezime, matični broj, prosek i datum rođenja za sve studente jednog godine (najviše 40). Posle unosa svih podataka traži se student s najboljim prosekom pa se na kraju programa na ekranu računara ispisuju svi podaci za studenta koji ima najbolji prosek.

Objašnjenje: U programu su definisane dve strukture: datum i student. Struktura datum služi za unos datuma rođenja studenta i sastoji se od dana, meseca i godine rođenja. Druga struktura se zove student i sadrži 2 niza znakova (za ime i prezime), broj indeksa i prosek ocena. Niz (polje) godina se sastoji od struktura tipa student. Na primer poziv za ispis promenljive *razred[0].rodjendan.godina* će dati za ispis godine rođenja za 1. studenta, a *razred[1].prosek* će dati ispis proseka za 2. studenta.

```
#include <iostream.h>
#include <string.h>
        struct datum
{int dan;
int mesec;
int godina;};
        struct student
{char ime[15];
char prezime[15];
int indeks;
float prosek;
struct datum rodjendan;// struktura rodjendan tipa datum
}razred[40];        /*polje razred ciji su clanovi strukture tipa student*/

int main()
{int n,i,k;
float max;
cout<<"\nKoliko ima studenta? ";
cin>>n;
if (n>40)
cout<<"Broj studenta ne moze biti >40";
else
{for(i=0;i<n;i++)        /*unos podataka za pojedinog studenta u polje razred*/
{cout<<"\nIme: ";
cin>>razred[i].ime;
cout<<"\nPrezime: ";
```

```
cin>>razred[i].prezime;
cout<<"\nindeks u imeniku: ";
cin>>razred[i].indeks;
cout<<"\nprosek: ";
cin>>razred[i].prosek;
cout<<"\nRodjen dan: ";
cin>>razred[i].rodjendan.dan;
cout<<"\nRodjen mesec: ";
cin>>razred[i].rodjendan.mesec;
cout<<"\nRodjen godina: ";
cin>>razred[i].rodjendan.godina;}
max=razred[0].prosek;    /*odredjivanje najboljeg proseka*/
k=0;
for(i=0;i<n;i++)
if (max<razred[i].prosek)
{max=razred[i].prosek;k=i;}
/*ispis najboljeg*/
cout<<"\nNajbolji je "<<razred[k].ime<<" " <<razred[k].prezime;
cout<<"\nindeks broj "<<razred[k].indeks;
cout<<"\nRodjen "<<razred[k].rodjendan.dan<<razred[k].rodjendan.mesec<<
razred[k].rodjendan.godina;
cout<<"\nS prosekom "<<razred[k].prosek;
}
return 0;
}
```

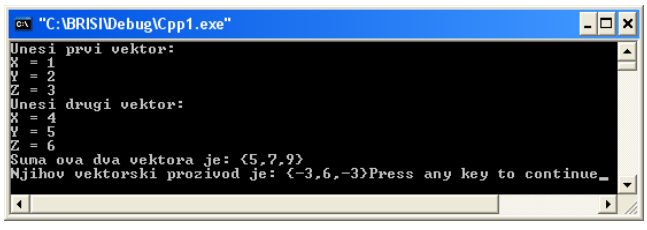
Izlaz:



11 Zadatak:

Strukture se *mogu prenositi kao parametri u funkcije*, kako po vrednosti, tako i po referenci. Takođe, za razliku od nizova, strukture se mogu i *vraćati kao rezultati iz funkcija*. Ova tehnika ilustrovana je u sledećem primeru, u kojem je definisan strukturni tip podataka "Vektor3d" koji opisuje vektor u prostoru koji se, kao što znamo, može opisati sa tri koordinate *x, y* i *z*. U prikazanom programu, prvo se čitaju koordinate dva vektora sa tastature, a zatim se računa i ispisuje njihova suma i njihov vektorski prozivod (pri čemu se vektor ispisuje u formi tri koordinate međusobno razdvojene zarezima, unutar vitičastih zagrada):

```
#include <iostream.h>
struct Vektor3d {double x, y, z;};
void UnesiVektor(Vektor3d &v)
{cout << "X = "; cin >> v.x;
cout << "Y = "; cin >> v.y;
cout << "Z = "; cin >> v.z;}
Vektor3d ZbirVektora(const Vektor3d &v1, const Vektor3d &v2)
{Vektor3d v3;
v3.x = v1.x + v2.x; v3.y = v1.y + v2.y; v3.z = v1.z + v2.z;
return v3;}

Vektor3d VektorskiProizvod(const Vektor3d &v1, const Vektor3d &v2)
{Vektor3d v3;
v3.x = v1.y * v2.z - v1.z * v2.y;
v3.y = v1.z * v2.x - v1.x * v2.z;
v3.z = v1.x * v2.y - v1.y * v2.x;
return v3;}

void IspisiVektor(const Vektor3d &v)
{cout << "{" << v.x << "," << v.y << "," << v.z << "}";}

int main()
{Vektor3d a, b;
cout << "Unesi prvi vektor:\n";
UnesiVektor(a);
cout << "Unesi drugi vektor:\n";
UnesiVektor(b);
cout << "Suma ova dva vektora je: ";
IspisiVektor(ZbirVektora(a, b));
cout << endl << "Njihov vektorski prozivod je: ";
IspisiVektor(VektorskiProizvod(a, b));
return 0;}
```

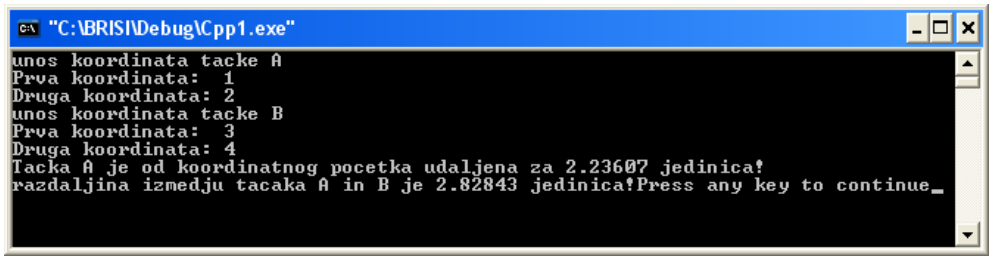


12 Zadatak:

Kreiraj strukturu tacka2D sa dve koordinate tipa //int. Napisi funkcije za unos podataka, izracunati udaljenje ove tacke od koorinatnog pocetka i funkciju za izracunavanje razdaljine izmedju tacaka!

```
#include <iostream.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>

struct tacka2D { int x,y;};
tacka2D A,B,C;
void unos(tacka2D &T) { //parameter je pozvan po referenci
  cout<<"Prva koordinata:  ";cin>>T.x;
  cout<<"Druga koordinata: ";cin>>T.y;}
float razdaljina (tacka2D T)
{ return (sqrt(float(pow(T.x,2)+pow(T.y,2))));}
float razdaljina(tacka2D T1, tacka2D T2)
{return (sqrt(float(pow(T1.x-T2.x,2)+pow(T1.y-T2.y,2))));}

void main(){
   puts("unos koordinata tacke A");unos(A);
   puts("unos koordinata tacke B");unos(B);
   cout<<"Tacka A je od koordinatnog pocetka udaljena za ";
   cout<<razdaljina(A)<<" jedinica!"<<endl;
   cout<<"razdaljina izmedju tacaka A in B je "<<razdaljina(A,B)<<" jedinica!";
   //Pretrazi podatke jos za tacku C! Koliki je obim trougaonika ABC?

   //Koja od tacaka A, B i C je najvise udaljena od koordinatnog pocetka?
   //Napisi funkciju HERON, koja ima za parametre tri tacke, koja vraca
   //povrsinu trougla koju dobijamo pozivom funkcije cout<<HERON(A,B,C);
   //Heronova formula p=sqrt(s*(s-a)*(s-b)*(s-c)), s=(a+b+c)/2

}
```

Izlaz:

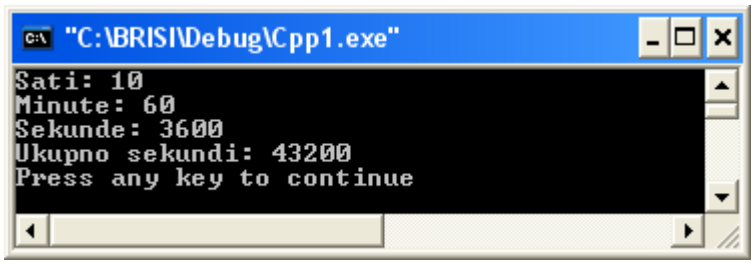


13 Zadatak:

Primer strukture ČAS, koja pamti podatke o trenutnom času. Sledi funkcija za pretvaranje časa u sekunde:

```
#include <iostream.h>
#include <conio.h>
struct Cas {
   int sati;
   int minute;
   int sekunde;
};

int VSekunde(Cas trenutni) {
   return 3600*trenutni.sati + 60*trenutni.minute + trenutni.sekunde;
}

void main() {
   Cas t;
   cout<<"Sati: ";cin >> t.sati;
   cout<<"Minute: ";cin >> t.minute;
   cout<<"Sekunde: ";cin >> t.sekunde;
   cout << "Ukupno sekundi: " << VSekunde(t) << endl;
}
```

Izlaz:



14 Zadatak:

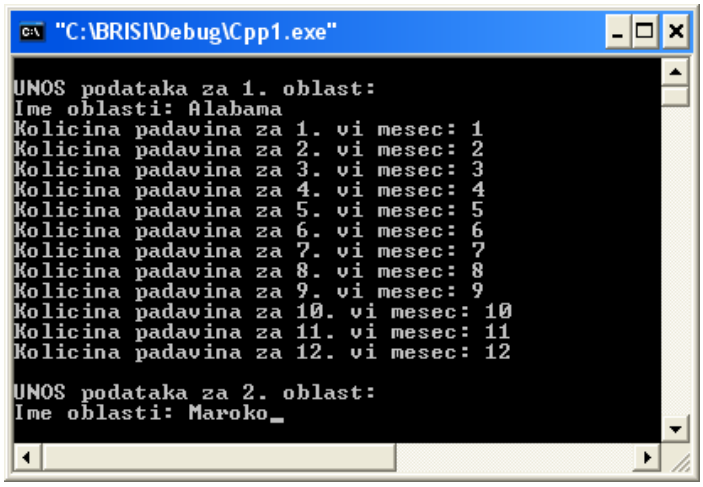
Za vodjenje evidencije o padavinama u zadnjoj godini potrebni su sledeci podaci:
-ime oblasti (do 20 char) i podaci o kolicini padavina u zadnjih 12 meseci
(12 realnih brojeva u polju/tabeli) (6. float)

Kreiraj strukturu podataka PADAVINE
Kreiraj tabelu OBLASTI, koja najvise ima 10 takvih struktura
Napisi funkciju UNOS(N)
Napisi funkciju POVP(OBLASTI,N),

```
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
```

```
struct PADAVINE
{   char ime_oblasti[30];
   float kolicina[12];};

PADAVINE OBLASTI[10];

void UNOS(int N)
{  cout<<"\nUNOS podataka za "<<N+1<<". oblast: \n";
  cout<<"Ime oblasti: ";
  cin>>OBLASTI[N].ime_oblasti;
  for (int i=0;i<12;i++)
{    cout<<"Kolicina padavina za "<<i+1<<". vi mesec: ";
    cin>>OBLASTI[N].kolicina[i];  }}

float POVP(PADAVINE OBLASTI[10], int N
{  float suma=0;
  for (int i=0;i<12;i++)
    suma=suma + OBLASTI[N].kolicina[i];
  return suma; }

void main()
{  for (int i=0;i<10;i++)
    UNOS(i);
  cout<<"Prosecna kolicina padavina oblasti sa indeksom 5 je "<<POVP(OBLASTI,5); }
```

Izlaz:



15 Zadatak:

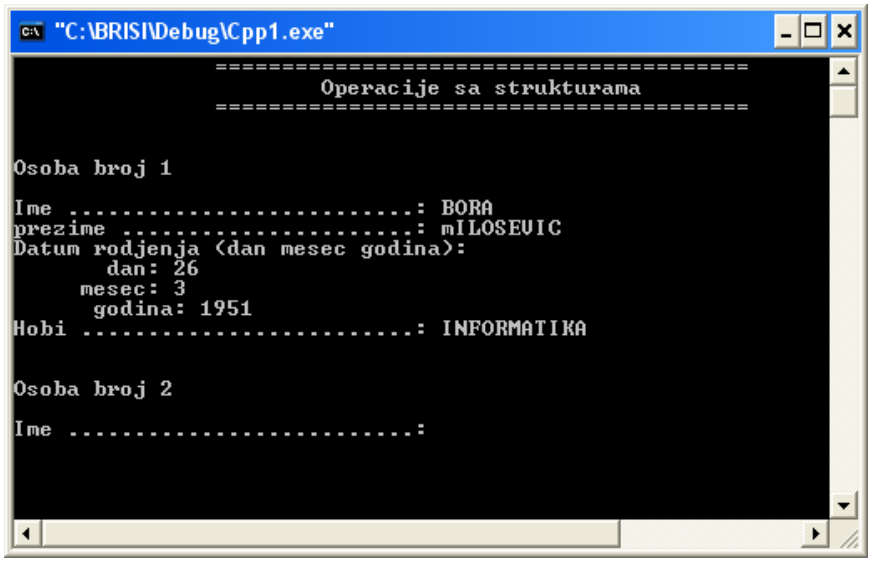
Napisite program, koji sakuplja podatke o 5 osoba. Svaka osoba ima sledece podatke: ime, prezime, datum rodjenja, najdrazi hobi.
Program ispisuje podatke o osobama uredjene po prezimenu i imenu, pa zatim uredjene po hobiju.
Ulazni podaci o osobama su: ime, prezime, dan, mesec, godina, hobi.

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <conio.h>
//inicializacija globalnih konstanti
const int n1 = 51;
//inicilazacija struktura
struct dat
{  int dan;
  int mesec;
  int godina;};


struct osoba
{  char ime[n1];
  char prezime[n1];
  dat datum;
  char hobi[n1];};
//preostale funkcije
//void izpis(osoba osobe[ ], int f1);

void swp(char *a, char *b)
   {char tmp[n1];
   strcpy(tmp, a);
   strcpy(a, b);
   strcpy(b, tmp);}

void swp_dat(int& a, int& b)
   {int temp;
   temp = a;
   a = b;
   b = temp;}

void swap(osoba& x, osoba& y)
  //funkcija ,koja menja dve strukture
  //x <=> y (samo nizi)
  {swp(x.ime, y.ime);
  swp(x.prezime, y.prezime);
  swp(x.hobi, y.hobi);
  //zamena datuma
  swp_dat(x.datum.dan ,y.datum.dan);
  swp_dat(x.datum.mesec, y.datum.mesec);
```

```
  swp_dat(x.datum.godina, y.datum.godina);}
//funkcija main
                void main() {
  //glava programa
  cout<< setw(55) << "======================================" <<endl;
  cout<< setw(55) << "         Operacije sa strukturama        " <<endl;
  cout<< setw(55) << "======================================" <<endl;
  cout<<endl;
  //inicializacija potrebnih struktura
  osoba osobe[5];
  //unos podataka u strukture pomocu for petlje
  for(int i = 0; i < 5; i++) {
    cout<<endl;
    cout<< "Osoba broj " << i + 1 <<endl;
    cout<<endl;
    cout<< "Ime ..........................: " ;
    cin>> osobe[i].ime ;
    cout<< "prezime ......................: " ;
    cin>> osobe[i].prezime ;
    cout<< "Datum rodjenja (dan mesec godina): \n" ;
    cout<< "       dan: "; cin>> osobe[i].datum.dan;
    cout<< "     mesec: "; cin>> osobe[i].datum.mesec;
    cout<< "      godina: "; cin>> osobe[i].datum.godina ;
    cout<< "Hobi .........................: " ;
    cin>> osobe[i].hobi ;
    cout<<endl;  }
  //uredjenje struktura po prezimenu i imenu
  for(int f2 = 0; f2 < 5; f2++) {
   for(int f3 = 0; f3 < 5; f3++){
     if(strcmp(osobe[ f2 ].prezime ,osobe[ f3 ].prezime) < 0) {//ce je prvi niz krajsi od drugega
se izvrsi zamenjava struktur
      swap(osobe[ f2 ], osobe[ f3 ] );
     }
     else if(strcmp(osobe[ f2 ].prezime ,osobe[ f3 ].prezime) == 0 && f2 != f3) {//ce sta niza
enaka se zacne gledati po imenu
      if(strcmp(osobe[ f2 ].ime, osobe[ f3 ].ime) < 0) {
        swap(osobe[ f2 ], osobe[ f3 ] );      }     }   }  }
  //ispis struktura
  cout<<endl;
  cout<< "IZPIS OSOBA UREDJENIH PO PREYIMENU I IMENU" <<endl;
  for(int f1 = 0; f1 < 5; f1++) {
    cout<<endl;
    cout<< osobe[ f1 ].prezime << " " ;
    cout<< osobe[ f1 ].ime << " ";
    cout<< osobe[ f1 ].datum.dan << " "<< osobe[f1].datum.mesec << " "
    << osobe[ f1 ].datum.godina << " ";
    cout<< osobe[ f1 ].hobi <<endl;
  }
  //uredjenje strukture po hobiju
  for(int f7 = 0; f7 < 5; f7++)  {
    for(int f8 = 0; f8 < 5; f8++) {
      if(strcmp(osobe[f7].hobi ,osobe[f8].hobi) < 0)
                {  swap(osobe[f7], osobe[f8]); }
```

```
   }
  }
  //ispis struktura
  cout<<endl;
  cout<< "IZPIS OSOBA UREDJENIH PO HOBIJU" <<endl;
  for(int f5 = 0; f5 < 5; f5++)
{    cout<<endl;
    cout<< osobe[ f5 ].prezime << " " ;
    cout<< osobe[ f5 ].ime << " ";
    cout<< osobe[ f5 ].datum.dan << " "<< osobe[ f5 ].datum.mesec << " "
    << osobe[ f5 ].datum.godina << " ";
    cout<< osobe[ f5 ].hobi <<endl;
  }
  cout<<endl;
  getch();
}
```

Izlaz:



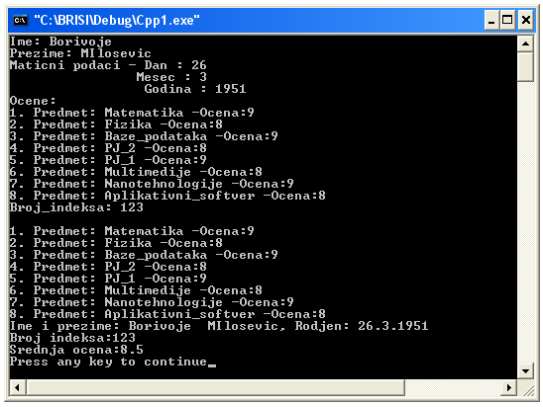
16 Zadatak:

Date su strukture datum, Predmet i apsolvent. Datum definise potrebne datume, struktura Predmet predmete {"Matematika","Fizika","Baze_podataka","PJ_2","PJ_1", "Multimedije", "Nanotehnologije","Aplikativni_softver"}, promenljiva M pamti podatke o jednom apsolventu, svakoj pojedinačnoj oceni koju je postigao na izabranim predmetima.
- napisi funkciju unos(M), za unos svih studentskih podataka
- napisi for petlju, koja ispisuje izabrane predmete i ocene
- napisi deo programa, koji ispisuje ime, prezime i datum rodjenja studenta, broj indeksa i srednju ocenu*/

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

```
#include<string.h>

struct datum {   int dan,mesec,godina;  };

struct Predmet {char imepredmeta[20];};
struct Predmet predmeti[8] ={"Matematika","Fizika","Baze_podataka","PJ_2","PJ_1",
"Multimedije", "Nanotehnologije","Aplikativni_softver"};
struct apsolvent
{  char ime[20],prezime[20];
  datum datum_rodjenja;      //Ugnježđena struktura
  int ocene[8],broj_indeksa; };
void unos(apsolvent &M)
{ cout<<"Ime: "; cin>>M.ime;
  cout<<"Prezime: "; cin>>M.prezime;
  cout<<"Maticni podaci - Dan : ";cin>>M.datum_rodjenja.dan;
  cout<<"               Mesec : ";cin>>M.datum_rodjenja.mesec;  // Poziv strukture u strukturi
  cout<<"               Godina : ";cin>>M.datum_rodjenja.godina;
  cout<<"Ocene:\n" ;
  for (int i=0;i<8;i++)
{   cout<<i+1<<". Predmet: "<<predmeti[i].imepredmeta<<" -Ocena:";
    cin>>M.ocene[i];  }
  cout<<"Broj_indeksa: ";cin>>M.broj_indeksa; }

void main()
{ float prolaz=0.0;
  apsolvent M;  // definisanje instance M strukture tipa apsolvent
  //funkcija za unos podataka
  unos(M);
  //Ocene na predmetima:
  for (int i=0;i<8;i++)
  { cout<<"\n"<<i+1<<". Predmet: "<<predmeti[i].imepredmeta<<" -Ocena:"
  <<M.ocene[i];prolaz=prolaz+M.ocene[i]; }
  //stavka, koja zapisuje ime, prezime i datum rodjenja studenta
  cout<<"\nIme i prezime: "<<M.ime<<"  "<<M.prezime
<<", Rodjen: "<<M.datum_rodjenja.dan<<"."<<M.datum_rodjenja.mesec
<<"."<<M.datum_rodjenja.godina<<endl;
  cout<<"Broj indeksa:"<<M.broj_indeksa<<endl;
  cout<<"Srednja ocena:"<<prolaz/8.<<endl;}
```

Izlaz:



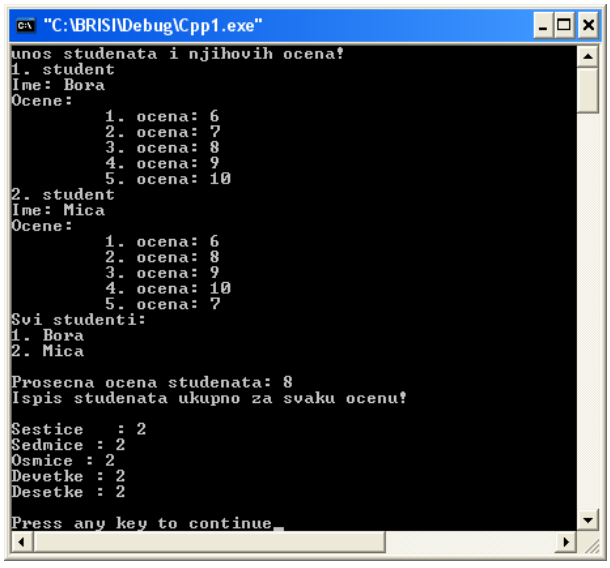
17 Zadatak:

Napisite program, koji najpre sakuplja podatke o N studenata i za svakoga
posebno po deset ocena.
- napisi funkciju za ispis studenata, koji imaju jednu ili vise negativnih ocena
- napisi funkciju, koja vraca prosecnu ocenu svih studenata u tabeli
- napisi funkciju, koja ispisuje imena svih studenata

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
//inicializacija globalnih konstanti
const int N = 2;
struct student {  //deklaracija strukture
  char ime[50];
  int ocene[5];
};
student tabela[N];  //tabela N struktur
void unos(student tabela[N]){  //unos podataka u tabelu
 cout<<"unos studenata i njihovih ocena!"<<endl;
 for (int i=0;i<N;i++){
   cout<<i+1<<". student"<<endl;
   cout<<"Ime: ";cin>>tabela[i].ime; //unos imena
   cout<<"Ocene:"<<endl;
   for (int j=0;j<5;j++){ //unos ocena za svakog studenta
       cout<<setw(10)<<j+1<<". ocena: ";
       cin>>tabela[i].ocene[j];
   }
 }
}
void ispisinepolozeno(student tabela[N]){ //i nepolozenim predmetima!\n"<<endl;
  for (int i=0;i<N;i++){
    for (int j=0;j<5;j++)
      if (tabela[i].ocene[j]==5){
        cout<<tabela[i].ime<<endl;
        break;
      }
  }
}
float prosecnaocena(student *D){  //parameter funkcije je ukazatelj na strukturu
  int suma=0; //pocetna vrednost
  for (int i=0;i<N;i++){
    for (int j=0;j<5;j++)
      suma=suma+D[i].ocene[j];  //sabiranje ocena
  }
 return float(suma)/(5*N);  //funkcija vraca prosecnu vrednost
}
void izpisimen(student *D) {  //funkcija ispise imena studenata
  cout<<"Svi studenti:"<<endl;
  for (int i=0;i<N;i++)
```

```
    cout<<i+1<<". "<<D[i].ime<<endl;
}
void pregledocen(student tabela[N]){ //Pregled i ispis prosecnih ocena
 cout<<"\nIspis studenata ukupno za svaku ocenu!\n"<<endl;
 int sest=0,sedam=0,osam=0,devet=0,deset=0;
 for (int i=0;i<N;i++)
{   for (int j=0;j<5;j++)
    switch (tabela[i].ocene[j])
 {     case 6: sest++;
            break;
       case 7: sedam++;
            break;
       case 8: osam++;
            break;
       case 9: devet++;
            break;
      case 10: deset++;
            break;     } }
 cout<<"Sestice   : "<<sest<<endl;
 cout<<"Sedmice : "<<sedam<<endl;
 cout<<"Osmice : "<<osam<<endl;
 cout<<"Devetke : "<<devet<<endl;
 cout<<"Desetke : "<<deset<<endl; }
void main()
 {  unos(tabela);
 ispisinepolozeno(tabela);
 izpisimen(tabela);
 cout<<"\nProsecna ocena studenata: "<<prosecnaocena(tabela);
 pregledocen(tabela);
 cout<<endl; }
```

Izlaz:

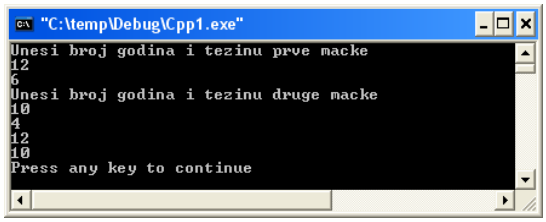


KLASE

1 Zadatak:

Primer klase macka, sa dodatim funkcijama koje vraćaju godine i težinu.

```
#include <iostream>
using namespace std;

class macka
{
public:
void jedi() {cout<<"mljac, mljac, mljac"<<endl;}
void spavaj() {cout<<"zzz..zzzzz"<<endl;}
void predi() {cout<<"prrr...."<<endl;}
void postavi_god(int x) {_god=x;}
void postavi_tezinu(int t) {_tezina=t;}
void vrati_god() {cout<<_god<<endl;}
void vrati_tezinu() {cout<<_tezina<<endl;}

private:
int _god;
int _tezina;
};

int main()
{
macka Garfild, Tom;
int x, t;

cout<<"Unesi broj godina i tezinu prve macke"<<endl;
cin>>x>>t;
Garfild.postavi_god(x);
Garfild.postavi_tezinu(t);
cout<<"Unesi broj godina i tezinu druge macke"<<endl;
cin>>x>>t;
Tom.postavi_god(x);
Tom.postavi_tezinu(t);
Garfild.vrati_god();
Tom.vrati_god();
return 0;
}
```



Svaki objekat sadrži sopstvenu kopiju podataka članova klase. Tom ima sopstvene vrednosti za težinu i godine, isto tako I Garfild. S druge strane postoji samo jedna kopija svake funkcije članice klase. Objekat Tom i Garfild pozivaju istu kopiju bilo koje određene funkcije članice. Videli smo da funkcija članica može da koristi članove svoje klase bez primene operatora za pristup članovima. Ako se funkcija vrati_god() pozove za objekat Tom onda podaci članovi kojima pristupa funkcija vrati_god() su podaci članovi objekta Tom. Ako se funkcija vrati_god() pozove za objekat Garfild, podaci članovi kojima ona pristupa su podaci članovi objekta Garfild. Kako funkcija vrati_god() zna kojim podacima treba da pristupi?

Odgovor na ovo pitanje je pokazivač this. Svaka funkcija članica sadrži pokazivač na adresu objekta za koji je ta funkcija pozvana i taj pokazivač se naziva this.

Ako imamo funkciju:

void vrat_god() {cout<<_god<<endl;} nju kompajler (interno) prevodi u: void vrati_god(macka *this) {cout<<this->_god<<endl;} dok poziv funkcije zapravo postaje umesto

Tom.vrati_god(); postaje vrati_god(&Tom);

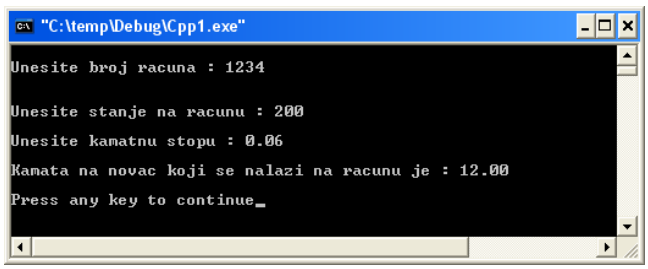
Dakle, svaki objekat ima kao član i pokazivač na njega samog, pokazivač this. Pokazivač this može se koristiti i eksplicitno unutar funkcije članice klase i o tome ćemo tek da pričamo.

2 Zadatak:

Napisati program koji kreira klasu bankovni račun i u funkciji main() izvršiti poziv njenih public funkcija-članica.

```
// Program ilustruje upotrebu funkcija koje su clanice klase
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
class Bankovni_Racun
{
private:
string broj_racuna;
double stanje_na_racunu;
double kamatna_stopa;
public:
Bankovni_Racun(string, double, double);
double Izracunaj_Kamatu();
};
Bankovni_Racun::Bankovni_Racun(string broj, double stanje, double
kamata)
{
broj_racuna = broj;
stanje_na_racunu = stanje;
kamatna_stopa = kamata;
}
```

```
double Bankovni_Racun::Izracunaj_Kamatu()
{
return stanje_na_racunu * kamatna_stopa;
}
int main()
{
cout << setprecision(2)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
string racun; // Promenljive za smestanje unetih podtaka
double na_racunu;
double stopa;
// Podaci o racunu koje unosi korisnik
cout << "\nUnesite broj racuna : ";
getline(cin, racun);
cout << endl;
cout << "Unesite stanje na racunu : ";
cin >> na_racunu;
cout << endl;
cout << "Unesite kamatnu stopu : ";
cin >> stopa;
// Inicijalizacija objekta
Bankovni_Racun Racun_1(racun, na_racunu, stopa);
cout << endl;
cout << "Kamata na novac koji se nalazi na racunu je : "
<< Racun_1.Izracunaj_Kamatu() << endl << endl;
return 0;}
```



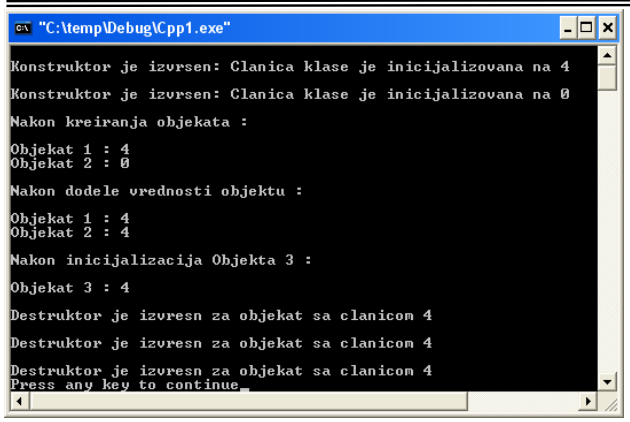
Primetimo da se iz spoljašnje funkcije, u ovom slučaju main() može direktno pristupiti isključivo javnim članicama klase. Privatnim članicama klase se može pristupiti isključivo korišćenjem javnih članica-funkcija, ukoliko je to dozvoljeno prilikom implementacije klase. Funkcije koje pristupaju privatnim članicama klase i ne menjaju njihovu vrednost su funkcije aksesori, dok se funkcije koje mogu da promene vrednost privatne članice nazivaju mutatori. Funkcije članice se nazivaju još i metode klase.

3 Zadatak:

Napisati program koji međusobno dodeljuje vrednosti objektima.

```
// Program ilustruje dodeljivanje jednog objekta drugome.
// Inicijalizacija objekta ne koristi konstruktor
// sa jednim argumentom.
```

```
#include <iostream>
using namespace std;
class Klasa_Za_Testiranje
{
private:
int n;
public:
Klasa_Za_Testiranje(int);
~Klasa_Za_Testiranje();
int Prosledi_Vrednost();
};
Klasa_Za_Testiranje::Klasa_Za_Testiranje(int i)
{
n = i;
cout << endl;
cout << "Konstruktor je izvrsen: Clanica klase je "
<< "inicijalizovana na " << n << endl;
}
Klasa_Za_Testiranje::~Klasa_Za_Testiranje()
{
cout << endl;
cout << "Destruktor je izvresn za objekat sa clanicom "
<< n << endl;
}
int Klasa_Za_Testiranje::Prosledi_Vrednost()
{
return n;
}
int main()
{
Klasa_Za_Testiranje objekat1(4);
Klasa_Za_Testiranje objekat2(0);
cout << endl;
cout << "Nakon kreiranja objekata :" << endl <<endl;
cout << "Objekat 1 : " << objekat1.Prosledi_Vrednost() << endl;
cout << "Objekat 2 : " << objekat2.Prosledi_Vrednost()
<< endl << endl;
objekat2 = objekat1;
cout << "Nakon dodele vrednosti objektu :" << endl << endl;
cout << "Objekat 1 : " << objekat1.Prosledi_Vrednost() << endl;
cout << "Objekat 2 : " << objekat2.Prosledi_Vrednost()
<< endl << endl;
Klasa_Za_Testiranje objekat3 = objekat1;
cout << "Nakon inicijalizacija Objekta 3 :" << endl << endl;
cout << "Objekat 3 : " << objekat3.Prosledi_Vrednost() << endl;
return 0;
}
```

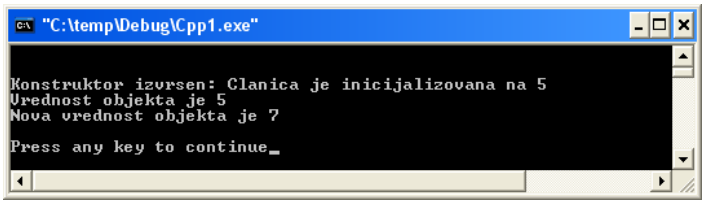


4 Zadatak:

Napisati program koji koristi pokazivač na objekat, kao i operator ->.

```
// Program prikazuje upotrebu pokazivaca na objekat
// kao i upotrebu operatora -> za pozivanje metode.
#include <iostream>
using namespace std;
class Klasa_Za_Testiranje
{
private:
int n;
public:
Klasa_Za_Testiranje(int i = 0); // Konstrukor sa jednim
// clanom i default vrednoscu
int Prosledi_Vrednost() const;
void Promeni_Vrednost(int);
};
Klasa_Za_Testiranje::Klasa_Za_Testiranje(int i)
{
n = i;
cout << endl << endl;
cout << "Konstruktor izvrsen: Clanica je inicijalizovana na "
<< n << endl;
}
int Klasa_Za_Testiranje::Prosledi_Vrednost() const
{
return n;
}
void Klasa_Za_Testiranje::Promeni_Vrednost(int i)
{
n = i;
}
int main()
{
Klasa_Za_Testiranje objekat1(5);
Klasa_Za_Testiranje* pokazivac_na_objekat = &objekat1;
```

```
cout << "Vrednost objekta je "
<< pokazivac_na_objekat -> Prosledi_Vrednost();
pokazivac_na_objekat -> Promeni_Vrednost(7);
// Menja vrednost na koju pokazuje pointer
cout << endl;
cout << "Nova vrednost objekta je "
<< pokazivac_na_objekat -> Prosledi_Vrednost();
cout << endl << endl;
return 0;
}
```

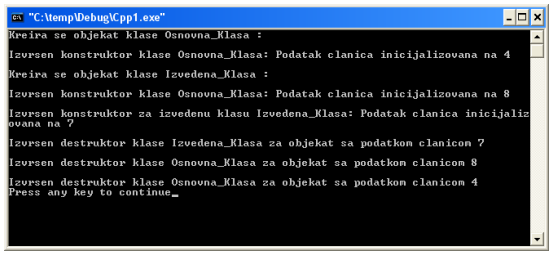


5 Zadatak:

Napisati program koji izvršava konstruktore i destruktore u osnovnoj i izvedenoj klasi.

```
// Program pokazuje kako se izvrsavaju konstruktori i destruktori
// u osnovnoj klasi
#include <iostream>
using namespace std;
class Osnovna_Klasa
{
protected:
int n;
public:
Osnovna_Klasa(int i = 0); // Konstruktor sa jednim argumentom
~Osnovna_Klasa();
};
Osnovna_Klasa::Osnovna_Klasa(int i) : n(i)
{
cout << endl;
cout << "Izvrsen konstruktor klase Osnovna_Klasa: "
<< "Podatak clanica inicijalizovana na " << n << endl;
}
Osnovna_Klasa::~Osnovna_Klasa()
{
cout << endl;
cout << "Izvrsen destruktor klase Osnovna_Klasa "
<< "za objekat sa podatkom clanicom " << n << endl;
}
class Izvedena_Klasa : public Osnovna_Klasa
{
protected:
int m;
public:
```

```
Izvedena_Klasa(int j = 0);
~Izvedena_Klasa();
};
Izvedena_Klasa::Izvedena_Klasa(int j) : Osnovna_Klasa(j+1), m(j)
{
cout << endl;
cout << "Izvrsen konstruktor za izvedenu klasu Izvedena_Klasa: "
<< "Podatak clanica inicijalizovana na "
<< m <<endl;
}
Izvedena_Klasa::~Izvedena_Klasa()
{
cout << endl;
cout << "Izvrsen destruktor klase Izvedena_Klasa "
<< "za objekat sa podatkom clanicom "
<< m << endl;
}
int main()
{
cout << "Kreira se objekat klase Osnovna_Klasa : " << endl;
Osnovna_Klasa obekt_osnovne_klase(4);
cout << endl;
cout << "Kreira se objekat klase Izvedena_Klasa : " << endl;
Izvedena_Klasa obekt_izvedene_klase(7);
return 0;
}
```



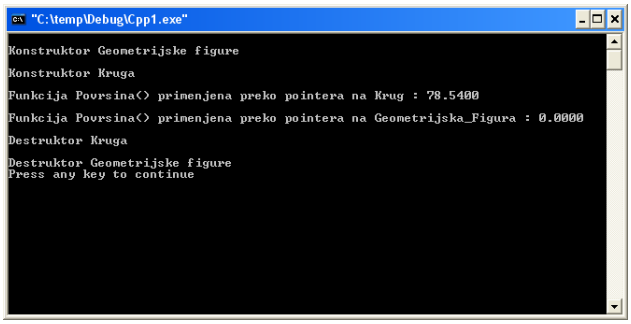
6 Zadatak:

Napisati program u kome se defiu geometrijske figure korišćenjem izvedenih klasa.

```
// Program prikzuje upotrebu poitera i metoda
// u hijerarhiji klase.
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

class Geometrijska_Figura
{
protected:
```

```
string tip_geometrijske_figure;
public:
Geometrijska_Figura
(string tip_figure = "Geometrijska figura"):
tip_geometrijske_figure(tip_figure)
{cout << endl << "Konstruktor Geometrijske figure" << endl;}
string Prosledi_Tip() {return tip_geometrijske_figure;}
~Geometrijska_Figura()
{cout << endl << "Destruktor Geometrijske figure" << endl;}
double Povrsina() {return 0.0;}
};
class Pravougaonik : public Geometrijska_Figura
{
protected:
double visina, sirina;
public:
Pravougaonik(double v, double s):
Geometrijska_Figura("Pravougaonik"), visina(v),sirina(s)
{cout << endl << "Konstruktor Pravougaonika" << endl;}
~Pravougaonik()
{cout << endl << "Destruktor Pravougaonika" << endl;}
double Povrsina() {return visina * sirina;}
};
class Krug : public Geometrijska_Figura
{
protected:
double poluprecnik;
public:
Krug(double p) : Geometrijska_Figura("Krug"), poluprecnik(p)
{cout << endl << "Konstruktor Kruga" << endl;}
~Krug() {cout << endl << "Destruktor Kruga" << endl;}
double Povrsina() {return 3.1416 * poluprecnik * poluprecnik;}
};
class Trapez : public Geometrijska_Figura
{
protected:
double osnova1, osnova2, visina;
public:
Trapez(double o1, double o2, double v):
Geometrijska_Figura("Trapez"), osnova1(o1), osnova2(o2),
visina(v)
{cout << endl << "Konstruktor Trapeza" << endl;}
~Trapez() {cout << endl << "Destruktor Trapeza" << endl;}
double Povrsina() {return 0.5 * visina * (osnova1 + osnova2);}
};

int main()
{
cout << setprecision(4)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
Geometrijska_Figura* pokazivac_na_geometrijsku_figuru_1;
Krug* pokazivac_na_krug_1;
```

```
Krug krug_1(5);
pokazivac_na_geometrijsku_figuru_1 = &krug_1;
// Pokazivac na figuru pokazuje na krug
pokazivac_na_krug_1 = &krug_1;
// Pokazivac na krug pokazuje na krug
cout << endl;
cout << "Funkcija Povrsina() primenjena preko pointera na Krug : "
<< pokazivac_na_krug_1 -> Povrsina() << endl << endl;
cout << "Funkcija Povrsina() primenjena "
<< "preko pointera na Geometrijska_Figura : "
<< pokazivac_na_geometrijsku_figuru_1 -> Povrsina()
<< endl;
return 0;
}
```

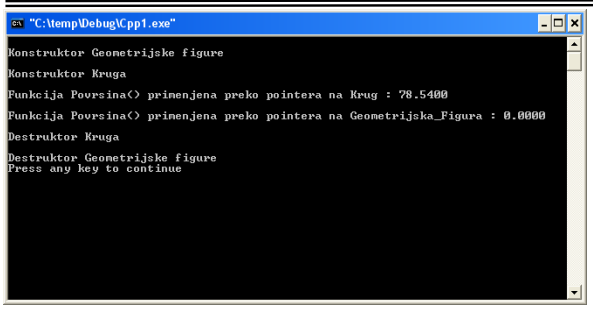


7 Zadatak:

Napisati program u kome se koristi virtual funkcija.

```
// Program ilustruje upotrebu virtual funkcije
// u hijerarhihi klase.
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
class Geometrijska_Figura
{
protected:
string tip_geometrijske_figure;
public:
Geometrijska_Figura(string tip_figure= "Geometrijska_Figura"):
tip_geometrijske_figure(tip_figure)
{cout << endl << "Konstruktor Geometrijske figure" << endl;}
string Prosledi_Tip() {return tip_geometrijske_figure;}
~Geometrijska_Figura()
{cout << endl << "Destruktor Geometrijske figure" << endl;}
virtual double Povrsina() {return 0.0;}
};
class Pravougaonik : public Geometrijska_Figura
{
```

```
protected:
double visina, sirina;
public:
Pravougaonik(double v,double s):
Geometrijska_Figura("Pravougaonik"), visina(v),sirina(s)
{cout << endl << "Konstruktor Pravougaonika" << endl;}
~Pravougaonik()
{cout << endl << "Destruktor Pravougaonika" << endl;}
double Povrsina() {return visina * sirina;}
};
class Krug : public Geometrijska_Figura
{
protected:
double poluprecnik;
public:
Krug(double p) : Geometrijska_Figura("Krug"), poluprecnik(p)
{cout << endl << "Konstruktor Kruga" << endl;}
~Krug() {cout << endl << "Destruktor Kruga" << endl;}
double Povrsina() {return 3.1416 * poluprecnik * poluprecnik;}
};
class Trapez : public Geometrijska_Figura
{
protected:
double osnova1, osnova2, visina;
public:
Trapez(double o1, double o2, double v):
Geometrijska_Figura("Trapez"), osnova1(o1), osnova2(o2),
visina(v)
{cout << endl << "Trapezoid Constructor" << endl;}
~Trapez() {cout << endl << "Trapezoid Destructor" << endl;}
double Povrsina() {return 0.5 * visina * (osnova1 + osnova2);}
};
int main()
{
cout << setprecision(4)
<< setiosflags(ios::fixed)
<< setiosflags(ios::showpoint);
Geometrijska_Figura* pokazivac_na_geometrijsku_figuru_1;
Krug* pokazivac_na_krug_1;
Krug krug_1(5);
pokazivac_na_geometrijsku_figuru_1 = &krug_1;
// Pokazivac na figuru pokazuje na krug
pokazivac_na_krug_1 = &krug_1;
// Pokazivac na krug pokazuje na krug
cout << endl;
cout << "Funkcija Povrsina() primenjena preko pointera na Krug : "
<< pokazivac_na_krug_1 -> Povrsina() << endl << endl;
cout << "Funkcija Povrsina() primenjena "
<< "preko pointera na Geometrijska_Figura : "
<< pokazivac_na_geometrijsku_figuru_1 -> Povrsina()
<< endl;
return 0;}
```



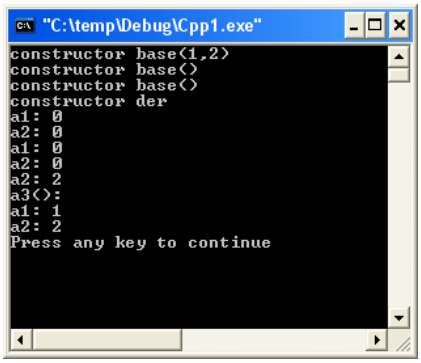
Ključna reč virtual ispred funkcije u osnovnoj klasi saopštava kompajleru da postoji odgovarajuća funkcija u izvedenoj klasi i da nju treba izvršiti prilikom poziva objekta izvedene klase. Ukoliko se ne navede ključna reč virtual, program će izvršiti funkciju osnovne klase, jer kompajler neće imati informaciju o postojanju odgovarajuće funkcije u izvedenoj klasi.

8 Zadatak:

Primer konstruktora osnovne klase

```
#include <iostream.h>
class base
{private: int a1;
protected: int a2;
public:
base()
{cout << "constructor base() \n";
a1 = 0;
a2 = 0; }
base(int x, int y)
{cout << "constructor base(" << x << "," << y << ")\n";
a1 = x;
a2 = y;}
void a3()
{cout << "a1: " << a1 << endl
<< "a2: " << a2 << endl;}
};
class der : public base
{private: base d1;
protected: base d2;
public:
der(int x, int y) : base(x, y)
{cout << "constructor der\n"; }
void d3()
{d1.a3() ;
d2.a3 () ;
cout << "a2: " << a2 << endl;
cout << "a3():" << endl;
a3();}
} ;
```

```
void main()
{der x(1, 2);
x.d3() ; }
```



9 Zadatak:

Primer destruktora osnovne klase

Program 10.2 prikazuje primer klase sa destruktorom za znakovne nizove. Pošto su znakovni nizovi međusobno vrlo različitih dužina, objekti za njihovo predstavljanje obično sadrže samo pokazivač na sam tekst. To je učinjeno i u ovom primeru.

Jedini atribut klase String je pokazivač na sam niz znakova. Zbog toga se prilikom inicijajizacije (konstruktorom String (char*)) vrši dodela memorijskog prostora potrebne veličine i kopiranje niza znakova u tako dodeljenu memoriju. Zadatak destruktora (~String( ) je da oslobodi taj memorijski prostor prilikom utavanja objekta.

Podrazumevani konstruktor (String ()) osigurava da svaki objekat tipa String bude inicijalizovan do te mere da bi kasnije smelo da se na njega primeni destruktor. Naime, posledice primene operatora delete na pokazivač slučajnog sadržaja su nepredvidljive, ali primena operatora delete na pokaziva ~ NULL (0) je uvek bezbedna.

Naredba niz=0; u destruktoru nije potrebna ako se destruktor poziva samo implicitno prilikom utavanja objekata. Me|utim, poželjno je da objekat bude ostavljen u " ispravnom praznom " stanju zbog eventualnog eksplicitnog pozivanja destruktora. Tada će objekat, verovatno, i dalje postojati pa bi moglo da zasmeta ako pokazivački atribut pokazuje na nešto u dinamičkoj zoni memorije što više ne postoji.

Konstruktor kopije (String(String&)) samo treba da prekopira celokupan sadržaj svog pravog argumenta u skriveni argument (to je argument koji se inicijalizuje). Pošto se vrši stvaranje novog objekta, ne pretpostavlja se ta o zatečenom sadržaju parčeta memorije koje se inicijalizuje.

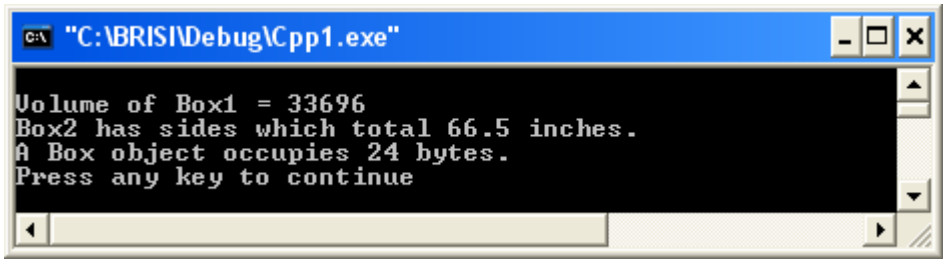
U glavnom programu na kraju programa, prvo se stvara objekat pozdrav tipa String koji se inicijalizuje običnim znakovnim nizom (char*). Tu će se pozivati konstruktor String(char*) koji je, u stvari, konverzija iz tipa char* u tip String. Posle toga, objekat a tipa String se inicijalizuje kopijom sadržaja objekta pozdrav pomoću konstruktora kopije String (String&) , dok objekat b se podrazumevanim konstruktorom String ( ) inicijalizuje kao "prazan" objekat. Ono što je bitno, atribut niz u njemu se postavlja na nulu, pa implicitno pozivanje destruktora (b.~String()) na kraju programa, neće da napravi nikakvu štetu.

10 Zadatak:

Kreiranje klase Box I postavljanje vrednosti njenih clanova:

```
#include <iostream.h>
class Box            // Class definition at global scope
{
  public:
    double length;   // Length of a box in inches
    double breadth;  // Breadth of a box in inches
    double height;   // Height of a box in inches
};

int main(void)
{
  Box Box1;              // Declare Box1 of type Box
  Box Box2;              // Declare Box2 of type Box
  double volume = 0.0;     // Store the volume of a box here
  Box1.height = 18.0;      // Define the values
  Box1.length = 78.0;      // of the members of
  Box1.breadth = 24.0;     // the object Box1
  Box2.height = Box1.height - 10;  // Define Box2
  Box2.length = Box1.length/2.0;   // members in
  Box2.breadth = 0.25*Box1.length; // terms of Box1

// Calculate volume of Box1
  volume = Box1.height * Box1.length * Box1.breadth;
  cout << endl
       << "Volume of Box1 = " << volume;
  cout << endl
       << "Box2 has sides which total "
       << Box2.height + Box2.length + Box2.breadth
       << " inches.";
  cout << endl      // Display the size of a box in memory
       << "A Box object occupies "
       << sizeof Box1 << " bytes.";
  cout <<endl;

  return 0;
}
```

IZLAZ:



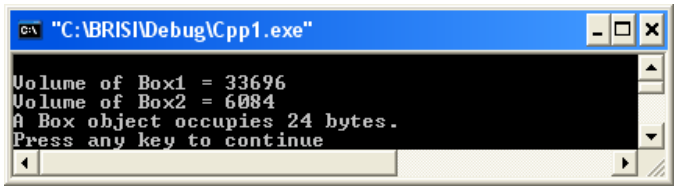
11 Zadatak:

PRIMER kako pristupati clanovima klase I kako klasi Box pridruziti funkciju.

// Izracunavanje zapremine koriscenjem funkcije clasnice klase

```
#include <iostream.h>

class Box               // Class definition at global scope
{
public:
  double length;        // Length of a box in inches
  double breadth;       // Breadth of a box in inches
  double height;        // Height of a box in inches
  // Function to calculate the volume of a box
  double Volume(void)
  {
    return length * breadth * height;
  }
};

int main(void)
{
  Box Box1;             // Declare Box1 of type Box
  Box Box2;             // Declare Box2 of type Box
  double volume = 0.0;  // Store the volume of a box here
  Box1.height = 18.0;   // Define the values
  Box1.length = 78.0;   // of the members of
  Box1.breadth = 24.0;  // the object Box1
  Box2.height = Box1.height - 10;  // Define Box2
  Box2.length = Box1.length/2.0;   // members in
  Box2.breadth = 0.25*Box1.length; // terms of Box1
  volume = Box1.Volume();          // Calculate volume of Box1
  cout << endl
       << "Volume of Box1 = " << volume;
  cout << endl
       << "Volume of Box2 = "
       << Box2.Volume();
  cout << endl
       << "A Box object occupies "
       << sizeof Box1 << " bytes.";
  cout << endl;

  return 0;
}
```

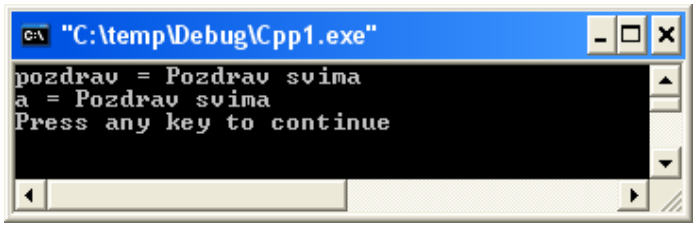
IZLAZ:



12 Zadatak:

Klasa String sa konstruktorima i destruktorima

```
#include <iostream.h>
class String {
char *niz; // Pokazivac na sam tekst.
public:
String () { niz = 0; } // Inicijalizacija praznog niza.
String (const char *) ; // Inicijalizacija nizom znakova.
String (const String &) ; // Inicijalizacija tipom String.
~String () ; // Unistavanje objekta tipa String.
void pisi () const; // Ispisivanje niza.
};
#include <string.h>
#include <iostream.h>
String:: String (const char *t)
{ if ( (niz = new char [strlen(t) +1] ) != 0) strcpy(niz,t) ; }
String::String (const String &s)
{if ((niz = new char [strlen(s.niz)+1]) != 0) strcpy (niz, s.niz); }
String::~String ()
{ delete [] niz; niz = 0; }
void String::pisi () const
{ cout << niz; }
int main () {
String pozdrav ("Pozdrav svima") ; // Poziva se String (char *)
String a = pozdrav; // Poziva se String (String &).
String b; // Poziva se String ().
cout << "pozdrav = "; pozdrav.pisi (); cout << endl;
cout << "a = "; a.pisi () ; cout << endl;
return 0;
}
// Ovde se poziva destruktor za sva tri objekta.
```



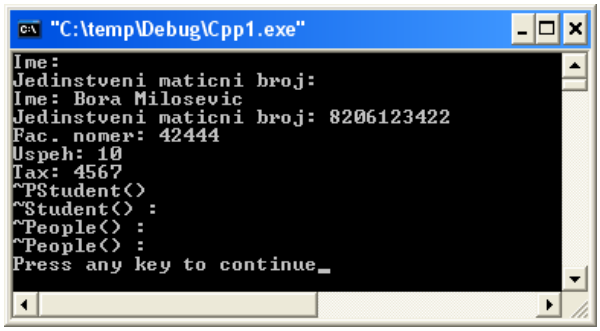
13 Zadatak:

U definisanim klasama People, Student i PStudent sagledati upotrebu konstruktora i destruktora.

```
#include <iostream.h>
#include <string.h>
//deklaracija klase People
```

```
class People
{public:
People(char * = "", char * = "");
void PrintPeople() const;
~People();
private:
char * name;
char * egn;
};
//definicija konstruktora klase People
People::People(char *str, char *num)
{name = new char[strlen(str)+1];
strcpy(name, str);
egn = new char[11];
strcpy(egn, num); }
//definicija metode PrintPeople
void People::PrintPeople() const
{cout << "Ime: " << name << endl;
cout << "Jedinstveni maticni broj: " << egn << endl; }
//definicija destruktora klase People
People::~People()
{cout << "~People() : " << endl;
delete name;
delete egn; }
//deklaracija klase Student
class Student : public People
{ public:
Student(char * = "", char * = "", long = 0, double = 0);
void PrintStudent() const;
~Student()
{cout << "~Student() : " << endl; }
private:
long facnom;
double usp; };
//definicija konstruktora klase Student
Student::Student(char *str, char * num, long facn,
double u) : People(str, num)
{facnom = facn;
usp = u; }
//definicija metode PrintStudent
void Student::PrintStudent() const
{PrintPeople();
cout << "Fac. nomer: " << facnom << endl;
cout << "Uspeh: " << usp << endl; }
// deklaracija klase PStudent
class PStudent : public Student
{ public:
PStudent(char * = "", char * = "", long = 0,
double = 0, double = 0);
~PStudent()
{cout << "~PStudent() \n"; }
void PrintPStudent() const;
protected:
```

```
double tax; } ;
//definicija konstruktora klase PStudent
PStudent::PStudent(char *str, char *num, long facn,
double u, double t) : Student(str, num, facn, u)
{tax = t; }
//definicija metode PrintPStudent
void PStudent::PrintPStudent() const
{PrintStudent();
cout << "Tax: " << tax << endl; }
void main()
{People pe;
pe.PrintPeople();
PStudent PStud("Bora Milosevic", "8206123422", 42444, 10.0, 4567);
PStud.PrintPStudent();
}
```

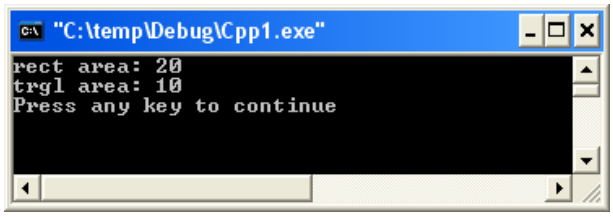


14 Zadatak:

PRIMER klase CPolygon i naslednih klasa CRectangle I CTriangle

```
#include<iostream.h>
class CPolygon {
protected:
int width,height;
public:
void set_values (int a,int b)
{ width=a;height=b; }
};
class CRectangle: public CPolygon { //klasa naslednik:public osnovna klasa
public:
int area (void)
{ return (width * height); }};
class CTriangle: public CPolygon { //klasa naslednik:public osnovna klasa
public:
int area (void)
{ return (width * height/2); }};
main() {
    CRectangle rect;//instanca, novi objekat
    CTriangle trgl; //instanca, novi objekat
    rect.set_values(4,5);
    trgl.set_values(4,5);
```

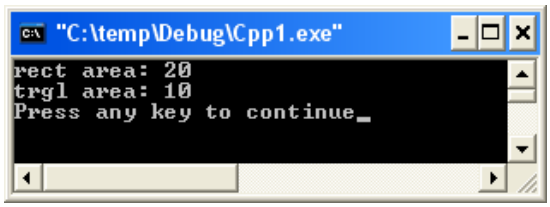
```
        cout<< "rect area: "<<rect.area()<<endl;
        cout<< "trgl area: "<<trgl.area()<<endl;
    return 0;
    }
```



15 Zadatak:

PRIMER - Polimorfizam:

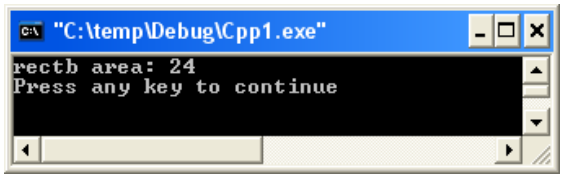
```
#include<iostream.h>
class CPolygon {
protected:
int width,height;
public:
void set_values (int a,int b)
{ width=a;height=b; }
};
class CRectangle: public CPolygon { //klasa naslednik:public osnovna klasa
public:
int area (void)
{ return (width * height); }};
class CTriangle: public CPolygon { //klasa naslednik:public osnovna klasa
public:
int area (void)
{ return (width * height/2); }};

main() {      CRectangle rect;
    CTriangle trgl;
    CPolygon *ppoly1=&rect;
    CPolygon *ppoly2=&trgl;
    ppoly1->set_values(4,5);
    ppoly2->set_values(4,5);
    cout<< "rect area: "<<rect.area()<<endl;
    cout<< "trgl area: "<<trgl.area()<<endl;
return 0;}
```



16 Zadatak:

PRIMER Prijateljske funkcije:

```
//PRIMER klase CRectangle sa uvodjenjem prijateljske funkcije
#include<iostream.h>
class CRectangle {
int width,height;
public:
void set_values (int,int);  //konstruktor kao funkcija
int area (void) { return (width*height); }//f-ja clanica klase
friend CRectangle duplicate (CRectangle);
};
void CRectangle::set_values (int a, int b)
{ width=a; height=b; }
CRectangle duplicate (CRectangle rectparam)
{CRectangle rectres;
rectres.width=rectparam.width*2;
rectres.height=rectparam.height*2;
return (rectres); }

main(){
      CRectangle recta,rectb;//novi objekat, instanca klase
      recta.set_values(2,3);
      rectb=duplicate (recta);
      cout<< "rectb area: "<<rectb.area()<<endl;
}
```

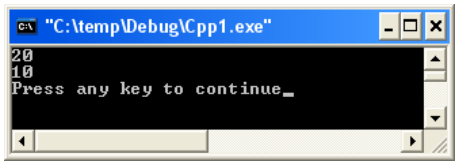


16 Zadatak:

PRIMER Virtualne funkcije:

```
#include<iostream.h>
class CPolygon
{protected:
 int width,height;
 public:
 void set_values (int a,int b)
{ width=a;height=b; }
 virtual int area (void) =0;};
class CRectangle: public CPolygon //klasa naslednik:public osnovna klasa
{public:
int area (void)
{ return (width * height); }};
class CTriangle: public CPolygon { //klasa naslednik:public osnovna klasa
```

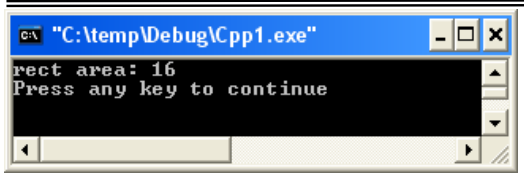
```
public:
int area (void)
{ return (width * height/2); }};

main() {
      CRectangle rect;
      CTriangle trgl;
      CPolygon *ppoly1=&rect;
      CPolygon *ppoly2=&trgl;
      ppoly1->set_values(4,5);
      ppoly2->set_values(4,5);
      cout<< ppoly1->area()<<endl;
      cout<< ppoly2->area()<<endl;
return 0;}
```



17 Zadatak:

PRIMER 13 – Prijateljske klase:

```
//PRIMER klase CRectangle sa uvodjenjem prijateljske klase CSquare
#include<iostream.h>
class CSquare;
class CRectangle
{int width,height;
public:
int area (void) {return (width*height);}    //f-ja clanica klase
void convert (CSquare a);};
class CSquare
{private:
int side;
public:
void set_side (int a)
{ side=a; }
friend class CRectangle; };
void CRectangle::convert (CSquare a)
{ width=a.side;  height=a.side; }

main(){
CSquare sqr;
CRectangle rect;
      sqr.set_side(4);
      rect.convert(sqr);
      cout<< "rect area: "<<rect.area()<<endl;
return 0;}
```



18 Zadatak:

Demonstracija deklaracije klase i definicija objekata

```
#include <iostream>
#include <string>
using namespace std;

class person
{
public:
  string name;
  int age;
};

int main ()
{
  person a, b;
  a.name = "Bora";
  b.name = "Srdjan";
  a.age = 60;
  b.age = 40;
  cout << a.name << ": " << a.age << endl;
  cout << b.name << ": " << b.age << endl;
  return 0;
}
```

Izlaz