Elektrotehnička škola Tuzla

Informatika 2. razred

Računari i programiranje 3. razred

PROGRAMSKI JEZIK

C++

Skripta za internu upotrebu

Verzija: 3.1

Zadnja revizija: 28.3.2007-

Autori:

Selma Krajinović, dipl. ing el.

Edin Imširović, dipl. ing. el

1. PROGRAMSKI JEZICI

Prvi računari bila su vrlo složeni za korištenje. Njih su koristili isključivo stručnjaci koji su bili osposobljeni za komunikaciju s računarom. Ta komunikacija se sastojala od dva osnovna koraka: davanje instrukcija računaru i čitanje rezultata obrade. I dok se čitanje rezultata vrlo brzo učinilo koliko-toliko snošljivim uvođenjem štampača na kojima su se rezultati ispisivali, unošenje instrukcija - programiranje - se sastojalo od mukotrpnog unosa niza nula i jedinica. Ti nizovi su davali računaru upute kao što su: "saberi dva broja", "premjesti podatak s neke memorijske lokacije na drugu", "skoči na neku instrukciju izvan normalnog slijeda instrukcija" i slično. Programski jezici koji su omogućavali ovu komunikaciju nazivaju se ***mašinski programski jezici***. Kako je takve programe bilo vrlo složeno pisati, a još složenije čitati i ispravljati, ubrzo su se pojavili prvi programerski alati nazvani asembleri (engl. *assemblers*).

U ***asemblerskom*** jeziku svaka mašinska instrukcija predstavljena je mnemonikom koji je razumljiv ljudima koji čitaju program. Tako se sabiranje npr. obavlja mnemonikom ADD, dok se premještanje podataka obavlja mnemonikom MOV. Time se postigla bolja čitljivost programa, no i dalje je bilo vrlo složeno pisati programe i ispravljati ih jer je bilo potrebno davati sve, pa i najmanje instrukcije računaru za svaku pojedinu operaciju. Javlja se problem koji će kasnije, nakon niza godina, dovesti i do pojave C++ programskog jezika: potrebno je razviti programerski alat koji će osloboditi programera rutinskih poslova te mu dopustiti da se usredotoči na problem koji rješava.

Zbog toga se pojavljuje niz ***viših programska jezika***, koji preuzimaju na sebe neke "dosadne" programerske poslove. Tako je FORTRAN bio posebno pogodan za matematičke proračune, zatim BASIC koji se vrlo brzo učio, te COBOL koji je bio u pravilu namijenjen upravljanju bazama podataka.

Oko 1972. se pojavljuje jezik C, koji je direktna preteča današnjeg jezika C++. To je bio prvi jezik opšte namjene te je postigao neviđen uspjeh. Više je razloga tome: jezik je bio jednostavan za učenje, omogućavao je modularno pisanje programa, sadržavao je samo naredbe koje se mogu jednostavno prevesti u mašinski jezik, davao je brzi kôd. Jezik je omogućavao vrlo dobru kontrolu mašinskih resursa te je na taj način omogućio programerima da optimiziraju svoj kôd. Do unatrag nekoliko godina, 99% komercijalnih programa bili su pisani u C-u, ponegdje dopunjeni odsječcima u mašinskom jeziku kako bi se kritični dijelovi učinili dovoljno brzima.

No kako je razvoj programske podrške napredovao, stvari su se i na području programskih jezika počele mijenjati. Složeni projekti od nekoliko stotina hiljada, pa i više redova više nisu rijetkost, pa je zbog toga bilo potrebno uvesti dodatne mehanizme kojima bi se takvi programi učinili jednostavnijima za izradu i održavanje, te kojima bi se omogućilo da se jednom napisani kôd iskoristi u više različitih projekata.

1.2. Proceduralno i objektno orjentisano programiranje

Proceduralno programiranje se zasniva na posmatranju programa kao niza jednostavnih programskih cjelina: procedura. Svaka procedura je konstruisana tako da obavlja jedan manji zadatak, a cijeli se program sastoji od niza procedura koje međusobno sudjeluju u rješavanju zadatka.

Princip kojim bismo mogli obilježiti proceduralno strukturirani model jest *podijeli-pa-vladaj*: cjelokupni program je presložen da bi ga se moglo razumjeti pa se zbog toga on rastavlja na niz manjih zadataka - procedura - koje su dovoljno jednostavne da bi se mogle izraziti pomoću naredbi programskog jezika. Pri tome, pojedina procedura također ne mora biti riješena monolitno: ona može svoj posao obaviti kombinirajući rad niza drugih procedura.

Npr. pretpostavimo da treba napisati program za nalaženje korijena kvadratne jednačine. Ovaj problem mogli bismo riješiti u slijedećim koracima:

1. Unesi vrijednosti a,b,c
2. Izračunaj diskriminantu

3. U ovisnosti o vrijednosti diskriminante, izracunaj korijene

4. Odštampaj korijene

Ovakav programski pristup je bio vrlo uspješan do kasnih osamdesetih, kada su njegovi nedostaci postajali sve očitiji. Pokazalo se složenim istodobno razmišljati o problemu i odmah strukturirati rješenje. Umjesto rješavanja problema, programeri su mnogo vremena provodili pronalazeći načine da programe usklade sa zadanom strukturom. Također, današnji programi se pokreću pomoću miša, prozora, menija i dijaloga. Programiranje je *vođeno događajima* (engl. *event-driven*) za razliku od starog, sekvencijalnog načina. Proceduralni programi su korisniku u pojedinom trenutku prikazivali niz ekrana nudeći mu pojedine opcije u određenom redoslijedu. No vođeno *događajima* znači da se program ne odvija po unaprijed određenom slijedu, već se programom upravlja pomoću niza događaja. Događaja ima raznih: pomicanje miša, pritisak na tipku, izbor stavke iz menija i slično. Sada su sve opcije dostupne istovremeno, a program postaje interaktivan, što znači da promptno odgovara na korisnikove zahtjeve i odmah (ovo ipak treba uvjetno shvatiti) prikazuje rezultat svoje akcije na monitoru računara.

Kako bi se takvi zahtjevi jednostavnije proveli u praksi, razvijen je ***objektni pristup programiranju***. Osnovna ideja je razbiti program u niz zatvorenih cjelina koje zatim međusobno sarađuju u rješavanju problema. Umjesto specijaliziranih procedura koje barataju podacima, radimo s objektima koji objedinjavaju i operacije i podatke. Pri tome je važno ***šta objekt radi***, a ***ne kako*** on to radi. Jedan objekt se može izbaciti i zamijeniti drugim, boljim, ako oba rade istu stvar.

1.3. Program

Sam računar, čak i kada se uključi u struju, nije kadar učiniti ništa korisno. Ono što vam nedostaje jest pamet neophodna za koristan rad računara: programi. Pod programom podrazumijevamo niz naredbi u mašinskom jeziku koje procesor u vašem računaru izvodi i shodno njima obrađuje podatke, provodi matematičke proračune, ispisuje tekstove, iscrtava krivulje, itd. Pokretanjem programa s diska, diskete ili CD-ROM-a, program se učitava u radnu memoriju računara i procesor počinje s mukotrpnim postupkom njegova izvođenja.

Programi koje pokrećete na računaru su u ***izvršnom obliku*** (engl. *executable*), razumljivom samo procesoru vašeg (i njemu sličnih) računara. U suštini se mašinski kôd sastoji od nizova binarnih cifara: nula i jedinica. Budući da su današnji programi tipično dužine nekoliko megabajta, naslućujete da ih autori nisu pisali izravno u mašinskom kôdu.

Gotovo svi današnji programi se pišu u nekom od ***viših programskih jezika*** (FORTRAN, BASIC, Pascal, C) koji su donekle razumljivi i ljudima. Naredbe u tim jezicima se sastoje od mnemonika. Kombinovanjme tih naredbi programer slaže ***izvorni kôd*** (engl. *source code*) programa, koji se pomoću posebnih programa ***kompajlera*** (engl. *compiler*) i ***povezivača*** (engl. *linker)* prevodi u izvršni kôd. Prema tome, pisanje programa u užem smislu podrazumijeva pisanje izvornog kôda. Međutim, kako pisanje kôda nije samo sebi svrhom, pod pisanjem programa u širem smislu podrazumijeva se i prevođenje, odnosno povezivanje programa u izvr [ni kôd. Stoga možemo govoriti o četiri faze izrade programa:

1. **pisanje izvornog kôda**

2. **prevođenje izvornog kôda,**

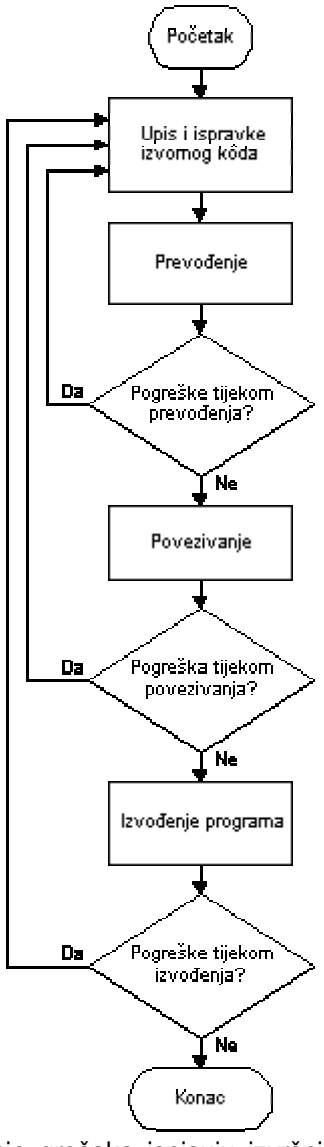
3. **povezivanje u izvršni kôd te**

4. **testiranje programa.**

Prva faza programa je pisanje izvornog kôda. U principu se izvorni kôd može pisati u bilo kojem programu za uređivanje teksta (engl. *text editor*), međutim velika većina današnjih kompajlera i povezivača se isporučuje kao cjelina zajedno s ugrađenim programom za upis i ispravljanje izvornog kôda. Te programske cjeline poznatije su pod nazivom *integrisane razvojne okoline* (engl. *integrated development environment, IDE*). Nakon što je pisanje izvornog kôda završeno, on se pohrani u datoteku izvornog kôda na disku. Toj datoteci se obično daje nekakvo smisleno ime, pri čemu se ono za kôdove pisane u programskom jeziku

C++ obično proširuje nastavkom .cpp, .cp ili samo .c, na primjer pero.cpp. Nastavak je potreban samo zato da bismo izvorni kôd kasnije mogli lakše pronaći.

Slijedi prevođenje izvornog kôda. U integrisanim razvojnim okolinama program za prevođenje se pokreće pritiskom na neku tipku na ekranu, pritiskom odgovarajuće tipke na tastaturi ili iz nekog od *menija* (engl. *menu*) - ako kompajler nije integrsan, poziv je nešto složeniji. Kompajler tokom prevođenja *provjerava sintaksu napisanog izvornog kôda i u slučaju uočenih ili naslućenih grešaka ispisuje odgovarajuće poruke o greškama*, odnosno upozorenja. Greške koje prijavi kompajler nazivaju se *greškama pri prevođenju* (engl. *compile-time errors*). Nakon toga programer će pokušati ispraviti sve navedene greške i ponovo prevesti izvorni kôd - sve dok prevođenje kôda ne bude uspješno okončano, neće se moći pristupiti povezivanju kôda. Prevođenjem izvornog dobiva se datoteka ***objektnog kôda*** (engl. *object code*), koja se lako može prepoznati po tome što obično ima nastavak .o ili .obj (u našem primjeru bi to bio pero.obj).



u Nakon što su ispravljene sve greške uočene prilikom prevođenja i kôd ispravno preveden, pristupa se ***povezivanju objektnih kôdova izvršni kod***. U većini slučajeva objektni kôd dobiven prevođenjem programerovog izvornog kôda treba povezati s postojećim ***bibliotekama*** (engl. *libraries*). Biblioteke su datoteke u kojima se nalaze već prevedene gotove funkcije ili podaci. One se isporučuju zajedno s kompajlerem, mogu se zasebno kupiti ili ih programer može tokom rada sam razvijati. Bibliotekama se izbjegava ponovno pisanje vrlo često korištenih operacija. Tipičan primjer za to je biblioteka matematičkih funkcija koja se redovno isporučuje uz kompajlere, a u kojoj su definisane sve funkcije poput trigonometrijskih, hiperbolnih, eksponencijalnih i sl. Prilikom povezivanja provjerava se mogu li se svi pozivi kôdova realizovati u izvršnom kôdu. Uoči li povezivač neku nepravilnost tokom povezivanja, ispisat će poruku o grešci i onemogućiti generisanje izvornog kôda. Ove greške nazivaju se ***greškama pri povezivanju*** (engl. *link-time errors*) - sada programer mora prionuti ispravljanju grešaka koje su nastale pri povezivanju. Nakon što se isprave sve greške, kôd treba ponovno prevesti i povezati.

Uspješnim povezivanjem dobiva se ***izvršni kôd***. Međutim, takav izvršni kôd još uvijek ne garantuje da će program raditi ono što ste zamislili. Na primjer, može se dogoditi da program radi pravilno za neke podatke, ali da se za druge podatke ponaša nepredvidivo. U tom se slučaju radi o ***greškama pri izvođenju*** (engl. *run-time errors*). Da bi program bio potpuno korektan, programer treba istestirati program da bi uočio i ispravio te greške, što znači ponavljanje cijelog postupka u lancu –

“i spravljanje izvornog kôda-prevođenje-povezivanje-testiranje'.

Za ispravljanje grešaka pri izvođenju, programeru na raspolaganju stoje ***programi za otkrivanje grešaka*** (engl. *debugger*). Radi se o programima koji omogućavaju prekid izvođenja izvedbenog kôda programa koji testiramo na unaprijed zadanim naredbama, izvođenje programa naredbu po naredbu, ispis i promjene trenutnih vrijednosti pojedinih podataka u programu. Najjednostavniji programi za otkrivanje grešaka ispisuju izvršni kôd u obliku mašinskih naredbi. Međutim, većina današnjih naprednih programa za otkrivanje grešaka su *simbolički* (engl. *symbolic debugger*) - iako se izvodi prevedeni, mašinski kôd, izvođenje programa se prati preko izvornog kôda pisanog u višem programskom jeziku. To omogućava vrlo lagano lociranje i ispravljanje grešaka u programu.

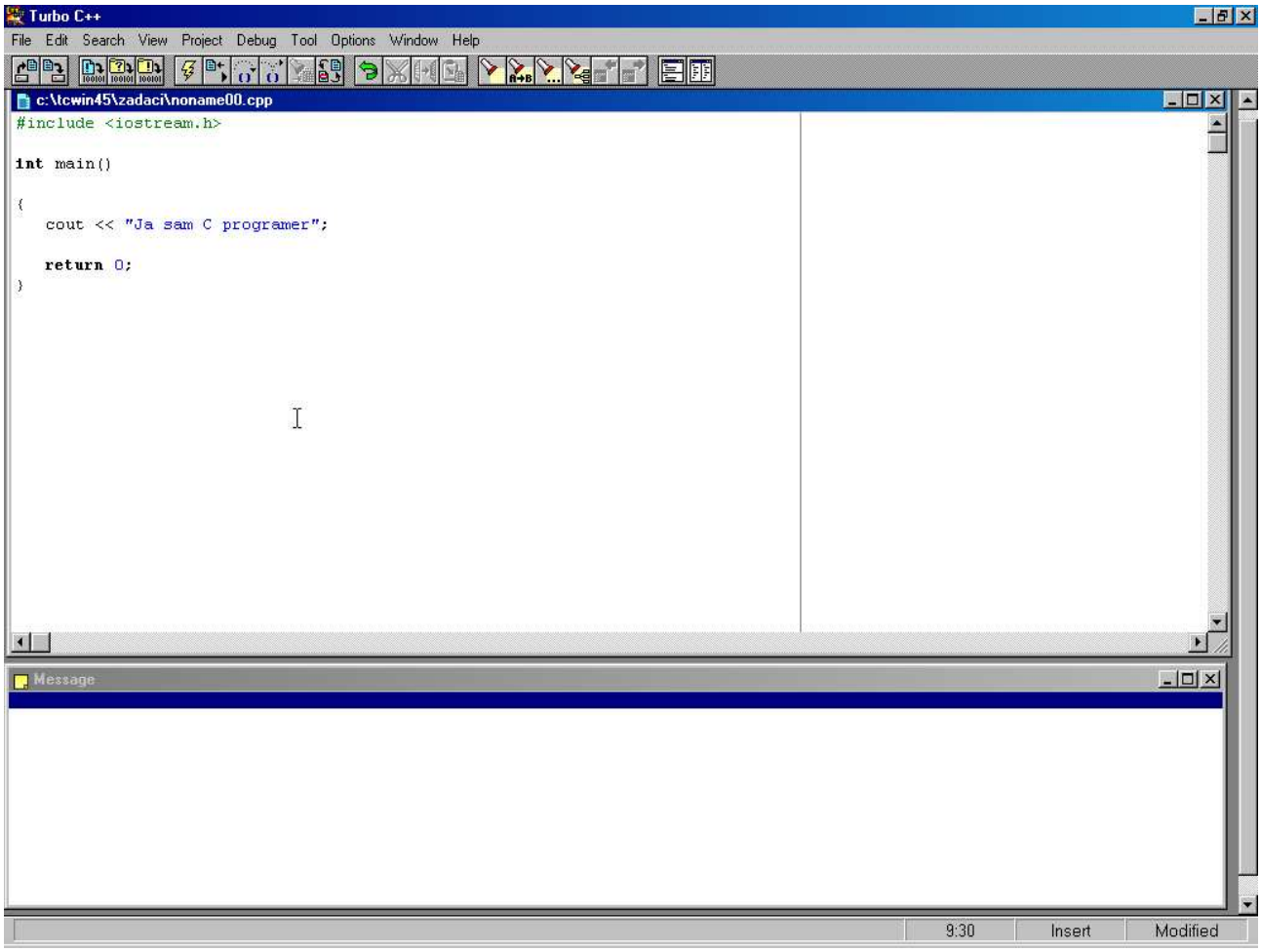
Osim grešaka, kompajler i linker redovno javljaju i ***upozorenja***. Ona ne onemogućavaju nastavak prevođenja, odnosno povezivanja kôda, ali predstavljaju potencijalnu opasnost. Upozorenja se mogu podijeliti u dvije grupe. Prvu grupu čine upozorenja koja javljaju da kôd nije potpuno korektan. Kompajler ili povezivač će zanemariti našu pogrešku i prema svom nahođenju izgenerisati kôd. Drugu grupu čine poruke koje upozoravaju da "nisu sigurni je li ono što smo napisali upravo ono što smo željeli napisati", tj. radi se o dobronamjernim upozorenjima na zamke koje mogu proizići iz načina na koji smo program napisali. Iako će, unatoč upozorenjima, program biti preveden i povezan (možda čak i korektno), pedantan programer neće ta

upozorenja nikada zanemariti - ona često upućuju na uzrok grešaka pri izvođenju gotovog programa. Za precizno tumačenje poruka o greškama i upozorenja neophodna je dokumentacija koja se isporučuje uz kompajler i povezivač.

2. RAZVOJNO OKRUŽENJE BORLAND TURBO C++

2.1.Korištenje razvojnog okruženja Borland C++

Razvojno okruženje koje će biti korišteno u ovom kursu jeste Borland Turbo C++ verzija 4.5.



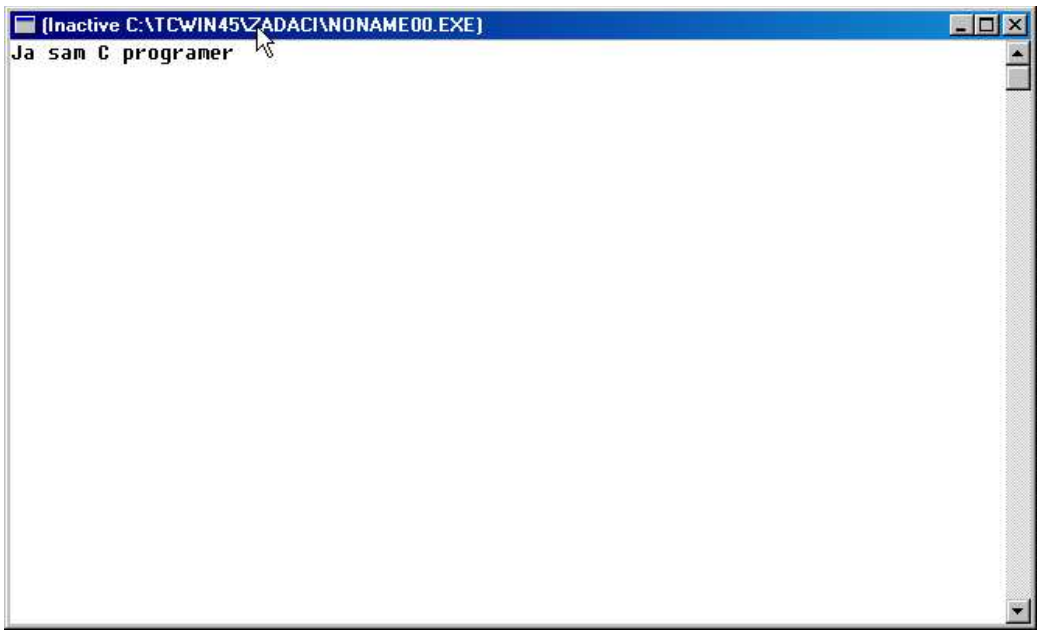
Razvojno okruženje obuhvata:

1. meni komandi
2. text editor za unos izvornog koda (prozor c:\win45\zadaci\noname00.cpp),
3. prozor za ispis poruka o grešakama (Message).

Nakon unošenja izvornog koda napisanog u programskom jeziku C++, program je potrebno prevesti (kompajlirati) pomoću komande ***Project, Compile***, iz menija komandi. Ukoliko je došlo do grešaka, u prozoru Message pojavit će se poruke o greškama prevođenja.

Većina integrisanih razvojnih okruženja kao što je i Borlandov C++ omogućava integrisano prevođenje i povezivanje programa sa bibliotekama funkcija. Ovo se pokreće pomoću komande ***Project, Build All***. Ovom komandom dakle, pozivamo najprije kompajler, a zatim i linker.

Startovanje programa realizuje se komadom ***Debug, Run***. Nakon toga, pojavit će se novi prozor u kome je pozvana izvršna verzija programa.



C++ izvorni kod može se snimiti kao i svaka druga datoteka pomoću komadni iz menija File.

Datoteka koja sadrži izvorni kod automatski dobije ekstenziju .cpp, objektne datoteke imaju ekstenziju .obj, a izvršne ekstenziju .exe.

2.2. Prvi program

U prethodnom poglavlju korišten je jednostavan program koji ispisuje tekst na ekranu:

```
#include <iostream.h>
int main()
{
        cout << "Ja sam C programer";
        return 0;
}
```

Slijedi objašnjenje pojedinih linija koda.

U drugom redu je napisano int main(). main je naziv za glavnu funkciju koju posjeduje svaki C++ program - izvođenje svakog programa počinje naredbama koje se nalaze u ovoj funkciji. Riječ int ispred oznake glavne funkcije ukazuje na to da će main() po završetku izvođenja naredbi i funkcija sadržanih u njoj kao rezultat tog izvođenja vratiti cijeli broj (int dolazi od engleske riječi *integer* koja znači *cijeli broj*).

Slijedi otvorena ***vitičasta zagrada***. Ona označava početak *bloka* u kojem će se nalaziti naredbe glavne funkcije, dok zatvorena vitičasta zagrada u zadnjem redu označava kraj tog bloka. Na kraju bloka nalazi se naredba ***return 0***. Tom naredbom glavni program vraća pozivnom programu broj 0, a to je poruka operacionom sistemu da je program uspješno okončan.

Uočimo ***znak ;*** (tačka-zarez) iza naredbi! On označava kraj naredbe te služi kao poruka kompajleru da sve znakove koji slijede interpretira kao novu naredbu.

U prvom redu nalazi se naredba ***#include <iostream.h>*** kojom se od kompajlera zahtijeva da u naš program uključi biblioteku *iostream*. U toj biblioteci nalazi se *izlazni tok* (engl. *output stream*) tj. funkcije koje omogućavaju ispis podataka na ekranu. Ta biblioteka nam je neophodna da bismo u prvom redu glavnoga programa ispisali tekst poruke. Naglasimo da #include nije naredba C++ jezika, nego se radi o *pretprocesorskoj naredbi*. Kada kompajler dođe do nje, on će prekinuti postupak prevođenja kôda u tekućoj

datoteci, skočiti u datoteku iostream.h, prevesti je, a potom se vratiti u početnu datoteku na red iza naredbe #include. ***Sve pretprocesorske naredbe počinju znakom #.***

iostream.h je primjer *datoteke zaglavlja* (engl. *header file*, odakle i slijedi nastavak .h). U takvim datotekama se nalaze deklaracije funkcija sadržanih u odgovarajućim bibliotekama. Jedna od osnovnih osobina jezika C++ jest vrlo oskudan broj funkcija ugrađenih u sam jezik. Ta oskudnost olakšava učenje samog jezika, te bitno pojednostavnjuje i ubrzava postupak prevođenja. Za specifične zahtjeve na raspolaganju je veliki broj odgovarajućih biblioteka funkcija i klasa.

cout je ime izlaznog toka definisanog u biblioteci iostream, pridruženog ekranu računara. Operatorom << (dva znaka "manje od") podatak koji slijedi upućuje se na izlazni tok, tj. na ekran računara. U gornjem primjeru to je kratka tekstualna poruka:

Ja sam C programer

3. STRUKTURA PROGRAMA

Osnovni elementi programa u C++ programskom jeziku su:

1. komentari
2. deklaracija promjenljivih (varijabli)
3. funkcija main

```
/*****************************************************
* komentari zaglavlja programa                       *
*****************************************************/

… deklaracija promjenljivih

int main()
{
 …. izvršne naredbe….
                                                  return 0;
}
```

3.1. Komentari

Kada kompajler naleti na ***dvostruku kosu crtu***, on će zanemariti sav tekst koji slijedi do kraja tekućeg reda i prevođenje će nastaviti u sljedećem redu. Komentari dakle ne ulaze u izvršni kôd programa i služe programerima za ***opis značenja pojedinih naredbi ili dijelova kôda***. Komentari se mogu pisati u zasebnom redu ili redovima, što se obično odnosi za duže opise, na primjer šta određeni program ili funkcija radi. Za kraće komentare, na primjer za opise promjenljivih ili nekih operacija, komentar se piše u nastavku naredbe.

```
// izracunavanje vrijednosti diskriminante
```

Uz gore navedeni oblik komentara, jezik C++ podržava i komentare unutar para znakova */* *****/***. Takvi komentari započinju slijedom /* (kosa crta i zvjezdica), a završavaju slijedom */ (zvjezdica i kosa crta). Kraj reda ne znači podrazumijevani završetak komentara, pa se ovakvi komentari mogu protezati na nekoliko redova izvornog kôda, a da se pritom znak za komentar ne mora ponavljati u svakom redu:

```
/*                    Ovakav način komentara
                      preuzet je iz programskog jezika C.
*/
```

Iako komentari iziskuju dodatno vrijeme i napor, u kompleksnijim programima oni se redovno isplate. Dogodi li se da neko drugi mora ispravljati vaš kôd, ili da nakon dugo vremena vi sami morate ispravljati svoj kôd, komentari će vam olakšati da razumijete kod. Svaki ozbiljniji programer ili onaj ko to želi postati mora biti svjestan da će nakon desetak ili stotinjak napisanih programa početi zaboravljati čemu pojedini program služi. Zato je vrlo korisno na početku datoteke izvornog programa u komentaru navesti osnovne "generalije", na primjer ime programa, ime datoteke izvornog kôda, kratki opis onoga što bi program trebao raditi, funkcije i klase definisane u datoteci te njihovo značenje, autor(i) kôda, uslovi pod kojima je program preveden (operativni sistem, ime i oznaka kompajlera), zabilješke, te naknadne izmjene, kao u primjeru u slijedećem poglavlju.

3.2. Deklaracija promjenljivih

C++ omogućava pohranjivanje vrijednosti u promjenljivim ili varijablama. Svaki program sadrži u sebi podatke koje obrađuje. Njih možemo podijeliti na

- nepromjenjive: ***konstante***,

- promjenjive: ***varijable ili promjenjive***.

Najjednostavniji primjer konstanti su brojevi (5, 10, 3.14159). Promjenljive su podaci koji mogu mijenjati svoj iznos. Stoga se oni u izvornom kôdu predstavljaju ne svojim iznosom već simboličkom oznakom, *imenom promjenljive*.

Svaki podatak ima dodijeljenu oznaku tipa, koja govori o tome kako se dotični podatak pohranjuje u memoriju računara, koji su njegovi dozvoljeni rasponi vrijednosti, kakve su operacije moguće s tim podatkom i sl. Tako razlikujemo cjelobrojne, realne, logičke, pokazivačke podatke. U poglavlju 4 upoznat ćemo se sa ugrađenim tipovima podataka i pripadajućim operatorima. Svaka promjenljiva koja će biti korištena u programu mora biti deklarisana. *Deklaracija promjenljivih* podrazumijeva određivanje identifikatora (imena) promjenljive i tipa podataka, tj. vrijednosti koje ta promjenljiva može poprimiti. U primjeru koji slijedi, promjenljive su deklarisane linijom:

```
int a, b, c;
```

Ovom naredbom deklarisane su tri promjenljive sa imenima a, b i c, te određeno da one mogu poprimiti cjelobrojne vrijednosti, tj. bilo koju vrijednost iz skupa cijelih brojeva (tip podataka int bit će detaljnije objašnjen u nastavku).

3.3. Funkcija main

Kao što je već spomenuto, ovo je osnovna funkcija svakog C++ programa koja označava dio programa koji treba izvršiti. Ona sadrži ***blok naredbi*** - jednu ili više naredbi koje su omeđene parom otvorena-zatvorena vitičasta zagrada {}. Izvana se taj blok ponaša kao jedinstvena cjelina, kao da se radi samo o jednoj naredbi. Blokovi naredbi se redovno pišu uvučeno. To uvlačenje radi se isključivo radi preglednosti, što je naročito važno ako imamo blokove ugniježđene jedan unutar drugog.

Naredbe određuju radnje koje je potrebno izvršiti da bi se ulazni podaci transformisali u izlazne, i na taj način riješio problem za koji je program napisan. C++ raspolaže određenim skupom naredbi koje ćemo u nastavku upoznati. Svaka naredba bloka mora završiti znakom ;.

3.4. Dodjela vrijednosti promjenljivim

Suštinski zadatak programa je da izvrši transformaciju ulaznih podataka u izlazne podatke, prema definisanom zadatku. Stoga je dodjela vrijednosti promjenljivih operacija koju neizostavno koristimo u programu.

Npr. naredba **a=4;** je naredba kojom se promjenljivoj a dodjeljuje vrijednost 4.

Operator = je ***operator dodjele***. Operatorom dodjele mijenja se vrijednost neke promjenljive. Najčešći operator dodjele je znak jednakosti (=) kojim se promjenljivoj na lijevoj strani pridružuje neka vrijednost s desne strane. Očito je da s lijeve strane operatora pridruživanja mogu biti isključivo promjenjive, pa se stoga ne može pisati ono što je inače matematički korektno:

```
2 = 4 / 2      // greška!!!
3.14159 = pi   // greška!
```

U primjeru program koji slijedi pojavljuje se naredba

```
c= a+b;
```

Ovom naredbom se promjenljivoj c dodjeljuje zbir vrijednosti promjenljivih a i b.

3.5. Naredbe cout i cin

U prvom programu napisanom u C++ programskom jeziku vidjeli smo naredbu **cout**, koja štampa vrijednosti

na ekranu. Korištenjem **cout** i ekrana (najčešćeg izlaznog uređaja) možete štampati informacije na način koji želite. Program postaje moćniji ako mu dodate i mogućnost učitavanja podataka sa tastature pomoću naredbe **cin**. Za razliku od **cout** koja šalje vrijednost na ekran, **cin** prihvata vrijednosti koje korisnik ukuca pomoću tastature. U narednim poglavljima biće pojašnjen način korištenja ove dvije naredbe.

Vježbe

Struktura tipičnog C++ programa ilustrovana je u slijedećem primjeru:

```
/*************************************************************
Program:     Moj drugi C++ program
Datoteka:    Drugi.cpp
Funkcije:    main() - cijeli program je u jednoj datoteci
Opis:        Učitava dva broja i ispisuje njihov zbir
Autori:      Bajro; Jusuf
Okruženje:   Pendžer
Kompajler: Borland C++ ver 4.5
Bilješke:  Iako zasad izgleda glupo, ovo je itekako vazan dio na putu da postanes programer
Izmjene:     15.07.03.       (jš) prva verzija
             21.01.04.       (bm) svi podaci su iz float    promijenjeni u int
*************************************************************/
#include <iostream.h>
int a, b, c;

int main() {
    cout << "Upisi prvi broj:";
    cin >> a;                  // očekuje prvi broj
    cout << "Upisi i drugi broj:";
    cin >> b;                  // očekuje drugi broj
    c = a + b;                 // računa njihov zbir
    // ispisuje rezultat:
    cout << "Njihov zbir je: " << c << endl;
    return 0;
}
```

4. STANDARNI ULAZ/IZLAZ

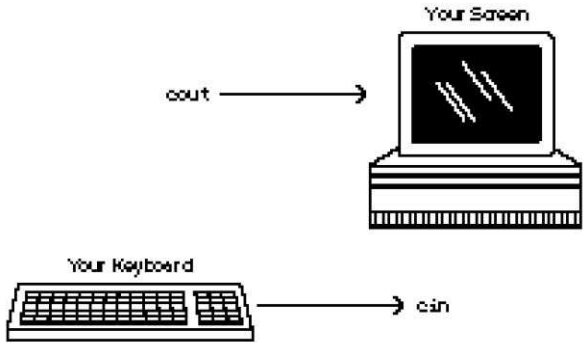


Figure 7.1. The actions of cout and cin.

Već je napomenuto da se za ispis vrijednosti koristi naredba **cout**, a za učitavanje podataka sa tastature naredba **cin**. U ovom poglavlju ove dvije naredbe biće objašnjene sa više detalja.

Da bismo koristili cout i cin, u program moramo uključiti biblioteku standarnih ulazno/izlaznih funkcija **iostream.h**.

4.1.cout

Cout šalje podatke na standardni izlazni uređaj. Standardni izlazni uređaj je monitor, iako se izlaz može preusmjeriti i na drugi uređaj.

U opštem slučaju, cout koristimo na slijedeći način:

cout << podaci [<< podaci]

pri čemu podaci mogu biti promjenljive, stringovi, izrazi ili njihova kombinacija.

String je kombinacija karaktera omeđena znakovima navoda. Npr. da bismo odštampali rečenicu: “Proljeće dolazi u Bosnu.” zadajemo naredbu

cout << " Proljece dolazi u Bosnu ";

cout ne obezbjeđuje automatski prelazak kursora u slijedeću liniju. To znači da će nakon izvršavanje naredbe cout kursor ostati u istom redu, iza poslednjeg odštampanog karaktera stringa navedenog u navodnicima.

cout može kao parametre uzimati i promjenljive. Prilikom štampanja, **cout** će odštampati vrijednosti promjenljive. Npr.

```
int i;
i=5;
cout << i;
```

rezultirat će štampanjem «5» na ekranu. Umjesto nevedene naredbe cout, možemo navesti i

cout << "i=" << i;

što će na ekranu dati “i=5”. U ovom primjeru kao parametar naredbe dajemo kombinaciju stringa i promjenljive.

Kao parametar cout možemo zadati i izraz. Tako npr.

cout << "zbir=" << (i+j);

prikazat će na ekranu zbir= “ zbir vrijednosti promjenljivh i i j”

Šta će biti odštampano nakon slijedećih naredbi?

cout << "Linija 1"; cout << "Linija 2"; cout << "Linija 3";

Odgovor je

Linija 1Linija 2 Linija3

što vjerovatno ne želite. cout omogučava formatiranje izlaza korištenjem skupa specijalnih karaktera. Tako npr., u cout možete uključiti specijalan karakter ***\n*** koji nazvačava prelazak na novu liniju, kada god želite pomjeriti kursor nakon ispisa stringa navedenog u cout. To postižemo naredbom:

cout << "Linija 1\n"; cout << "Linija 2\n"; cout << "Linija 3\n";

što daje kao rezultat:

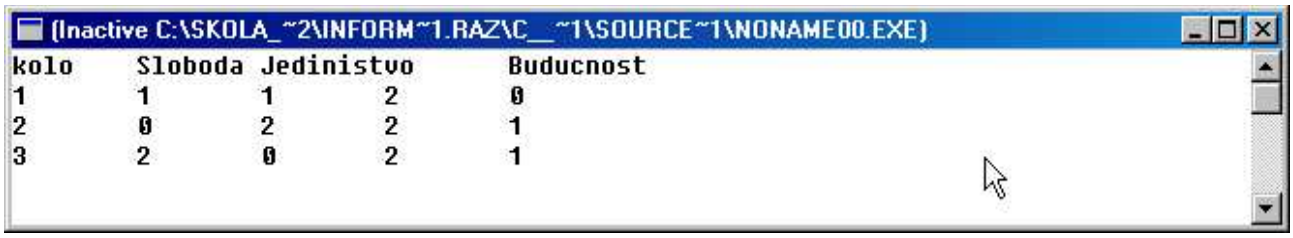
Linija 1
Linija 2
Linija3

Ukoliko želimo preskočiti više od jedne linije, zadajemo onoliko \n koliko linija želimo preskočiti.

Ako želimo štampati tabelu možemo koristi specijalni karakter **\t**. Npr. da bismo odštampali tabelu u kojoj prikazujemo broj osvojenih bodova za tri fudbalska tima u tri kola, program bi izgledao ovako:

```
#include <iostream.h>
int main()
{
                    cout << "kolo" << "\tSloboda" << "\tJedinistvo" << "\tBuducnost" << "\n";
                                        cout <<"1\t1\t1\t2\t0\n";
                                        cout <<"2\t0\t2\t2\t1\n";
                                        cout <<"3\t2\t0\t2\t1\n";
                                                return 0;
}
```

Rezultat je



Umjesto specijalnog karaktera ***\n*** možemo koristiti i ***endl***. Prethodni primjer dao bi isti izlaz i sa slijedećim programom:

```
#include <iostream.h>
int i,j,k;
int main()
{
                    cout << "kolo" << "\tSloboda" << "\tJedinistvo" << "\tBuducnost"  << endl;
                                        cout <<"1\t1\t1\t2\t0" << endl;
                                        cout <<"2\t0\t2\t2\t1"<< endl;
                                        cout <<"3\t2\t0\t2\t1"<< endl;
                                                cin >> i>>j>>k;
                                                return 0;
}
```

Pored spomenutih, za formatiranje izlaza možemo koristiti i druge specijalne znakove, kao što je prikazano u tabeli:

Specijalni znakovi	
\n	novi red
\t	horizontalni tabulator
\v	vertikalni tabulator
\b	pomak za mjesto unazad (*backspace*)
\r	povrat na početak retka (*carriage return*)
\f	nova stranica (*form feed*)
\a	zvučni signal (*alert*)
\\	kosa crta ulijevo (*backslash*)
\?	upitnik
\'	jednostruki navodnik
\"	dvostruki navodnik
\0	završetak znakovnog niza
ddd	znak čiji je kôd zadan oktalno s 1, 2 ili 3 cifre
\x*ddd*	znak čiji je kôd zadan heksadekadski

Ukoliko želimo obrisati sadržaj ekrana možemo koristiti funkciju **clrscr()**. To je jedna od gotovih funkcija koje nudi biblioteka **conio.h**. Ova funkcija briše tekustalni sadržaj ekrana i postavlja kursor u gornji lijevi ugao.

4.2.cin

Format cin je

cin >> promjenljiva [>> promjenljiva]

cin učitava vrijednost koju korisnik unosi pomoću tastature i dodjeljuje je promjenljivoj čije ime navedemo kao parametar.

Postoji velika razlika između “punjenja” promjenljive pomoću cin i dodjele konstantne vrijednosti promjenljvoj pomoću operatora dodjele. Iako obje naredbe u osnovi dodjeljuju vrijednost promjenljivoj, dodjela vrijednosti promjenljivoj pomoću operatora dodjele vrši se unutar programa. Naredbom dodjele programer tačno definiše egzaktnu vrijednost promjenljivoj. Svaki puta kada startujete program, ta promjenljiva dobit će istu vrijednost. Kada koristite cin, promjenljiva će dobiti vrijednost koju korisnik učita. To može biti bilo koja vrijednost i ovisi o korisniku.

Korištenje cin može biti uzrok problema – cin očekuje da korisnik upiše tačno određeni tip podataka. Npr. ako je parametar cin promjenljiva tipa int, onda korisnik može upsati samo cio broj. U principu, korisnik može

postupiti i drugačije. Da bi se ovo izbjeglo može pomoći ako prije svakog cin zadamo i cout u kome korisniku ispišemo kakvu vrijednost od njega očekujemo. Npr.

```
cout << "Unesite cio broj =";
cin>> i;
```

Pomoću cin možemo učitati i više od jedne promjenljive. Npr.

```
cin >> i >> j >> k;
```

očekuje od korisnika tri brojne vrijednosti. Nakon unosa svake od tih vrijednosti korisnik mora pritisnuti taster Enter da bi omogučio učitavanje slijedeće vrijednosti.

Vježbe:

Primjer 1

Unesite slijedeći program:

```
#include <iostream.h>
int main()
{   cout << "Hello, world!\n";
    return 0;
}
```

Snimite program pod imenom a:\primjer1.cpp. Prevedite ga i pogledajte dobiveni ispis. Zatvorite C++ i pogledajte datoteke dobivene prevođenjem. Pokrenite dobiveni program iz komandne linije. Vratite se u C++.

Primjer 2

Načinite slijedeće promjene u gornjem programu:

```
cout << "Ovo " << "je pr" << "va vježba!\n";
cout << "\n1\n2\n3\n";
cout << "\n1\t2\t3\n";
// U ovom primjeru ispisujemo cijeli broj
cout << endl << 100 << endl;
cout << 100 << " " << 200 << endl;
cout << 2.55 << endl; /* U ovom primjeru ispisujemo realni broj */
```

Zadatak 1

Napišite program koji će ispisati na ekranu vaše ime i adresu.

Zadatak 2

Napišite program koji će na ekranu ispisati u jednom redu brojeve 2002 i 12.34.

Primjer 2

Isprobajte slijedeći program:

```
#include <iostream.h>
#include <conio.h>
int main()
{   int x, y;
    clrscr();   // Funkcija koja briše ekran
    x = 100;    // Dodjeljuje vrijednost 100 varijabli x
    cout << "x = " << x << endl;
    y = x + 45; // Dodjeljuje vrijednost 145 varijabli y
    cout << "y = " << y << endl;
    return 0;
}
```

Zadatak 3

Napišite program koji će izračunati i ispisati ukupan otpor dva serijski spojena otpora, ako prvi iznosi 220 , a drugi 560 .

Primjer 3

Isprobajte i komentarišite slijedeći program:

```
#include <iostream.h>
int main()
{   int x;
    cout << x << endl;
    return 0;
}
```

Zamijenite naredbu `int x;` s `int x = 1000;`

Primjer 4

Isprobajte i komentirišite slijedeći program:

```
#include <iostream.h>
int main()
{   float x = 10.45, y = 100000000;
    cout << x << endl << y << endl;
    return 0;
}
```

Zadatak 4

Napišite program koji će obrisati ekran i zatim na ekranu ispisati: u prvom redu ime škole, u drugom redu adresu škole i u trećem redu vaš razred.

Zadatak 5

Napišite program koji će izračunati i ispisati prosječnu brzinu automobila koji je put od 180km prešao za 2 sata i 30 minuta.

Zadatak 6

Napišite program koji ispisuje

```
 CCCC            +                                                    +
C                                                                    +
                                                                     +
C                                        ++++++
+++++
C                                                                    +
                                                                     +
 CCCC            +                                                    +
```

5. DEKLARACIJA PROMJENLJIVIH

Promjenljive koje koristimo u progamu imaju svoje karakteristike. Kada trebamo neku promjenljivu u programu moramo je najprije deklarisati.

Deklaraciju promjenljve možemo izvršiti na dva mjesta u programu:

1. prije koda koji koristi tu promjenljivu,
2. prije funkcije main() u glavnom programu

Pri deklarisanju promjenljive treba imati na umu slijedeće karakteristike:

- Svaka promjenljiva ima svoje ***ime***
- Svaka promjenljiva ima ***tip***
- Svaka promjenljiva sadrži ***vrijednost*** koju joj dodjeljujemo najčešće operatorom dodjele vrijednosti.

Npr. naredbom

```
int i,j;
```

deklarišemo dvije cjelobrojne promjenljive kojima dodjeljujemo imena i i j. Možemo reći da ovom naredbom "kažemo": *Koristit ću dvije cjelobrojne promjenljive u programu. Njihova imena su i i j. Kada im dodijelim vrijednost, to će biti cio broj.*

Ako bismo izostavili deklaraciju, promjenljivim i i j kasnije ne bismo mogli dodijeliti vrijednost. S***vaka promjenljiva mora biti deklarisana prije korištenja***.

Deklaracija je u prethodnom primjeru obavljena u jednoj naredbi, ali smo to mogli uraditi i na slijedeći način:

```
int i;
int j;
```

5.1. Imenovanje promjenljivih

U programu, u opštem slučaju, koristimo više promjenljivih, stoga svaka mora imati svoje jedinistveno ime. Prilikom imenovanja promjenljivih, treba se pridržavati slijedećih pravila:

- Imena promjenljivih mogu biti jedan karakter ili maksimalno 32 karaktera.
- Ime mora početi slovom ili znakom _, poslije čega možemo koristiti brojeve, slova ili specijalne karaktere.
- Praznine nisu dozvoljene u imenu promjenljive, umjesto njih može se koristit znak _.
- Iako je dozvoljeno koristiti bilo kakvo ime (u skladu sa gore navedenim pravilima) preporučuje se izbor imena promjenljive koje asocira na vrijednost koja će se u njoj čuvati.
- C++ razlikuje mala i velika slova; promjenljive ***suma, Suma, SUMA*** su za C različite promjenljive
- Promjenljive ne treba da imaju isto ime kao i nazivi C++ komandi i ugrađenih funkcija

5.2. Tipovi podataka

Promjenljive koje koristimo u programima mogu sadržati različite vrste podataka. U C++ jeziku ugrađeno je nekoliko osnovnih tipova podataka i definisane su operacije nad njima. U sljedećim poglavljima prvo ćemo se upoznati s brojevima, da bismo kasnije prešli na znakove, logičke vrijednosti i pobrojane tipove. Potom ćemo upoznati i operatore koje je dozvoljeno koristiti nad pojedinim tipom podataka.

U tabeli koja slijedi dat je zbirni pregled tipova podataka.

TIP	ključna riječ	*opis*	broj bajta	OPSEG VRIJEDNOSTI
znakovni	char	znakovni	1	
cijeli brojevi	int	cjelobrojni	2 4	–32768 do 32767 –2147483648 do 2147483647
	unsigned int	neoznačeni cjelobrojni		
	signed int	označeni cjelobrojni (isti kao int)		
	short int	kratki cjelobrojni	2	–32768 do 32767
	unsigned short int	Unsigned short integer		
	signed short int	označeni kratki cjelobrojni (isti kao short int)		
	long	Long integer	4	–2147483648 do 2147483647
	long int	Long integer (same as long)		
	signed long int	Signed long integer (same as long int)		
	unsigned long int	Unsigned long integer		
realni brojevi	float	Floating-point	4	$-3{,}4\cdot10^{38}$ do $-3{,}4\cdot10^{-38}$ i $3{,}4\cdot10^{-38}$ do $3{,}4\cdot10^{38}$
	double	Double floating-point	8	$-1{,}7\cdot10^{308}$ do $-1{,}7\cdot10^{-308}$ i $1{,}7\cdot10^{-308}$ do $1{,}7\cdot10^{308}$
	long double	Long double floating-point	10	$-1{,}1\cdot10^{4932}$ do $-3{,}4\cdot10^{-4932}$ i $3{,}4\cdot10^{-4932}$ do $1{,}1\cdot10^{4932}$

5.2.1. Brojevi

U jeziku C++ ugrađena su dva osnovna tipa brojeva: ***cijeli brojevi*** (engl. *integers*) i ***realni brojevi*** (tzv. *brojevi s pokretnom decimalnom tačkom*, engl. *floating-point*).

Najjednostavniji tip brojeva su cijeli brojevi. Pod cijelim brojem podrazumijevamo bilo koji broj koji ne sadrži decimalni dio. Cjelobrojna promjenljiva deklariše se riječju ***int*** i njena će vrijednost u memoriji računara obično zauzeti dva *bajta* (engl. *byte*), tj. 16 bita. Prvi bit je rezervisan za predznak, tako da preostaje 15 bita za pohranu vrijednosti.

Za većinu praktičnih primjena taj opseg vrijednosti je dovoljan. Međutim, ukaže li se potreba za većim cijelim brojevima promjenljivu možemo deklarisati kao ***long int***, ili kraće samo ***long***:

Za promjenljive koje će čuvati realne brojeve najčešće se koristi tip ***float***: Realini brojevi su brojevi koji sadrže cio i decimalni dio.

```
float pi = 3.141593;
float brzinaSvjetlosti = 2.997925e8;
float nabojElektrona = -1.6E-19;
```

Prvi primjer odgovara uobičajenom načinu prikaza brojeva s decimalnim zarezom. U drugom i trećem primjeru brojevi su prikazani u naučnoj notaciji, kao proizvod mantise i eksponenta na bazi 10 ($2{,}997925 \cdot 10^{8}$, odnosno $-1{,}6 \cdot 10^{-19}$). Kod prikaza u naučnoj notaciji, slovo '`e`' koje razdvaja mantisu od eksponenta može biti veliko ili malo.

Osim šta je ograničen opseg vrijednosti koje se mogu prikazati `float` tipom, treba znati da je i broj decimalnih cifara u mantisi također ograničen na 7 decimalnih mjesta. Čak i ako se napiše broj s više cifara, kompajler će zanemariti sve niže decimalne cifre. Stoga će ispis slijedećih promjenljivih dati potpuno isti broj.

```
float pi_tacniji    = 3.141592654;
float pi_manjeTacan = 3.1415927;
```

Ako tačnost na sedam decimalnih cifri ne zadovoljava ili ako se koriste brojevi veći od 10^{38} ili manji od 10^{-38}, tada se umjesto `float` mogu koristiti brojevi s dvostrukom tačnošću tipa ***double*** koji pokrivaju opseg vrijednosti od $1.7 \cdot 10^{-308}$ do $1.7 \cdot 10^{308}$, ili ***long double*** koji pokrivaju još širi opseg od $3.4 \cdot 10^{-4932}$ do $1.1 \cdot 10^{4932}$.

Ako želite sami provjeriti veličinu pojedinog tipa, to možete učiniti i pomoću ***sizeof*** operatora, kao u primjeru 5.

```
cout << Veličina cijelih brojeva:  << sizeof(int);
cout << Veličina realnih brojeva:  << sizeof(float);
```

Svi navedeni brojevni tipovi mogu biti deklarisani i bez predznaka, dodavanjem riječi ***unsigned*** ispred njihove deklaracije:

```
unsigned int i = 40000;
unsigned long int li;
unsigned long double BrojZvijezdaUSvemiru;
```

U tom slučaju će kompajler bit za predznak upotrijebiti kao dodatnu binarnu cifru, pa se najveća moguća vrijednost udvostručuje u odnosu na promjenljive s predznakom, ali se naravno ograničava samo na pozitivne vrijednosti. Pridružimo li `unsigned` promjenljivoj negativan broj, ona će poprimiti vrijednost nekog pozitivnog broja. Na primjer, programski slijed

```
unsigned int = -1;
cout << i << endl;
```

će za promjenljivu `i` ispisati broj 65535 (uočimo da je on za 1 manji od najvećeg dozvoljenog `unsigned int` broja 65536). Zanimljivo da kompajler za gornji kôd neće javiti grešku. Kompajler ne prijavljuje grešku ako se `unsigned` promjenljivoj pridruži negativna konstanta - korisnik mora sam paziti da se to ne dogodi!

Vježbe

Primjer 1:

```
/*                                  Demonstracija upotrebe promjenljivih
Izracunavanje povrsine sa zadatim vrijednostima*/
#include<iostream.h>
int Duzina;
int Sirina;
int Povrsina;
int main()
{
                                        Duzina = 10;
                                        Sirina = 20;
```

```
                                        Povrsina = Duzina * Sirina;
                                    cout << "Duzina= " <<  Duzina << endl;
                                    cout << "Sirina= " << Sirina << endl;
                                   cout <<"Povrsina= " <<Povrsina << endl;
                                                return 0;
}
```

Primjer 2:

```
/*                                  Demonstracija upotrebe promjenljivih
                                Izracunavanje povrsine sa ucitanim cjelobrojnim vrijednostima
*/

#include<iostream.h>
int Duzina, Sirina, Povrsina;

int main()

{
                                cout << "IZRACUNAVANJE POVRSINE" << endl;
                                        cout << "Unesite duzinu -->";
                                                cin >> Duzina;
                                        cout << "Unesite sirinu -->";
                                                cin >> Sirina;

                                        Povrsina = Duzina * Sirina;

                                cout <<"Zadana duzina = " << Duzina << endl;
                                cout <<"Zadana sirina = " << Sirina << endl;
                                cout <<"Povrsina = " << Povrsina << endl;

                                                return 0;
}
```

Primjer 3:

```
/*                                  Demonstracija upotrebe promjenljivih
                                Izracunavanje povrsine sa ucitanim realnim vrijednostima
*/



#include<iostream.h>

float Duzina, Sirina, Povrsina;

main()

{
                                cout << "IZRACUNAVANJE POVRSINE" << endl;
                                        cout << "Unesite duzinu -->";
                                                cin >> Duzina;                          // ucitavanje duzine
                                        cout << "Unesite sirinu -->";
                                       cin >> Sirina;    // ucitavanje sirine

                                        Povrsina = Duzina * Sirina;

                                cout << "Zadana duzina = " << Duzina << endl;
                                cout << "Zadana sirina =" << Sirina << endl;
                                cout << "Povrsina= " <<  Povrsina << endl;

                                                return 0;
}
```

Primjer 4:

```
/* Primjeri deklaracije podataka sa zadatim vrijednostima*/

int CioBrojX, CioBrojY, CioBrojZ ;
float RealanBrojX, RealanBrojY, RealanBrojZ;

main()
{
                                        cout << "**** CIJELI BROJEVI (tip int) **** \n";
                                                CioBrojX = 100;
                                                CioBrojY = 1000;
                                            CioBrojZ = CioBrojX/CioBrojY;

                                        cout << "CioBrojX = " << CioBrojX << endl;
                                        cout << "CioBrojY = " <<  CioBrojY<< endl;
                                cout << "Kolicnik cijelih brojeva je uvijek cio broj! " << endl;
                            cout << CioBrojX << ":" <<CioBrojY << " = " << CioBrojZ << endl;

                                    cout << "\n**** REALNI BROJEVI (tip float) **** \n";
                                                RealanBrojX = 100.0;
                                                RealanBrojY = 1000.0;
                                            RealanBrojZ = RealanBrojX/RealanBrojY;

                                        cout << "RealanBrojX = " << RealanBrojX << endl;
                                        cout << "RealanBrojY = " << RealanBrojY << endl;
                                cout << "Kolicnik realnih brojeva je uvijek realan broj! "  << endl;
                        cout << RealanBrojX << ":" <<RealanBrojY << " = " << RealanBrojZ << endl;
                                                return 0;
}
```

Zadatak za vježbu:

Napisati program koji računa količnik cijelih, a zatim realnih brojeva kao u prethodnom primjeru, ali omogučiti da korisnik učitava brojeve.

Primjer 5:

Napisati program koji prikazuje koliko bajta se koristi za pamćenje promjenljivih tipa int, unsigned int, long, unsigned long i short, te koje su minimalne i maksimalne vrijednosti.

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

int main()
{
                                        cout << "\n********** INT tip \n";
                                                int bajtInt=sizeof(int);
                                                int maxInt = INT_MAX;
                                                int minInt = INT_MIN;
                                    cout << "Broj bajta \t \t" << bajtInt << "\n";
                                cout << "Najveca vrijednost \t" << maxInt << "\n";
                                cout << "Najmanja vrijednost \t " << minInt << "\n";

                                    cout << "\n********** UNSIGNED INT tip \n";
                                                int bajtUInt=sizeof(unsigned int);
                                            unsigned int maxUInt = UINT_MAX;
                                    cout << "Broj bajta \t \t" << bajtUInt << "\n";
                                cout << "Najveca vrijednost \t" << maxUInt << "\n";

                                        cout << "\n********** LONG tip \n";
                                                int bajtLong=sizeof(long);
```

```
                                        long maxLong = LONG_MAX;
                                        long minLong = LONG_MIN;
                                    cout << "Broj bajta \t \t" << bajtLong << "\n";
                                cout << "Najveca vrijednost \t"  << maxLong << "\n";
                                cout << "Najveca vrijednost \t"  << minLong << "\n";

                                cout << "\n********** UNSIGNED LONG tip \n";
                                        int bajtULong=sizeof(unsigned long);
                                    unsigned long maxULong = ULONG_MAX;
                                    cout << "Broj bajta \t \t" << bajtULong << "\n";
                                cout << "Najveca vrijednost \t"  << maxULong << "\n";

                                    cout << "\n********** SHORT tip \n";
                                        int bajtShort=sizeof(long);
                                    short maxShort = SHRT_MAX;
                                    short minShort = SHRT_MIN;
                                cout << "Broj bajta \t \t" << bajtShort << "\n";
                                cout << "Najveca vrijednost \t"  << maxShort << "\n";
                                cout << "Najveca vrijednost \t"  << minShort << "\n";
                                                return 0;
}
```

Primjer 6:

Napisati program koji prikazuje koliko bajta se koristi za pamćenje promjenljivih tipa float, double, te koje su minimalne i maksimalne vrijednosti.

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

int main()
{

                                    cout << "\n********** FLOAT tip \n";
                                        int bajtFloat=sizeof(float);
                                    float maxFloat = MAXFLOAT;
                                    float minFloat = MINFLOAT;
                                cout << "Broj bajta \t \t" << bajtFloat << "\n";
                            cout << "Najveca vrijednost \t" << maxFloat << "\n";
                            cout << "Najmanja vrijednost \t " << minFloat << "\n";

                                cout << "\n********** DOUBLE tip \n";
                                        int bajtDouble=sizeof(double);
                                    double maxDouble = MAXDOUBLE;
                                    double minDouble = MINDOUBLE;
                                cout << "Broj bajta \t \t" << bajtDouble << "\n";
                            cout << "Najveca vrijednost \t"  << maxDouble << "\n";
                            cout << "Najveca vrijednost \t"  << minDouble << "\n";

                                                return 0;
}
```

Zadatak za vježbu:

Pozvati iz “help-a” header datoteke limits.c i values.c. Proučiti konstante koje su pohranjene u ovim header datotekama i napisti program za ispisivanje tih konstanti uz odgovarajuće objašnjenje.

5.2.2. Znakovi

Znakovne konstante tipa ***char*** pišu se uglavnom kao samo jedan znak unutar jednostrukih znakova navoda:

```
char SlovoA = 'a';    cout << 'b' << endl;
```

Znakovne konstante najčešće se koriste u *znakovnim nizovima* (engl. *strings*) za ispis tekstova, te ćemo ih detaljnije upoznati u poglavlju koje se odnosi na string. Za sada samo spomenimo da se znakovni nizovi sastoje od nekoliko znakova unutar dvostrukih navodnika. Zanimljivo je da se `char` konstante i varijable mogu uspoređivati, poput brojeva:

```
cout << ('a' < 'b') << endl;
cout << (a < B) << endl;
cout << ('A' > 'a') << endl;
cout << ('\'' != '\"') << endl;     // uporedba navodnika ` i "
```

pri čemu se u biti uspoređuju njihovi brojni ekvivalenti u nekom od standarda. Najraširenija je ASCII niz (skraćenica od *American Standard Code for Information Interchange*) u kojem su svim ispisivim znakovima, brojevima i slovima engleskog alfabeta pridruženi brojevi od 32 do 127, dok specijalni znakovi imaju kôdove od 0 do 31 uključivo. U prethodnom primjeru, prva naredba ispisuje 1, jer je ASCII kôd malog slova `a` (97) manji od kôda za malo slovo `b` (98). Druga naredba ispisuje 0, jer je ASCII kôd slova B jednak 66 (nije važno što je B po abecedi iza a!). Treća naredba ispisuje 0, jer za veliko slovo `A` kôd iznosi 65. ASCII kôdovi za jednostruki navodnik i dvostruki navodnik su 39 odnosno 34, pa zaključite sami šta će četvrta naredba ispisati.

Primjer 7:

Nad znakovima se mogu primjenjivati operacije + i -.

```
#include <iostream.h>
char znak;

int main()
{
                                        znak = 'A';
                                cout << "Znak = " << znak << endl;
                                        znak = znak + 1;
                                cout << "Znak uvecan za 1 = "  << znak << endl;
                                        znak = znak - 1;
                                cout << "Znak umanjen za 1 = "  << znak << endl;
                                        return 0;
}
```

5.2.3. Logički tip podataka

Logički podaci su takvi podaci koji mogu poprimiti samo dvije vrijednosti, na primjer: da/ne, istina/laž, dan/noć. Jezik C++ za prikaz podataka logičkog tipa ima ugrađen tip ***bool***, koji može poprimiti vrijednosti ***true*** (engl. *true* - *tačno*) ili ***false*** (engl. *false* - *pogrešno*):

```
bool JeLiDanasNedjelja = true;
bool SunceSije = false;
```

Pri ispisu logičkih tipova, te pri njihovom korištenju u aritmetičkim izrazima, logički tipovi se pretvaraju u `int`: `true` se pretvara u cjelobrojni 1, a `false` u 0.

5.2.4. Pobrojani tip

Ponekad su promjenljive u programu elementi pojmovnih skupova, tako da je takvim skupovima i njihovim elementima zgodno dodijeliti imena. Za takve slučajeve obično se koriste *pobrojani tipovi* (engl. *enumerated types*):

enum dani {ponedjeljak, utorak, srijeda, cetvrtak, petak, subota, nedjelja};

Ovom deklaracijom uvodi se novi tip podataka `dani`, te sedam nepromjenjivih identifikatora (`ponedjeljak`, `utorak`,...) toga tipa. Prvom identifikatoru kompajler pridjeljuje vrijednost 0, drugom 1, itd. Sada možemo definisati varijablu tipa `dani`, te joj pridružiti neku od vrijednosti iz niza:

dani HvalaBoguDanasJe = petak;
dani ZakajJaNeVolim = ponedjeljak;

Naredbom

cout << HvalaBoguDanasJe << endl;

na ekranu se ispisuje cjelobrojni ekvivalent za `petak`, tj. broj 4.

Promjenljive tipa `dani` mogli smo deklarisati i neposredno uz definiciju pobrojanog tipa:

enum dani{ponedjeljak, utorak, srijeda, cetvrtak, petak, subota, nedjelja} ThankGodItIs, SunnyDay;
ThankGodItIs = petak; SunnyDay = nedjelja;

Korištenje pobrojanih vrijednosti upoznaćemo u poglavlju nizovi.

5.2.5. Konstante

U programima se često koriste simboličke veličine čija se vrijednost tokom izvođenja ne želi mijenjati. To mogu biti fizičke ili matematičke konstante, ali i parametri poput maksimalnog broja prozora ili maksimalne dužine znakovnog niza, koji se inicijalizuju prije prevođenja kôda i ulaze u izvršni kôd kao konstante.

Konstante definišemo navođenjem ključne riječi ***const***, tipa podataka i naziva konstante, a zatim joj odmah dodjeljujemo vrijednost. U main funkciji, konstanta više ne može promijeniti vrijednost. Na svaki pokušaj promjene vrijednosti takve promjenljive, kompajler će javiti grešku:

const double pi = 3.14159265359;
pi = 2 * pi; // greška!

Drugi često korišteni pristup zasniva se na pretprocesorskoj naredbi `#`***`define`***:

#define PI 3.14159265359

U primjeru koji slijedi definisana je konstanta pi.

Primjer 8

```
/* Demonstracija deklarisanje konstanti */
#include <iostream.h>;

const float pi=3.14159;
float Poluprecnik, Obim, Povrsina;

main()
{
                    cout << "Izracunavanje obima i povrsine kruga" << endl;
                        cout << "Unesite poluprecnik kruga -->";
                            cin >> Poluprecnik;
```

```
                Obim = 2* Poluprecnik * pi;
                Povrsina=Poluprecnik*Poluprecnik*pi;

                cout << "Obim = " << Obim << endl ;
                cout << "Povrsina = " << Povrsina << endl;

                return 0;
}
```

6. OPERATORI

6.1. Aritmetički operatori

Da bismo promjenljive u programu mogli mijenjati, na njih treba primijeniti odgovarajuće operacije. Za ugrađene tipove podataka definisani su osnovni operatori, poput sabiranja, oduzimanja, množenja i dijeljenja (vidi tablelu). Ti operatori se mogu podijeliti na ***unarne***, koji djeluju samo na jednu promjenljivu, te na ***binarne*** za koje su neophodne dvije promjenljive.

Aritmetički operatori		
unarni operatori	`+x`	unarni plus
	`-x`	unarni minus
	`x++`	uvećaj nakon
	`++x`	uvećaj prije
	`x--`	umanji nakon
	`--x`	umanji prije
binarni operatori	`x + y`	sabiranje
	`x - y`	oduzimanje
	`x * y`	množenje
	`x / y`	dijeljenje
	`x % y`	modulo

6.1.1. Unarni operatori

Unarni operatori djeluju na samo jednu promjenljivu. Npr, broju možemo dodijeliti pozitivnu ili negativnu vrijednost primjenom unarnog operatora + ili -. Npr.

```
a = -25;                    //dodjeljuje a –25
b = +25;                 // dodjeljuje a  25 (+ nije neophodan)
c = -a;           // dodjeljuje c negativnu vrijednost promjenljive a (-25)
d = +b;           // dodjeljuje d pozitivnu vrijednost b (+ nije neophodan)
```

Osim unarnog plusa i unarnog minusa koji mijenjaju predznak broja, u jeziku C++ definisani su još i unarni operatori za ***uvećavanje (inkrementiranje) i umanjivanje vrijednosti (dekrementiranje)*** broja. Operator `++` uvećava vrijednost varijable za 1, a operator `--` umanjuje vrijednost varijable za 1. Tako naredbe:

```
x=x+1
z=z-1
```

možemo zamijeniti naredbama:

x++
z—

Primjer koji slijedi demonstrira korištenje ovih operatora.

```
/* Demonstracija izraza, operatori inkrement i dekrement*/
#include <iostream.h>
int Prvi, Drugi;
int Pomocna;
main()
{
                                        cout << "Unesite prvi broj -> ";
                                                cin >> Prvi;
                                        cout << "Unesite drugi broj -> ";
                                                cin >>  Drugi;

                                                /* Zamjena brojeva*/
                                                Pomocna=Drugi;
                                                Drugi=Prvi;
                                                Prvi=Pomocna;

                                cout << "\nBrojevi su zamijenjeni. Sada imaju vrijedosti: \n";
                                        cout << "Prvi=" << Prvi << endl;
                                        cout << "Drugi=" << Drugi <<endl;

                                        /*Inkrement - uvecavanje za 1*/
                                                Prvi++;                                                 /*
Prvi=Prvi+1 */
                                                Drugi++;                                /* Drugi = Drugi +1 */
                                cout << "Brojevi su inkrementirani. Sada imaju vrijednosti:\n";
                                        cout << "Prvi=" << Prvi << endl;
                                        cout << "Drugi=" << Drugi <<endl;

                                        /*Dekrement - umanjivanje za 1*/
                                                Prvi--;                                                 /*
Prvi = Prvi -1 */
                                                Drugi--;                                /* Drugi = Drugi -1 */
                                cout << "Brojevi su dekrementirani. Sada imaju vrijednosti:\n";
                                        cout << "Prvi=" << Prvi << endl;
                                        cout << "Drugi=" << Drugi <<endl;

        return 0;

}
```

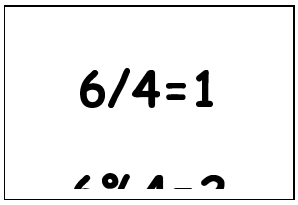
Operator inkrement (dekrement) različito se ponaša kada je on napisan ispred promjenljive i kada je napisan iza nje. U prvom slučaju (*prefiks* operator), vrijednost varijable će se prvo uvećati / umanjiti, a potom će biti dohvaćena njena vrijednost. U drugom slučaju (*postfiks* operator) je obrnuto: prvo se dohvati vrijednost varijable, a tek onda slijedi promjena. Slijedeći primjer to ilustruje.

```
/* Operatori inkrement i dekrement - prefix i postfix*/
#include <iostream.h>

int broj;

main()
{
                                        cout << "Unesite broj -> ";
                                                cin >> broj;

                                cout << "Ucitani broj je =" << broj << endl;
```

```
                                        cout << endl;

                                // prefix najprije doda 1, pa ispise novu vrijednost
                                    cout << "++broj =" << (++broj) << endl;
                                        cout << "broj =" << broj << endl;
                                            cout << endl;

  // postfix najprije ispise vrijednost, a zatim doda 1
                                    cout << "broj++ =" << (broj++) << endl;
                                        cout << "broj =" << broj << endl;

                                                return 0;
}
```

6.1.2. Binarni operatori

Na raspolaganju imamo pet binarnih aritmetičkih operatora: za sabiranje, oduzimanje, množenje, dijeljenje i *modul* operator:



Operator ***modul*** kao rezultat vraća ostatak dijeljenja dva cijela broja:

```
int i = 6;
int j = 4;
cout << (i % j) << endl;    // ispisuje 2
```

On se vrlo često koristi za ispitivanje djeljivosti cijelih brojeva: ako su brojevi djeljivi, ostatak nakon dijeljenja će biti nula.

```
/* Demonstracija operatora modul % */
#include <iostream.h>

int Prvi, Drugi, Ostatak;

int main()
{
                                    cout << "Operator modul\n \n";
                                     cout <<"Unesite prvi broj - ";
                                            cin >> Prvi;
                                    cout << "Unesite drugi broj - ";
                                            cin >> Drugi;

                                /* Izracunavanje ostatka dijeljenja*/
                                        Ostatak=Prvi%Drugi;

                            cout << "\nOstatak od cjelobrojnog dijeljenja brojeva \n";
                                cout << Prvi << " % " << Drugi << "=" << Ostatak;

                                                return 0;
}
```

izraz	rezultat	
10/2	5	nema ostatka
300/100	3	nema ostatka
10/3	3	ostatak zanemaren
300/165	1	ostatak zanemaren

Operacija djeljenja može dati neočekivan rezultat, ukoliko ne obratimo pažnju na deklaraciju promjenljivih koje dijelimo. Ako su oba operatora cjelobrojne promjenljive, C++ će dati rezultat koji je opet cio broj, zanemarujući ostatak, bez obzira da li su brojevi djeljivi ili ne. U tabeli su prikazane neke od tih situacija.

Često puta pišemo brojne izraze u kojima želimo «ažurirati» vrijednost neke varijable – učiniti je aktuelnom. Npr:

```
prosjek = prosjek *1.2
```

Ovakvi izrazi mogu biti napisani u komprimiranoj formi,

i++ =2

C++ nudi nekoliko ovakvih operatora, kao što je prikazano u tabeli.

Operator	Primjer	ekvivalentan izraz
+=	x+=500;	x=x+500;
-=	x-=50;	x=x-50;
*=	x*=1.2;	x=x*1.2;
/=	x/=.50;	x=x/.50;
%=	x%=7;	x=x%7;

6.1.3. Miješanje tipova promjenljivih u brojnim izrazima

U brojnim izrazima možemo navoditi promjenljive deklarisane različito; možemo miješati tipove. Uopšteno, C++ vrši konverziju "manjeg" tipa u "veći". Npr. ako dodate double na integer promjenljivu, C++ najprije pretvori integer u double, a zatim vrši sabiranje. Rezultat je, naravno, tipa float. Automatska konverzija je samo privremena, te se nakon izvršene operacije tip podataka zapravo ne mijenja. U navedenom primjeru integer promjenljiva ostaje integer.

Pogledajmo i sljedeći jednostavni primjer:

```
int Brojnik = 1;
int Nazivnik = 4;
float Koeficijent = Brojnik / Nazivnik;
cout << Koeficijent << endl;
```

Na ekranu će se kao rezultat ispisati `0.`! Iako smo rezultat dijeljenja pridružili `float` varijabli, pri izvođenju programa to pridruživanje slijedi tek nakon što je operacija dijeljenja dva cijela broja bila završena. Budući da su obje promjenljive, `Brojnik` i `Nazivnik` cjelobrojne, kompajler provodi cjelobrojno dijeljenje u kojem se zanemaruju decimalna mjesta. Stoga je rezultat cjelobrojni dio količnika varijabli `Brojnik` i `Nazivnik` (0.25), a to je 0. Slična je situacija i kada dijelimo cjelobrojne konstante:

```
float DiskutabilniKoeficijent = 3 / 2;
```

Brojeve 3 i 2 kompajler će shvatiti kao cijele, jer ne sadrže decimalnu tačku. Zato će primijeniti cjelobrojni operator `/`, pa će rezultat toga dijeljenja biti cijeli broj 1.

Ako ipak želimo pravilan rezultat, zadajemo:

```
float TacniKoeficijent = 3. / 2.;
```

Dovoljno bi bilo decimalnu tačku staviti samo uz jedan od operanada - prema pravilima aritmetičke konverzije i drugi bi operand bio sveden na `float` tip.

6.1.4. Operator dodjele tipa

Šta učiniti želimo li podijeliti dvije cjelobrojne promjenljive, a da pritom ne izgubimo decimalna mjesta? Dodavanje decimalne tačke iza imena varijable nema smisla, jer će kompajler javiti pogrešku. Za eksplicitnu promjenu tipa varijable primijenjujemo operator *dodjele tipa* (engl. *type cast*, kraće samo *cast*):

```
int Brojnik = 1;
```

```
int Nazivnik = 3;
float TacanKolicnik = (float)Brojnik / (float)Nazivnik;
```

Navođenjem ključnih riječi `float` u zagradama ispred operanada njihove vrijednosti se pretvaraju u decimalne brojeve prije operacije dijeljenja, tako da je rezultat korektan. I ovdje bi bilo dovoljno operator dodjele tipa primijeniti samo na jedan operand - prema pravilima aritmetičke konverzije i drugi bi operand bio sveden na `float` tip. Da ne bi bilo zabune, same varijable `Brojnik` i `Nazivnik` i nadalje ostaju tipa `int`, tako da će njihovo naknadno dijeljenje

```
float OpetPogresanKolicnik = Brojnik / Nazivnik;
```

opet kao rezultat dati rezultat cjelobrojnog dijeljenja.

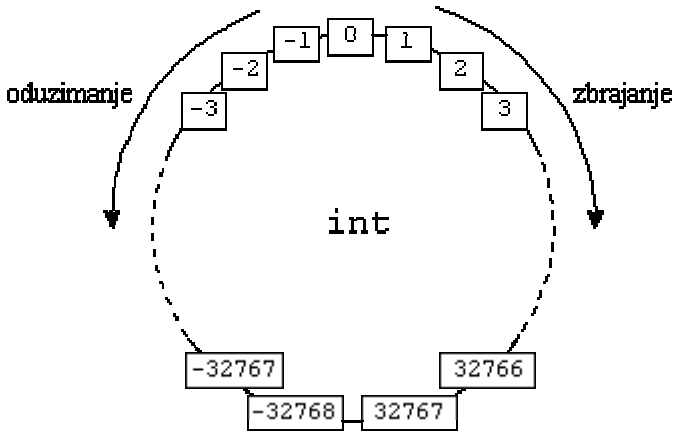
Jezik C++ dozvoljava i funkcijski oblik dodjele tipa u kojem se tip navodi ispred zagrade, a ime varijable u zagradi:

```
float TacanKolicnik = float(Brojnik) / float(Nazivnik);
```

6.1.5. Problem prekoračenja

Slijedeći problem vezan za aritmetičke operatore vezan je uz pojavu numeričkog preljeva, kada uslijed neke operacije rezultat nadmaši opseg koji dotični tip objekta pokriva. Razmotrimo primjer U donjem programu će prvo biti ispisan broj 32767, šta je najveći mogući broj tipa `int`. Izvođenjem naredbe u trećem redu, na ekranu računara će se umjesto očekivanih 32768 ispisati -32768, tj. najveći negativni `int`!

```
int i = 32766;
cout << (++i) << endl;
cout << (++i) << endl;
```



Uzrok tome je preljev podataka do kojeg došlo zbog toga što očekivani rezultat više ne stane u 15 bita predviđenih za `int` varijablu. Podaci koji su se "prelili" ušli su u bit za predznak (zato je rezultat negativan), a raspored preostalih 15 bita daje broj koji odgovara upravo onom šta nam je računar ispisao. Slično će se dogoditi oduzmete li od najvećeg negativnog broja neki broj. Ovo se može ilustrirati brojnom kružnicom na kojoj sabiranje odgovara kretanju po kružnici u smjeru kazaljke na satu, a oduzimanje odgovara kretanju u suprotnom smjeru kao na slici.

6.2.Logički operatori

Za logičke podatke definisana su svega tri operatora: `!` (*logička negacija*), `&&` (*logički i*), te `||` (*logički ili*) (vidi tabelu).

Logička negacija je unarni operator koji mijenja logičku vrijednost varijable: istinu pretvara u neistinu i obrnuto. **Logičko *i*** daje kao rezultat istinu samo ako su oba operanda istinita; radi boljeg razumijevanja u tabeli su dati rezultati logičkog *i* za sve moguće vrijednosti oba operanda. **Logičko *ili*** daje istinu ako je bilo koji od operanada istinit (vidi tabelu).

Logički operatori	
!x	**logička negacija**
x && y	**logički i**
x || y	**logički ili**

Logički operatori i operacije s njima uglavnom se koriste u naredbama za grananje toka programa, pa ćemo ih tamo još detaljnije upoznati.

6.3. Pogodbeni (relacioni) operatori

Osim aritmetičkih operacija, jezik C++ omogućava i usporedbu dva broja (vidi tabelu). Kao rezultat poređenja dobiva se tip `bool`: ako je uslov poređenja zadovoljen, rezultat je `true`, a ako nije rezultat je `false`. Tako će se izvođenjem kôda

```
cout << (5 > 4) << endl;      // je li 5 veće od 4?
cout << (5 >= 4) << endl;     // je li 5 veće ili jednako 4?
cout << (5 < 4) << endl;      // je li 5 manje od 4?
cout << (5 <= 4) << endl;     // je li 5 manje ili jednako 4?
cout << (5 == 4) << endl;     // je li 5 jednako 4?
cout << (5 != 4) << endl;     // je li 5 različito od 4?
```

na ekranu ispisati brojevi 1, 1, 0, 0, 0 i 1.

Relacioni operatori	
x < y	**manje od**
x <= y	**manje ili jednako**
x > y	**veće od_**
x >= y	**veće ili jednako**
x == y	**jednako**
x != y	**različito**

Relacioni operatori (od engl. *relational operators*) se koriste pretežno u naredbama za grananje toka programa, gdje se, ovisno o tome je li neki uslov zadovoljen, izvođenje programa nastavlja u različitim smjerovima. Stoga ćemo relacione operatore podrobnije upoznati kod naredbi za grananje.

Uočimo suštinsku razliku između jednostrukog znaka jednakosti (=) koji je simbol za pridruživanje, te dvostrukog znaka jednakosti (==) koji je relacioni operator!

6.4. Hijerarhija i redoslijed izvođenja operatora

U matematici postoji utvrđena hijerarhija operacija prema kojoj neke operacije imaju prednost pred drugima. Podrazumijevani slijed operacija je slijeva nadesno, ali ako se dvije operacije različitog prioriteta nađu jedna do druge, prvo se izvodi operacija s višim prioritetom. Na primjer u matematičkom izrazu

a + *b c* / *d*

množenje broja *b* s brojem *c* ima prednost pred sabiranjem s brojem *a*, tako da se ono izvodi prvo. Proizvod se zatim dijeli s *d* i tek se tada dodaje broj *a*.

I u programskom jeziku C++ definisana je hijerarhija operatora. Prvenstveni razlog tome je kompatibilnost s matematičkom hijerarhijom operacija, što omogućava pisanje računskih izraza na gotovo identičan način kao u matematici. Stoga gornji izraz u jeziku C++ možemo pisati kao

```
y = a + b * c / d;
```

Redoslijed izvođenja operacija će odgovarati matematički očekivanom. Operacije se izvode prema hijerarhiji operacija, počevši s operatorima najvišeg prioriteta. Ako dva susjedna operatora imaju isti prioritet, tada se operacije izvode prema redoslijedu izvođenja operatora. U tabeli su dati svi operatori svrstani po hijerarhiji od najvišeg do najnižeg. Operatori s istim prioritetom smješteni su u zajedničke blokove.

Hijerarhija i pridruživanje operatora

operator	značenje	pridruživanje	prioritet
++	uvećaj nakon	s lijeva na desno	najviši
--	umanji nakon	s lijeva na desno	
sizeof	veličina objekta	s desna na lijevo	
++	uvećaj prije	s desna na lijevo	
--	umanji prije	s desna na lijevo	
* & + - ! ~	unarni operatori	s desna na lijevo	
* / %	množenja	s lijeva na desno	
+ -	zbrajanja	s lijeva na desno	
< > <= >=	relacioni operatori	s lijeva na desno	
== !=	operatori jednakosti	s lijeva na desno	
&&	logički i	s lijeva na desno	
||	logički ili	s lijeva na desno	
?:	uslovni izraz	s desna na lijevo	
= *= /= += -= &= ^= |= %= >>= <<=	pridruživanja	s desna na lijevo	
,	razdvajanje	s lijeva na desno	najniži

Kao i u matematici, u izrazima možemo koristiti zagrade “(“ i “)”, i na taj način podesitii redoslijed izvršavanja operacija prema konkretnim potrebama.

7. NAREDBE ZA KONTROLU TOKA PROGRAMA

'Tko kontrolira prošlost,' glasio je slogan Stranke, 'kontrolira i budućnost: tko kontrolira sadašnjost kontrolira prošlost.'
George Orwell (1903-1950), "1984"

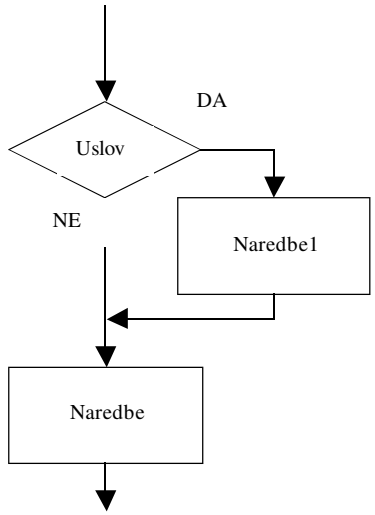
U većini realnih problema tok programa nije pravolinijski i jedinstven pri svakom izvođenjuGotovo uvijek se javlja potreba za grananjem toka, tako da se ovisno o postavljenom uslovu u jednom slučaju izvodi jedan dio programa, a u drugom slučaju drugi dio. Na primjer, želimo izračunati realne korijene kvadratne jednačine. Prvo ćemo izračunati diskriminantu - ako je diskriminanta veća od nule, izračunat ćemo oba korijena, ako je jednaka nuli izračunat ćemo jedini korijen, a ako je negativna ispisat ćemo poruku da jednadžba nema realnih korijena. Grananja toka i ponavljanja dijelova kôda omogućavaju posebne naredbe za kontrolu toka.

7.2. Grananje toka naredbom `if`

Naredba `if` omogućava uslovno grananje toka programa ovisno o tome da li je ili nije zadovoljen uslov naveden iza ključne riječi `if`.

Jednostruki izbor

Ovo je najjednostavniji oblik naredbe za uslovno grananje je. Ako je ispunjen uslov izvršit će se naredbe “Naredbe1”, a ako taj uslov nije ispunjen te naredbe se preskaču.




```
if ( logički_izraz )
    // blok_naredbi
```

Ako je vrijednost izraza iza riječi `if` logička istina (tj. bilo koji broj različit od nule), izvodi se blok naredbi koje slijede iza izraza. U protivnom se taj blok preskače i izvođenje nastavlja od prve naredbe iza bloka. Na primjer:

```
if (a < 0) {
```

```
        cout << "Broj a je negativan!" << endl;
    }
```

U slučaju da blok sadrži samo jednu naredbu, vitičaste zagrade koje omeđuju blok mogu se i izostaviti, pa smo gornji primjer mogli pisati i kao:

```
    if (a < 0)
        cout << "Broj a je negativan!" << endl;
```

ili

```
    if (a < 0) cout << "Broj a je negativan!" << endl;
```

Zbog preglednosti kôda i nedoumica koje mogu nastati prepravkama, početniku preporučujemo redovitu upotrebu vitičastih zagrada i pisanje naredbi u novom retku.

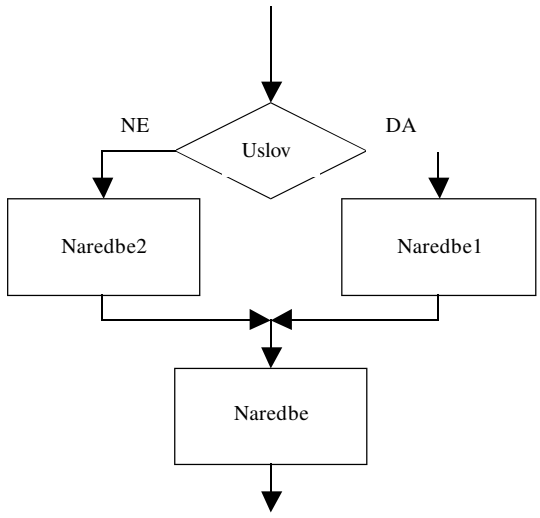
U protivnom se može dogoditi da nakon dodavanja naredbe u blok programer zaboravi omeđiti blok zagradama i time dobije nepredviđene rezultate:

```
    if (a < 0)
        cout << "Broj a je negativan!" << endl;
        cout << "Njegova apsolutna vrijednost je " << -a
             << endl;
```

Druga naredba ispod `if` uslova izvest će se i za pozitivne brojeve, jer više nije u bloku, pa će se za pozitivne brojeve ispisati pogrešna apsolutna vrijednost! Inače, u primjerima ćemo izbjegavati pisanje vitičastih zagrada gdje god je to moguće radi uštede na prostoru.

Dvostruki izbor

Želimo li da se ovisno u rezultatu izraza u `if` uslovu izvode dva nezavisna programska odsječka, primijenit ćemo sljedeći oblik uslovnog grananja:




```
if ( logički_izraz )
    // prvi_blok_naredbi
else
    // drugi_blok_naredbi
```

Kod ovog oblika, ako izraz u `if` uslovu daje rezultat različit od nule, izvest će se prvi blok naredbi. Po završetku bloka izvođenje programa nastavlja se od prve naredbe iza drugog bloka. Ako izraz daje kao rezultat nulu, preskače se prvi blok, a izvodi samo drugi blok naredbi, nakon čega se nastavlja izvođenje naredbi koje slijede. Evo jednostavnog primjera u kojem se računaju presjecišta pravca s koordinatnim osima. Pravac je zadan jednadžbom $ax + by + c = 0$.

```
#include <iostream.h>


int main() {
    float a, b, c;           // koeficijenti pravca
    cin >> a >> b >> c;      // učitaj koeficijente
    cout << "Koeficijenti: " << a << ","
         << b << "," << c << endl; ispiši ih


    cout << "Presjecište s apscisom: ";
    if (a != 0)
```

```
        cout << -c / a << ", ";
    else
        cout << "nema, "; // pravac je horizontalan


    cout << "presjecište s ordinatom: ";
    if (b != 0)
        cout << -c / b << endl;
    else
        cout << "nema" << endl;// pravac je vertikalan


         return 0;
     }
```

Blokovi `if` naredbi se mogu nadovezivati:

```
if ( logički_izraz1 )
    // prvi_blok_naredbi
else if ( logički_izraz2 )
    // drugi_blok_naredbi
else if ( logički_izraz3 )
    // treći_blok_naredbi
...
else
    // zadnji_blok_naredbi
```

Ako je *logički_izraz1* točan, izvest će se prvi blok naredbi, a zatim se izvođenje nastavlja od prve naredbe iza zadnjeg `else` bloka u nizu, tj. iza bloka *zadnji_blok_naredbi*. Ako *logički_izraz1* nije točan, izračunava se *logički_izraz2* i ovisno o njegovoj vrijednosti izvodi se *drugi_blok_naredbi*, ili se program nastavlja iza njega. Ilustrirajmo to primjerom u kojem tražimo realne korijene kvadratne jednačine:

```
#include <iostream.h>


int main() {
    float a, b, c;


    cout << "Unesi koeficijente kvadratne jednačine:"
          << endl;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> c;


      float diskr = b * b - 4. * a * c;          //
diskriminanta


    cout << "Jednadžba ima ";
    if (diskr == 0)
         cout << "dvostruki realni korijen." << endl;
    else if (diskr > 0)
         cout << "dva realna korijena." << endl;
    else
         cout << "dva kompleksna korijena." << endl;
```

```
    return 0;
}
```

Blokovi `if` naredbi mogu se ugnježđivati jedan unutar drugoga. Ilustrirajmo to primjerom u kojem gornji kôd poopćujemo i na slučajeve kada je koeficijent *a* kvadratne jednačine jednak nuli, tj. kada se kvadratna jednadžba svodi na linearnu jednadžbu:

```
#include <iostream.h>

int main() {
    float a, b, c;

    cout << "Unesi koeficijente kvadratne jednadzbe:"
         << endl;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> c;
    if (a) {
        float diskr = b * b - 4. * a * c;
        cout << "Jednadžba ima ";
        if (diskr == 0)
            cout << "dvostruki realni korijen." <<
endl;
        else if (diskr > 0)
            cout << "dva realna korijena." << endl;
```

```
            else
                cout << "kompleksne korijene." << endl;
        }
        else
            cout << "Jednadžba je linearna." << endl;

        return 0;
    }
```

Za logički izraz u prvom `if` uslovu postavili smo samo vrijednost varijable `a` - ako će ona biti različita od nule, uslov će biti zadovoljen i izvest će se naredbe u prvom `if` bloku. Taj blok sastoji se od niza `if-else` blokova identičnih onima iz prethodnog primjera. Ako početni uslov nije zadovoljen, tj. ako varijabla `a` ima vrijednost 0, preskače se cijeli prvi blok i ispisuje poruka da je jednadžba linearna. Uočimo u gornjem primjeru dodatno uvlačenje ugniježđenih blokova.

Pri definiranju logičkog izraza u naredbama za kontrolu toka početnik treba biti oprezan. Na primjer, želimo li da se dio programa izvodi samo za određeni opseg vrijednosti varijable `b`, naredba

```
if (-10 < b < 0) //...
```

neće raditi onako kako bi se prema ustaljenim matematičkim pravilima očekivalo. Ovaj logički izraz u biti se sastoji od dvije usporedbe: prvo se ispituje je li -10 manji od `b`, a potom se rezultat te usporedbe uspoređuje s 0, tj.

```
if ((-10 < b) < 0) //...
```

Kako rezultat prve usporedbe može biti samo `true` ili `false`, odnosno 1 ili 0, druga usporedba dat će uvijek logičku neistinu. Da bi program poprimio željeni tok, usporedbe moramo razdvojiti i logički izraz formulirati ovako: "ako je -10 manji od `b` *i* ako je `b` manji od 0":

```
if (-10 < b && b < 0) //...
```

Druga nezgoda koja se može dogoditi jest da se umjesto operatora za usporedbu ==, u logičkom izrazu napiše operator pridruživanja =. Na primjer:

```
if (k = 0)          // pridruživanje, a ne usporedba!!!
    k++;
else
    k = 0;
```

Umjesto da se varijabla `k` uspoređuje s nulom, njoj se u logičkom izrazu pridjeljuje vrijednost 0. Rezultat logičkog izraza jednak je vrijednosti varijable `k` (0 odnosno `false`), tako da se prvi blok naredbi nikada ne izvodi. Bolji kompajler će na takvom mjestu korisniku prilikom prevođenja dojaviti upozorenje.

7.3. Uslovni operator `? :`

Iako ne spada među naredbe za kontrolu toka programa, uslovni operator po strukturi je sličan `if-else` bloku, tako da ga je zgodno upravo na ovom mjestu predstaviti. Sintaksa operatora uslovnog pridruživanja je:

```
uslov ? izraz1 : izraz2 ;
```

Ako izraz *`uslov`* daje logičku istinu, izračunava se *`izraz1`*, a u protivnom *`izraz2`*. U primjeru

```
x = (x < 0) ? -x : x;        // x = abs(x)
```

ako je `x` negativan, izračunava se prvi izraz, te se varijabli `x` na lijevoj strani znaka jednakosti pridružuje njegova pozitivna vrijednost, tj. varijabla `x` mijenja svoj predznak. Naprotiv, ako je `x` pozitivan, tada se izračunava drugi izraz i varijabli `x` na lijevoj strani pridružuje njegova nepromijenjena vrijednost.
Izraz u uslovu mora kao rezultat vraćati aritmetički tip ili pokazivač. Alternativni izrazi desno od znaka upitnika moraju davati rezultat istog tipa ili se moraju dati svesti na isti tip preko ugrađenih pravila konverzije.

Uslovni operator koristite samo za jednostavna ispitivanja kada naredba stane u jednu liniju. U protivnom kôd postaje nepregledan.

Koliko čitaoca odmah shvaća da u sljedećem primjeru zapravo računamo korijene kvadratne jednačine?

```
((diskr = b * b -4 * a * c) >= 0) ?
    (x1 = (-b + diskr) / 2 / a, x2 = (-b + diskr) / 2 /
a)  :
    (cout << "Ne valja ti korijen!", x1 = x2 = 0);
```

7.4. Grananje toka naredbom `switch`

Kada izraz uslova daje više različitih rezultata, a za svaki od njih treba provesti različite odsječke programa, tada je umjesto `if` grananja često preglednije koristiti `switch` grananje. Kod tog grananja se prvo izračunava neki izraz koji daje cjelobrojni rezultat. Ovisno o tom rezultatu, tok programa se preusmjerava na neku od grana unutar `switch` bloka naredbi. Opšta sintaksa `switch` grananja izgleda ovako:

```
switch ( cjelobrojni_izraz ) {
  case konstantan_izraz1 :
    // prvi_blok_naredbi
  case konstantan_izraz2 :
    // drugi_blok_naredbi
    break;
  case konstantan_izraz3 :
  case konstantan_izraz4 :
    // treći_blok_naredbi
    break;
  default:
    // četvrti_blok_naredbi
}
```

Prvo se izračunava *cjelobrojni_izraz*, koji mora davati cjelobrojni rezultat. Ako je rezultat tog izraza jednak nekom od konstantnih izraza u `case` uslovima, tada se izvode sve naredbe koje slijede pripadajući `case` uslov sve do prve `break` naredbe. Nailaskom na `break` naredbu, izvođenje kôda u `switch` bloku se prekida i nastavlja se od prve naredbe iza `switch` bloka. Ako `izraz` daje rezultat koji nije naveden niti u jednom od `case` uslova, tada se izvodi blok naredbi iza ključne riječi `default`. Razmotrimo tokove programa za sve moguće slučajeve u gornjem primjeru. Ako *cjelobrojni_izraz* kao rezultat daje:

- *konstantan_izraz1*, tada će se prvo izvesti *prvi_blok_naredbi*, a zatim *drugi_blok_naredbi*. Nailaskom na naredbu `break` prekida se izvođenje naredbi u `switch` bloku. Program iskače iz bloka i nastavlja od prve naredbe iza bloka.

- *konstantan_izraz2*, izvodi se *drugi_blok_naredbi*. Nailaženjem na naredbu `break` prekida se izvođenje naredbi u `switch` bloku i program nastavlja od prve naredbe iza bloka.

- *konstantan_izraz3* ili *konstantan_izraz4* izvodi se *treći_blok_naredbi*. Naredbom `break` prekida se izvođenje naredbi u `switch` bloku i program nastavlja od prve naredbe iza bloka.

- Ako rezultat nije niti jedan od navedenih *konstantnih_izraza*, izvodi se *četvrti_blok_naredbi* iza `default` naredbe.

Evo i konkretnog primjera `switch` grananja (algoritam je prepisan iz priručnika za jedan stari programirljivi kalkulator):

```
#include <iostream.h>


int main() {
    cout << "Upiši datum u formatu DD MM GGGG:";
    int dan, mjesec;
    long int godina;
    cin >> dan >> mjesec >> godina;


    long datum;
    if (mjesec < 3) {
        datum = 365 * godina + dan + 31 * (mjesec - 1)
                + (godina - 1) / 4
                - 3 * ((godina - 1) / 100 + 1) / 4;
    }
    else {
        // uočimo operator dodjele tipa (int):
        datum = 365 * godina + dan + 31 * (mjesec - 1)
                - (int)(0.4 * mjesec + 2.3) + godina / 4
                - 3 * (godina / 100 + 1) / 4;
    }


    cout << dan << "." << mjesec << "." << godina
         << ". pada u ";
```

```
    switch (datum % 7) {
      case 0:
        cout << "subotu." << endl;
        break;
      case 1:
        cout << "nedjelju." << endl;
        break;
      case 2:
        cout << "ponedjeljak." << endl;
        break;
      case 3:
        cout << "utorak." << endl;
        break;
      case 4:
        cout << "srijedu." << endl;
        break;
      case 5:
        cout << "četvrtak." << endl;
        break;
      default:
        cout << "petak." << endl;
    }
    return 0;
}
```

Algoritam se zasniva na cjelobrojnim dijeljenjima, tako da sve varijable treba deklarirati kao cjelobrojne. štaviše, `godina` se mora definirati kao `long int`, jer se ona u računu množi s 365. U protivnom bi vrlo vjerojatno došlo do brojčanog preljeva, osim ako bismo se ograničili na datume iz života Kristovih suvremenika. Uočimo u gornjem kôdu operator dodjele tipa `(int)`

```
/*...*/  (int)(0.4 * mjesec + 2.3)  /*...*/
```

kojim se rezultat množenja i zbrajanja brojeva s pomičnim zarezom pretvara u cijeli broj, tj. odbacuju decimalna mjesta. U `switch` naredbi se rezultat prethodnih računa normira na neki od sedam dana u tjednu pomoću operatora `%` (*modulo*).
Blok `default` se smije izostaviti, ali ga je redovito zgodno imati da bi kroz njega program prošao za vrijednosti koje nisu obuhvaćene `case` blokovima. Ovo je naročito važno tijekom razvijanja programa, kada se u `default` blok može staviti naredba koja će ispisivati upozorenje da je *`cjelobrojni_izraz`* u `switch` poprimio neku nepredviđenu vrijednost.

Naprimjer, pretpostavimo da ucitavamo dva operanda i operator i pohranjujemo ih u promjenljive operand1, operand2 i operator. Slijedeći kod izvršava zadatu operaciju i pohranjuje rezultat u promjenljivu rezultat.

```
switch (operator) {
                                        case '+':     resultat = operand1 +
operand2;
                                                                    break;
                                        case '-':     resultat = operand1 -
operand2;
                                                                    break;
                                        case '*':     resultat = operand1 *
operand2;
                                                                    break;
                                        case '/':     resultat = operand1 /
operand2;
                                                                    break;
                                         default:          cout << "Nepoznat
operator: " << ch << '\n';
                                                                    break;
}
```

Kao što ilustruje prethodni primjer, uobičajeno je da iskazi switch naredbe sadrže break. Ipak, postoje situacije kada ima smisla imati case bez iskaza break. Npr, ako u prethodnom primjeru želimo da i operator „x“ omogučava množenje, kod bi bio slijedeći:

```
switch (operator) {
                                        case '+':     resultat = operand1 +
operand2;
                                                                    break;
                                        case '-':     resultat = operand1 -
operand2;
                                                                    break;
                                        case 'x':
```

```
                                                    case '*':      resultat = operand1 *
operand2;

                                                                                  break;
                                                    case '/':      resultat = operand1 /
operand2;

                                                                                  break;
                                                    default:           cout << "Nepoznat
operator: " << ch << '\n';

                                                                                  break;
}
```

Pošto case ‘x’ nema break, kada je ovaj uslov zadovoljen, izvršavanje se nastavlja izvršavanjem iskaza slijedećeg case iskaza, te se vrši množenje.

Ovaj primjer može se, naravno, napisati i pomoću iskaza „if“.

8. ISKAZI PONAVLJANJA

8.1.Iskaz WHILE

Iskaz While (nazivamo ga i while petljom) omogučava ponavljanje iskaza sve dok je zadovoljen uslov, To je jedan od tri iskaza ponavljanja koje nudi C++. Opšti oblik iskaza je:

```
while (uslov)
                                                    iskazi;
```

Najprije se izračunava vrijednost uslova. Ako je rezultat različit od nule, izvršavaju se iskazi i cijeli proces se ponavlja. Inače se izlazi iz petlje.

Pretpostavimo da želimo izračunati sumu svih brojeva od 1 do neke cjelobrojne vrijednosti definisane kao n. Kod koji bi odgovarao ovome bio bi:

```
i = 1;
suma = 0;
while (i <= n)
                                                suma += i++;
```

Posmatrajmo šta se dešava prilikom izvršavanja ovog koda

Iteracija	i	n	i <= n	suma += i++
1	1	5	1	1
2	2	5	1	3
3	3	5	1	6
4	4	5	1	10
5	5	5	1	15
Sixth	6	5	0	

8.2.Iskaz DO

Iskaz DO (zovemo ga i do petlja) sličan je iskazu while, osim što se najprije izvršava tijelo petlje a zatim provjerava uslov. Opšti oblik petlje je

```
                                                    do
                                                    iskazi;
                                                while (uslov);
```

Najprije se izvršavaju iskazi, a zatim testira uslov. Ako uslov ima vrijednost različitu od nule (neistina), cijeli proces se ponavlja. Inače, petlja se završava.

Do petlja se rjeđe koristi od while petlje. Korisna je u situacijama kada trebamo da se tijelo petlje izvrši najmanje jednom, neovisno o vrijednosti uslova. Naprimjer, želimo li da ponavljamo učitavanje broja i računamo njegov kvadrat, i zaustavimo se kada je učitana nula, koristili bismo slijedeći kod:

```
do {
```

```
                                        cin >> n;
                                    cout << n * n << '\n';
} while (n != 0);
```

8.3.Iskaz FOR

Iskaz for (nazivamo ga i for petlja) sličan je while petlji, ali ima dvije dodatne komponente: izraz koji se izračunava samo jednom prije svega ostalog (inicijalizacija), i izraz koji se izračunava na kraju svake iteracije (akcija). Opšti oblik iskaza je:

for (*inicijalizacija* ; uslov; akcija)
iskazi;

Najprije se inicjalizuje vrijednost promjenljive koja ima ulogu brojača ponavljanja petlje – ***inicijalizacija***. Pri svakom prolasku kroz petlju provjerava se tačnost ***uslova***, tj. Izračunava njegova logička vrijednost. Ako je rezultat različit od nule (tačno), izvršavaju se iskazi petlje i mijenja vrijednost brojača izvršavanjem ***akcije***. Inače, petlja završava izvršavanje.

Ako poredimo for iskaz sa iskazom while, onda je while iskaz koji odgovara for iskazu slijedeći:

iskaz$_1$;
while (uslov)
{
iskaz;
iskaz$_3$;
}

For najčešće koristimo u situacijama kada se promjenljive inkrementiraju ili dekrementiraju u svakom prolasku kroz petlju. Naprimjer, slijedeći kod računa sumu cijelih brojeva od 1 do n.

```
suma = 0;
for (i = 1; i <= n; ++i)
                                        suma += i;
```

C++ dozvoljava deklaraciju promjenljive korištene uz prvi iskaz. Npr.

```
for (int i = 1; i <= n; ++i)
                                        suma += i;
```

što je ekvivalentno kodu:

```
int i;
for (i = 1; i <= n; ++i)
                                        suma += i;
```

Pošto su petlje iskazi, one mogu biti ugniježdene u druge petlje. Npr.

```
for (int i = 1; i <= 3; ++i)
                                for (int j = 1; j <= 3; ++j)
                                                cout << '(' << i << ',' << j
<< ")\n";
```

generiše slijedeći izlaz

```
(1,1)                    (1,2)                    (1,3)
                         (2,1)                    (2,2)
                         (2,3)                    (3,1)
                         (3,2)                    (3,3)
```

Da li?

8.4.Iskaz CONTINUE

Iskaz continue prekida tekuću iteraciju petlje i "skače" na slijedeću iteraciju. Pogrešno je pozivati ga izvan petlje.

U petljama while i do, slijedeća iteracija ovisi o uslovu petlje. U for petlji, slijedeća iteracija ovisi o trećem iskazu. Npr, petlja u kojoj se ponavlja čitanje broja, učitava brojeve, ali ignoriše negativne brojeve te prekida izvršavanje petlje kada je broj nula, imala bi slijedeći kod:

```
                                    do {
                                            cin >> broj;
                                       if (broj < 0) continue;
                                    // ovdje se broj obradjuje...

}

while (broj != 0);
```

Ovo je ekvivalentno kodu:

```
do {
                                          cin >> broj;
                                        if (broj >= 0) {
                                                  // ovdje se broj obradjuje...
                                               }
} while (num != 0);
```

Slijedeća varijanta ove petlje loop čita broj tačno n puta (umjesto dok se učita nula)

```
for (i = 0; i < n; ++i)
{
                                          cin >> broj;
                                    if (broj < 0) continue;
                                                  // uzrokuje skoj na: ++i
                                    // ovdje se broj obradjuje...
}
```

Ako se iskaz continue pojavljuje unutar ugniježdene petlje, primjenjuje se na petlju unutar koje se nalazi, a ne na vanjsku petlju. Npr. U slijedećem kodu continue se odnosi na for petlju, a ne na while petlju:

```
while (jos) {
                                  for (i = 0; i < n; ++i) {
                                                              cin >> broj;
                                                    if (broj < 0) continue;

                                   // causes a jump to: ++i
                                                                  // naredbe...
                                              }
                                          //etc...
```

```
}
```

8.5.Iskaz BREAK

Iskaz break može se pojaviti unutar petlji (while, do ili for) ili iskazu switch. Uzrokuje skok izvan petlje ili switch naredbe i prekid njenog izvršavanja. Kao i continue, break se odnosi samo na petlju ponavljanja ili switch kojim je uokvirena. Korištenje iskaza break izvan petlji ili switch iskaza uzrokuje grešku.

Naprimjer, pretpostavimo da želimo čitati lozinku, ali broj pokušaja želimo ograničiti:

```
for (i = 0; i < broj_pokusaja; ++i)
{
                              cout << "Unesite password: ";
                                    cin >> password;
                                 if (Provjeri(password))   // check password
for correctness
                                                                        break;

                                                                 // drop out of the loop
                                  cout << "Pogresno!\n";
}
```

Ovdje je Provjeri funkcija koja provjerava lozinku i vraća istinu ili laž, a može bit ii izraz. Ovakvo provjeravanje možemo izvršiti i bez break, kao što slijedi. Korištenje break iskaza je očigledno efikasnije.

```
ispravno = 0;
for (i = 0; i < Broj_provjeravanja && !provjera; ++i) {
                              cout << "Unesite password: ";
                                    cin >> password;
                             ispravno = Provjeri(password));
                                     if (!ispravno)
                                                          cout << "Greska!\n";
}
```

8.6.Iskaz GOTO

Iskaz goto nudi najniži nivo “skakanja”. Opšti oblik je:

`goto` *label;*

gdje je label identifikator koji označava destinaciju iskaza goto – broj linije na koju treba da “skoči” program. Labela treba da se završi dvotačkom, a navodi se prije iskaza.

Npr. Umjesto break u prethodnom primjeru, možemo koristiti goto:

```
for (i = 0; i < broj_pokusaja; ++i)
{
                              cout << "Unesite password: ";
                                    cin >> password;
                                 if (Provjeri(password))   // check password
for correctness
                                                                      goto out;

                                 // drop out of the loop
                                  cout << "Pogresno!\n";
}
out:
  //etc...
```

Pošto goto omogučava slobodnu, nestruktuiranu formu “skakanja”, može lako uzrokovati greške. Većina današnjih programera izbjegava ovaj iskaz u ime “jednostavnog, čitljivog” koda. Ipak, ovaj iskaz još uvijek ima svoju primjenu.

8.7. Iskaz RETURN

Iskaz return omogučava funkciji da vrati vrijednost u strukturu iz koje je pozvana. Opšti oblik je:

```
return izraz;
```

gdje je izraz vrijednost koju funkcija vraća. Tip ove vrijednosti mora odgovarati tipu funkcije. Ako funkcija ne vraća vrijednost (void) izraz je prazan, te je u tom slućaju iskaz return u formi:

```
return;
```

Do sada smo koristili jedino funkciju main koja vraća tip int. Povratna vrijednost funkcije main je namjenjena operativnom sistemu kada se završava izvršavanje programa.

8.8. Vježbe:

1. Napišite program koji učitava težinu (u kilogramima) i visinu (u centimetrima) i ispisuje poruku: debeo, normalan ili mrsav, korištenjem slijedećih kriterija

   a. Mršav težina < visina/2.5

   b. Normalan visina/2.5 <=tezina <= visina/2.3

   c. Debeo: visina/2.3 < tezina

```
#include <iostream.h>
int main ()
{
                                    double visina, tezina;

                          cout << "Vasa visina (u centimeterima): ";
                                       cin >> visina;
                           cout << "Vasa teyina (u kilogramima): ";
                                       cin >> tezina;

                                   if (tezina < visina/2.5)
                                                          cout << "Mrsav\n";
                        else if (visina/2.5 <= tezina && tezina <= visina/2.3)
                                                       cout << "Normalno\n";
                                          else
                                                          cout << "Debeo\n";

                                                return 0;
}
```

2. Pretpostavimo da je n=20; šta će izvršiti ovaj kod:

```
if (n >= 0)
                                              if (n < 10)
                                                          cout << "n is small\n";
else
                                     cout << "n is negative\n";
```

3. Napišite program koji učitava datum u formatu dd/mm/yy i ispisuje ga u formatu “mjesec dd, yyyy”. Npr. Ulaz “25/12/64” postaje:

   Decembar 25, 1961

```
#include <iostream.h>

int main ()
{
                                    int day, month, year;
                                          char ch;

                                cout << "Input a date as dd/mm/yy: ";
                               cin >> day >> ch >> month >> ch >> year;

                                     switch (month) {
                                                                    case 1:
                                                                cout << "January";
                                          break;
                                                                    case 2:
                                                               cout << "February";
                                          break;
                                                                    case 3:
                                                             cout << "March";
                                                                     break;
                                                                    case 4:
                                                             cout << "April";
                                                                     break;
                                                                    case 5:
                                                               cout << "May";
                                                                     break;
                                                                    case 6:
                                                              cout << "June";
                                                                     break;
                                                                    case 7:
                                                              cout << "July";
                                                                     break;
                                                                    case 8:
                                                            cout << "August";
                                          break;
                                                                    case 9:
                                                    cout << "September";break;
                                                                   case 10:
                                    cout << "October";
                                          break;
                                                                   case 11:
                                    cout << "November";
                                          break;
                                                                   case 12:
                                    cout << "December";
                                          break;
                                             }
                          cout << ' ' << day << ", " << 1900 + year << '\n';
                                        return 0;
}
```

4. Napišite program koji učitava cio broj, provjerava da li je pozitivan, te ispisuje faktorijel, korištenjem formule:

```
Faktorijel(0) +1
Faktorijel(n) = n x faktorijel (n-1)

#include <iostream.h>
```

```
int main ()
{
                                        int n;
                                    int factorial = 1;

                                cout << "Input a positive integer: ";
                                        cin >> n;

                                    if (n >= 0) {
                                                                    while (n > 0)

                                    factorial *= n--;
            cout << "Factorial of " << n << " = " << factorial << '\n';
        }
                                        return 0;
}
```

5. Napisati program koji učitava oktalni broj i ispisuje odgovarajući decimalni broj. Npr.

```
Unesite oktalni broj: 214
(214)8= (532)10
```

```
#include <iostream.h>

int main ()
{
                                    int octal, digit;
                                    int decimal = 0;
    int                                power = 1;

                                cout << "Input an octal number: ";
                                        cin >> octal;

                                for (int n = octal; n > 0; n /= 10) {   // process
each digit
                                                                digit = n % 10;


                                                                        // right-most digit
                                                            decimal = decimal + power *
digit;
                                                                    power *= 8;
    }
            cout << "Octal(" << octal << ") = Decimal(" << decimal << ")\n";
                                        return 0;
}
```

6. Napišite program koji ispisuje tabelu slijedećeg formata:

```
1 x 1 = 1
1 x 2 = 2
...
9 x 9 = 81
#include <iostream.h>

int main ()
{
                        for (register i = 1; i <= 9; ++i)
                                        for (register j = 1; j <= 9; ++j)

cout << i << " x " << j << " = " << i*j << '\n';

                                    return 0;
}
```

8.9.Zadaci za samostalan rad

1. Učitati broj učenika jednog razreda i zatim težine svakog učenika-ce.Uz težinu treba ucitati i pol (M ili Z). Izračunati srednju težinu učenika-ce posebno.

2. Jedna bakterija podijeli se na dvije. U sljedećem ciklusu svaka od dvije nastale bakterije podijeli se ponovo itd. Nakon koliko ciklusa će nastati više od 10000 bakterija?

3. Napisati program koji će na ekranu ispisati sve cijele brojeve između D i G. Iznose cijelih brojeva D i G korisnik unosi na početku programa

4. Učitati cijeli broj n iz intervala 1-80. U slučaju da n nije između 1 i 80, učitavanje treba ponavljati sve dok učitani broj ne bude u zadanim granicama. Program treba ispisati n zvjezdica u jednom redu.

5. Napisati program kojim se računa zbir sljedeceg reda 2+4+6+8+…n (prirodan broj ucitati na početku programa).

6. Mate je u banku uložio ušteđevinu od 5500 eura. Ako je godišnja kamata 6,5% napisati program koji ce izračunati nakon koliko godina će imati ušteđevinu od 12000 eura. Ispisati izračunate godine i tačan iznos u eurima

7. Profesor ispravlja n testova učenika koji pripadaju 3t1 i 3t2 razredu. Potrebno je napisati program koji će ucitati ocjene n testova. Uz svaku ocjenu testa treba ucitati i oznaku razreda kojem test pripada. Cilj programa je izračunati srednju ocjenu posebno za razred 3t1 a posebno za razred 3t2

8. Napisati program kojim se računa zbir sljedeceg reda 1+1/3+1/5+1/7+…1/n (prirodan broj ucitati na početku programa).

9. Ispisati sve parne brojeve između 1 i 200. Izračunati srednju vrijednost tih brojeva

10. Napisati program koji ispisuje tablicu vrijednosti funkcija sinX i cosX za vrijednosti varijable X iz intervala od 0 do 5 u koracima po 0,25. Tablica mora imati ovakvo zaglavlje: X sin(x) cos(x)

11. Napisati program koji ucitava n parova brojeva, a zatim za svaki ucitani par ispisuje manji. Takodjer, treba odrediti koja je srednja vrijednost ucitanih brojeva.

12. Učitati dva prirodna broja i izvršite cjelobrojno dijeljenje uzastopnim oduzimanjem. Upustvo: Dijeljenje se može svesti na oduzimanje tako da od dijeljenika uzastopce oduzimamo djeljitelj sve dok ne dobijemo rezultat manji od djeljitelja. Broj oduzimanja je traženi rezultat.

13. Napisati program koji ce omogućiti korisniku da upisuje cijele brojeve toliko dugo dok ne upiše broj 0. Nakon toga program treba ispisati koliko je ucitano parnih a koliko neparnih brojeva. Broj 0 se ne računa.

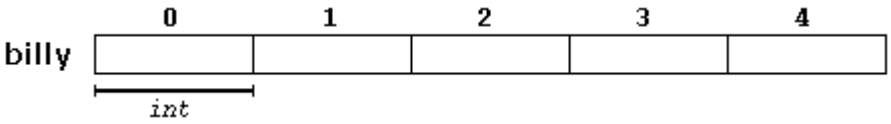
14. U kutiji je N otpora. Ucitajte broj otpora i zatim iznose otpora. Pri tome treba izracunati ukupan otpor u slucaju da si ti otpori spojeni a) serijski ; b) paralelno –

15. Napisati program koji ucitava n realnih brojeva i formira njihovu sumu. Takodjer, program broji koliko je medju ucitanim brojevima pozititvnih, i koliko je negativnih. Ispisuje se suma ucitanih brojeva, broj pozitivnih i broj negativnih u zasebnim redovima

9. NIZOVI

Niz je skup elemenata istog tipa. Elementi niza memorišu se u nizu susjednih mamorijskih lokacija. Svakom pojedinačnom elementu niza može se pristupiti navođenjem jedinstvenog identifikatora (imena) niza i indeksa odgovarajućeg elementa niza.

Naprimjer, umjesto da deklarišemo pet zasebnih promjenljivih, pet vrijednosti tipa int možemo pohraniti u niz. Ovaj niz, nazovimo ga *billy*, možemo prikazati na slijedeći način:



gdje svaki prazan pravougaonik predstavlja **element niza**, koji je, u ovom slučaju, cijel broj (tip int). Ovi elementi su numerisani od 0 do 4, budući da je indeks prvog elementa niza uvijek nula (0), neovisno o dužini niza.

9.1. Deklaracija niza

Kao i sve druge varijable, niz mora biti deklarisan prije no što bude korišten. Tipična deklaracija niza je:

tip *ime_niza*[**broj_elemenata**]

gdje je

- tip – tip podataka elemenata niza (int, float, char...)
- ime_niza - valjano ime promjenljive
- broj_elemenata - maksimalan broj elemenata niza

Stoga, da bismo deklarisali niz `billy`, zadajemo naredbu:

int billy [5];

Napomena:

Parametar *broj_elemenata*, naveden u `[] u deklaraciji niza mora biti konstanta, budući da je niz statička struktura podataka čija veličina mora biti određena prije izvršavanja programa. Ako ipak želimo koristiti niz varijabilne dužine, moramo koristiti pokazivače, o čemu će biti govora kasnije.`

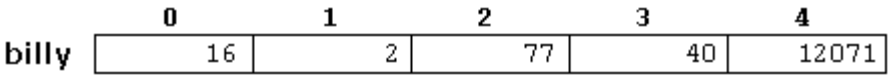
9.2. Inicijalizacija niza

Dodjelu početnih vrijednosti elementima niza možemo izvršiti tako što svakom elementu niza dodjelimo vrijednost pobrojanu unutar zagrada `{ }`.

Na primjer:

int billy [5] = { 16, 2, 77, 40, 12071 };

Ovakvom deklaracijom kreirali bismo niz kao na slici:



Broj vrijednosti naveden u zagradi ne smije biti veći od broja elemenata niza, koji smo naveli u uglastim zagradama `[ ]`.

Ako prilikom deklaracije niza izvršimo i inicijalizaciju elemenata niza, onda uglaste zagrade možemo ostaviti prazne. U tom slučaju, kompajler će pretpostaviti veličinu niza na osnovu broja inicijalnih vrijednosti koje su navedene u vitičastim zagradama.

```
int billy [] = { 16, 2, 77, 40, 12071 };
```

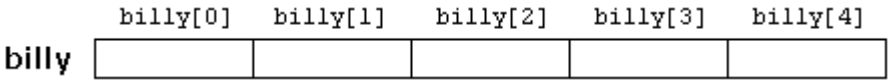
Nakon ove naredbe, niz billy bit će deklarisan kao niz od pet elemenata.

9.3. Pristup elementima niza

Unutar programa u kojem je deklarisan niz, možemo pristupiti bilo kom individualnom elementu niza. Elementima niza pristupamo navođenjem imena niza i indeksa elementa niza kojem želimo pristupiti. U opštem slučaju:

```
ime[index]
```

U prethodnom primjeru u kojem niz billy ima pet elemenata, pojedinačnim elementima pristupamo na slijedeći način:



Npr. da bismo dodijelili vrijednost 75 trećem elementu niza, zadajemo naredbu:

```
billy[2] = 75;
```

Ako vrijednost trećeg elementa niza želimo dodijeliti promjenljivoj a, zadajemo naredbu:

```
a = billy[2];
```

Primjetite da trećem elementu niza pristupamo navođenjem indeksa 2, tj. `billy[2]`, pošto je prvi element niza `billy[0]`, drugi je `billy[1]`, a zadnji `billy[4]`. Ako ipak u programu navedemo `billy[5]`, pristupit ćemo šestom elementu i prekoračiti deklarisani opseg niza. Npr, ako zadamo naredbu:

```
Cout << billy[5];
```

Ova naredba je sintaksno tačna. Međutim, ovo može biti problem za vrijeme izvođenja programa (runtime error).

Slijedi nekoliko primjera ispravnih naredbi sa elementima niza:

```
billy[0] = a;
```

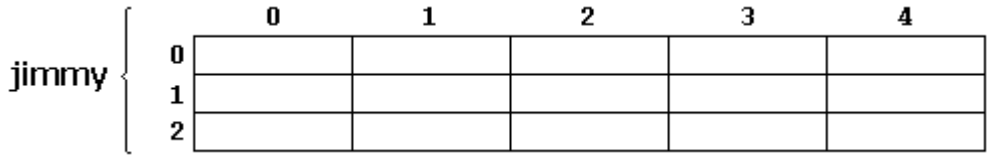
```
billy[a] = 75;
```

```
b = billy [a+2];
```

```
billy[billy[a]] = billy[2] + 5;
```

9.4. Višedimenzinalni nizovi

Višedimenzionalni niz može biti opisan kao “niz nizova”. Npr. dvodimenzionalni niz možemo zamisliti kao dvodimenzionalnu tabelu sačinjenu od elemenata istog tipa, kao na slici:

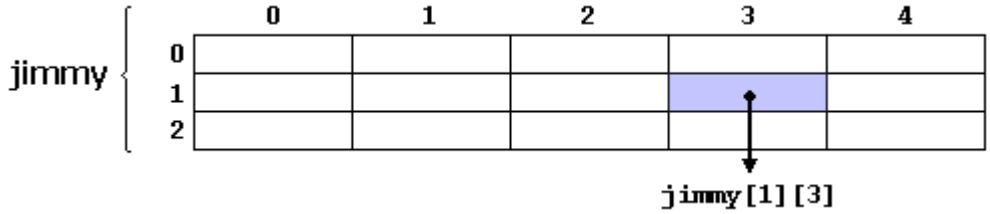


jimmy predstavlja dvodimenzionalni niz od 3 reda i 5 kolona. Da bismo ga deklarisali u C++, zadajemo naredbu:

int jimmy [3][5];

Da bismo pristupili elementu u drugom redu i četvrtoj koloni, koristimo:

jimmy[1][3]



(prvi element niza uvijek ima indeks 0).

Multidimenzionalni niz može imati onoliko dimenzija koliko je potrebno. Ali, oprez! Kapacitet memorije rapidno raste sa svakom novom dimenzijom. Npr. slijedećom naredbom

char century [100][365][24][60][60];

deklarišemo niz karaktera za svaku sekundu u jednom vijeku, što je više od 3 miliona karaktera. Stoga, ovakav niz može potrošiti više od 3 gigabajta!