

# PROGRAMSKI JEZICI 1



PresenterMedia

?



PresenterMedia



*Knjigu posvećujem svojoj porodici,  
uz posebnu zahvalnost Danici Milošević,  
diplomiranom filologu engleskog jezika  
i književnosti, za nesebičnu pomoć kod  
prevoda i lektorstva*

*Autor*

---

**Recenzenti:** Prof.dr Slobodan Obradović,  
*profesor Visoke elektrotehničke škole u Beogradu,*  
mr Hranislav Mihailović,  
*viši predavač Više tehničke škole u Nišu*

**Izdavač:** "SIIC" - Niš  
**Tehnička obrada:** Prof.dr Milošević Borivoje  
**Ilustracije:** Prof.dr Milošević Borivoje  
**Lektor i prevodi:** Milošević Danica, profesor engleskog  
jezika i književnosti  
**Štampa:** "SITOPRINT" - Niš 2013. god.

**СIP Каталогизација у публикацији  
Народна библиотека Србије, Београд**

004. 42. / .43

**МИЛОШЕВИЋ, Боривоје**  
**Programski jezici I / Milošević**  
**Borivoje. -Niš : SIIC, 2004 (Niš:**  
**Sitoprint). - 213 str. : ilustr. ; 24 cm**

**Tiraž 100.**

**ISBN 86-7178-085-6**

a) Програмирање b) Програмски језици  
**COBISS.SR - ID 117916428**

## 1. UVOD

### 1.1 UVOD U PROGRAMSKE JEZIKE

Programiranje u širem smislu je proces pripreme, razrade i pisanja programa u cilju rešavanja problema na elektronskom računaru.

Procesu programiranja predhode izvesne pripremene radnje i one zahtevaju:

- potpuno razmatranje i precizno formulisanje problema koji se rešava,
- izbor matematičkih metoda i postavljanje matematičkog modela,
- izbor i analiza numeričkih postupaka sa tačke gledišta efikasnog rešavanja datog problema na računaru.

Proces programiranja obuhvata sledeće faze:

- projektovanje i razradu algoritma,
- pisanje programa,
- testiranje programa,
- izradu dokumentacije.

U ovom poglavlju obuhvaćeni su programski jezici kao sredstvo za pisanje programa, dok su drugi aspekti programiranja razmatrani u narednom poglavlju.

Termin programiranje koji se koristi u ovom poglavlju odnosi se na programiranje u najužem smislu, tj. na pisanje (kodiranje) programa.

Pisanje programa podrazumeva poznavanje odgovarajućeg jezika koji se naziva programski jezik.

Jezik, kao sredstvo komunikacije između ljudi, je jedna od najvećih tekovina ljudske civilizacije. Prirodni jezici koji su u upotrebi u svakodnevnoj komunikaciji ljudi, nastali su iz potrebe za međusobnim sporazumevanjem u obavljanju različitih aktivnosti. Bogatstvo rečnika jednog prirodnog jezika zavisi od starosti i kulture naroda koji ga koristi, ali isto tako i od delatnosti kojima se taj narod bavi, kao i od klime i podneblja u kome živi. Na primer, u jeziku eskima postoji jako mnogo reči koje se odnose na sneg. Oni koriste posebne reči za novi, tvrdi, meki, ledeni, stari ili okopneo sneg.

Pored prirodnih postoje i *veštački* jezici koji su nastali kao unapred definisan skup pravila za sporazumevanje u određenoj oblasti.

Kao primer veštačkog jezika može da se uzme šatrovački jezik grupe dece koja u igri koriste neke svoje reči, poznate samo njima, koje se formiraju po pravilima koja su ona sama izmislila. Najpoznatiji primer veštačkog jezika je Esperanto koji je izmislio poljski lekar Zamenhof sa idejom da to bude univerzalni jezik planete. Jedan od najnovijih primera je i Klingonski jezik, definisan specijalno za potrebe TV serije Star Trek. Osnovan je čak i institut (Klingon Language Institute, Flourtown, USA) za proučavanje ovog jezika.

Jezik:

1. sistem izražavanja misli koji ima određena glasovna i gramatička pravila i služi kao glavno sredstvo za sporazumevanje među ljudima;
2. način sporazumevanja uopšte.

*Jezik* - skup znakova, dogovora i pravila koji se koriste za predavanje, saopštavanje informacija

*Prirodni jezik* - jezik čija se pravila zasnivaju na svakodnevnom korišćenju bez njihove eksplicitne definicije

*Veštački jezik* - jezik čija su pravila eksplicitno utvrđena pre njegovog korišćenja.

## 1.2. Definicija programskih jezika

Posebnu grupu veštačkih jezika čine *programski jezici*. Nastaju sa razvojem računara iz potrebe da se pojednostavi pisanje programa kojima se upravlja tokom obrade podataka na njima. Zbog toga kažemo da su programski jezici sredstvo komunikacije između ljudi i računara.

*Programski jezik* - Veštački jezik koji se koristi za pripremu programa za računar

*Programski jezik* - Jezik za izradu programa, sastavljen od simbola koje računar može da prevede u direktne radnje

*Algoritamski jezik* - Veštački jezik namenjen opisu algoritama

Prvi programi pisani su na *mašinskom jeziku (binarnom kodu)*, u kome se naredbe i podaci predstavljaju binarno, nizovima jedinica i nula. Primer 1.1 predstavlja kod programa od tri mašinske naredbe.

*Primer 1.1* Sekvenca od tri mašinske naredbe od po 32 bita.

0001	1101	1000	0000
0000	0000	1111	1111
0001	1100	0000	0000
0000	0000	1111	1100
0001	1110	0000	0000
0000	0000	0111	0010

U okviru mašinske naredbe svaka binarna cifra na određenom mestu ima unapred definisano značenje i odgovara upravljačkom signalu kojim se inicira određena aktivnost procesora. Sprega između mašinskog koda i procesora je direktna i neposredna. Međutim, programiranje u mašinskom kodu zahteva detaljno poznavanje strukture procesora kojim se upravlja i potpunu kontrolu nad smeštajem koda programa i podataka u operativnoj memoriji računara. Sve to postaje toliko komplikovano da je u praksi skoro neprimenljivo. U cilju jednostavnije upotrebe računara, već sa prvim

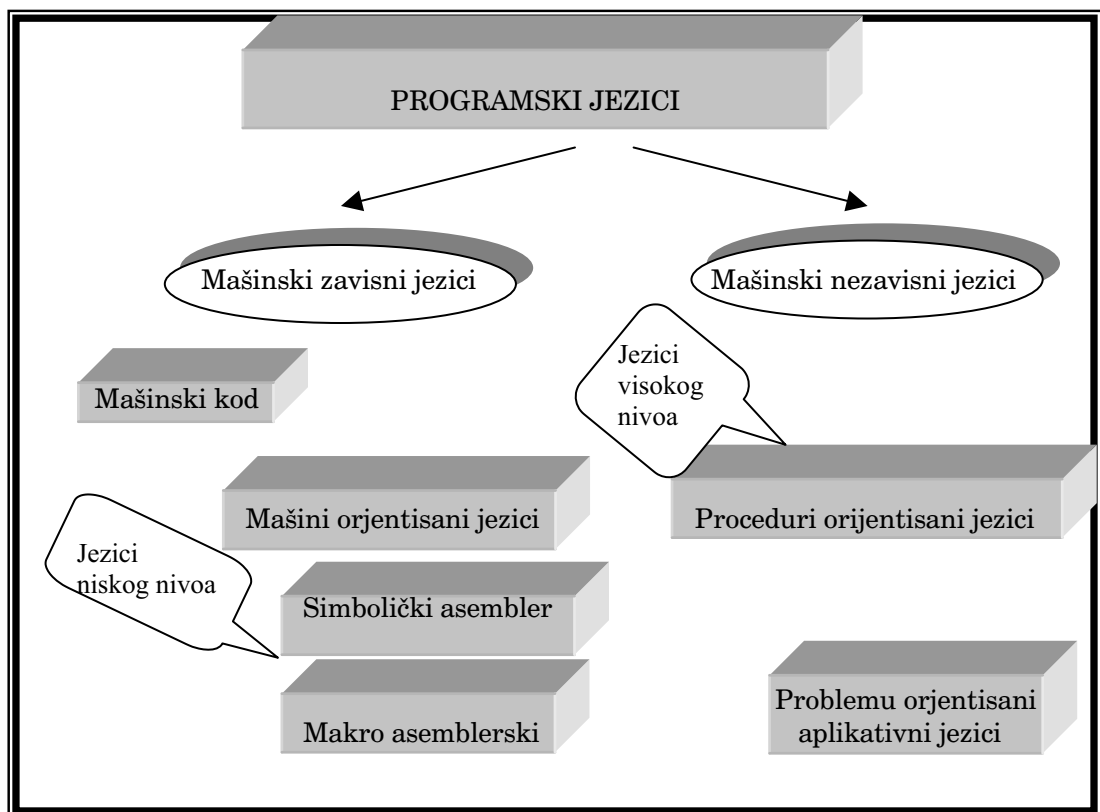
generacijama računara, počinju da se razvijaju simbolički jezici slični jezicima koje ljudi koriste u svom sporazumevanju.

*Jezik četvrte generacije ( Fourth Generation Language )* - Engleskom sličan jezik veoma visokog nivoa koji se koristi za programiranje i upite. Neproceduralni jezik.

*Jezik veoma visokog nivoa ( Very High Level Languages VHLL )* - Programski jezik sličan engleskom.

*Upitni jezik ( Query Language )* - Jezik visokog nivoa, sličan engleskom koji omogućava korisniku pretraživanje ( traženje ) informacija iz datoteke unošenjem proste komande.

Danas je u upotrebi jako mnogo programskih jezika od kojih neki imaju šire značenje i služe za definisanje programa na računarima opšte namene, dok su drugi namenjeni primeni u sasvim uskim oblastima ili na specijalizovanim računarskim sistemima. Ukupan broj programskih jezika je nemoguće tačno utvrditi. Procene se kreću od nekoliko stotina do nekoliko hiljada. Zbog svega toga pozabavićemo se najpre problemom klasifikacije programskih jezika i hronologijom njihovog razvoja.



Slika 1.3.1. Podela programskih jezika

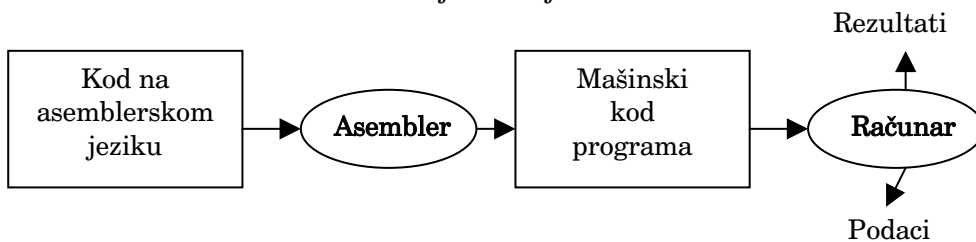
### 1.3. Klasifikacija programskih jezika po stepenu zavisnosti od računara

Po stepenu zavisnosti od računara razlikuju se *mašinski zavisni* i *mašinski nezavisni* programski jezici. U mašinski zavisne jezike svrstavaju se jezici koji su u direktnoj zavisnosti sa procesorom na kome se program izvršava. To su pre svega mašinski jezik i mašini orjentisani *asemblerski* i *makro asemblerski* jezici. Oni čine grupu *jezika niskog nivoa*. Grupu *mašinski nezavisnih jezika* čine *programski jezici visokog nivoa* ili *proceduri orjentisani* programski jezici i *programski jezici veoma visokog nivoa*, *aplikativni* ili *problemu orjentisani* programski jezici.

Hronologija razvoja ovako klasifikovanih programskih jezika bila bi u sledećem:

Prvi korak ka pojednostavljenju mašinskih jezika napravljen je korišćenjem heksadecimalnog koda za skraćeno zapisivanje binarnih nizova čime se dobija samo to da zapisi programa postaju četiri puta kraći. Za konverziju heksadecimalnog koda u binarni dovoljan je konvertor koda koji se jednostavno realizuje kao prekidačka mreža.

Sledeci korak u razvoju su asemblerski jezici. Za prevođenje asemblerskih jezika u mašinske koriste se programski moduli koji se nazivaju asembleri po čemu su i jezici dobili naziv. Slika 1.3.2. ilustruje funkciju asemblera.



Slika 1.3.2. Asembleri

Asemblerske naredbe su u suštini simbolički zapisane mašinske naredbe. Umesto binarnih nizova koriste se simbolička imena za predstavljanje pojedinih delova naredbi. Veliki korak napred u okviru asemblerskih jezika je korišćenje simboličkih imena umesto apsolutnih adresa naredbi i podataka. Primer 1.2 je ilustracija sekvence asemblerskih naredbi.

*Primer 1.2* *Primer asemblerskog koda kojim se sabiraju podaci A i B i rezultat pamti kao podatak C.*

	LD	\$R1.	<A
	ADD	\$R1.	<B
	STO	\$R1.	<C
A	DC	5	
B	DC	13	
C	DS		

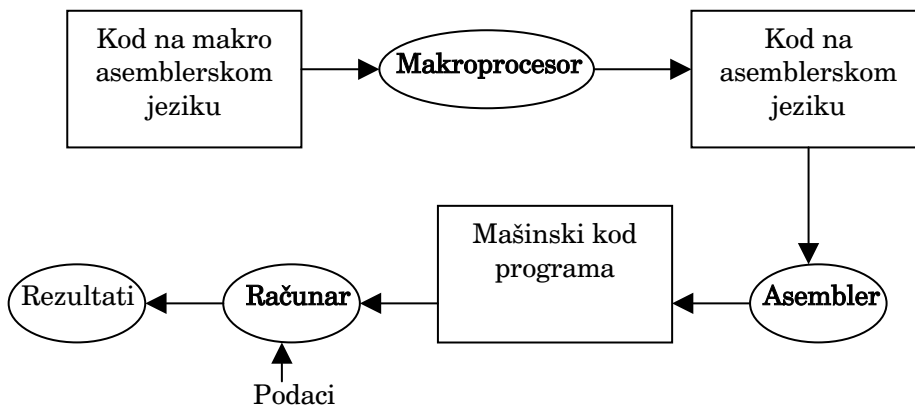
Tumačenje sekvence naredbi iz primera 1.2 bi bilo sledeće: Prvom naredbom se broj 5, smešten u memorijskoj lokaciji koja je u programu simbolički označena sa A, prenosi u registar R1 procesora. Drugom naredbom se sadržaj registra R1 sabira sa sadržajem memorijske lokacije imenovane sa B (brojem 13). Rezultat sabiranja ostaje u registru R1 procesora. Trećom naredbom se zbir iz registra R1 u procesoru prenosi u



memorijsku lokaciju imenovanu sa C koja je zadnjom linijom koda rezervisana za smeštaj podatka.

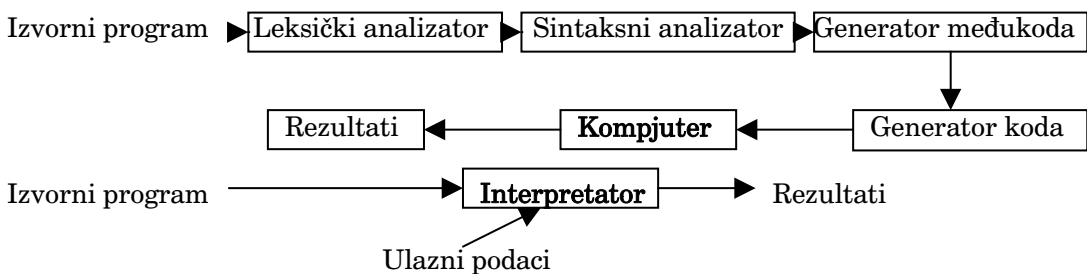
Mašinski kod iz primera 1.1 je ekvivalent ovom asemblerskom kodu i u njemu se umesto simboličkih kodova operacija i simboličkih adresa memorijskih lokacija i registra procesora koriste binarni kodovi i apsolutne adrese.

U okviru naprednijih asemblerskih jezika obično postoje mogućnosti za definisanje makro naredbi, složenijih naredbi koje predstavljaju sekvence više asemblerskih naredbi. Jednom definisane makro naredbe mogu se višestruko koristiti u programu u kome su definisane. Koncept funkcioniše tako što se za prevodenje ovako napisanog koda koristi poseban programski modul, makroprocesor koji najpre zapamti definicije makro naredbi, a zatim ih jednostavno ugradi u asemblerski kod tamo gde se one pozivaju, Slika 1.3.3. Rezultat je čisto asemblerski kod koji se pomocu asemblera prevodi u mašinski. Programski jezici sa ovim mogućnostima nazivaju se makroassemblerski jezici.



Slika 1.3.3. Makroprocesor

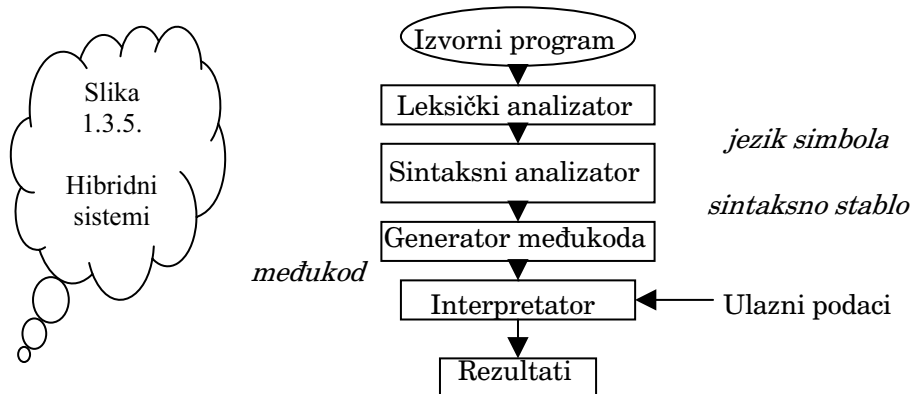
Sledeći korak u razvoju programskih jezika su viši programski jezici kod kojih se program definiše prema proceduri opisanoj algoritmom i u okviru kojih obično postoje koncepti za predstavljanje standardnih algoritamskih koraka. U literaturi se za ovu generaciju jezika koriste i nazivi algoritamski ili proceduralni jezici. Prvi viši programski jezici pojavili su se krajem pedesetih i početkom šezdesetih godina. Njihov razvoj traje do danas iako su osnovni koncepti u suštini ostali isti. Program se piše kao sekvenca naredbi sa unapred definisanom sintaksom, sastavljenih od reči engleskog jezika, a prevodi se u asemblerski, odnosno mašinski kod, pomocu programskih prevodilaca. Za prevodenje sa viših programskih jezika u mašinski kod mogu se koristiti kompilatori i interpretatori.



Slika 1.3.4. Kompilator i interpretator

Razlika između jednih i drugih ilustrovana je na slici 1.3.4. i uglavnom je u tome što kompilatori prevode kod programa u celini i formiraju izlaznu datoteku sa mašinskim kodom programa koji se zatim ubacuje u memoriju računara i izvršava, dok se kod interpretatora faza prevodenja prepliće sa fazom izvršavanja programa. Interpretatori prevode naredbu za naredbom i čim se formira celina koja se može izvršiti ona se i izvršava. Oba ova koncepta imaju svoje prednosti i primenjuju se kod određenih jezika. Komandni jezici i uopšte jezici namenjeni interaktivnoj komunikaciji sa računarom se obično realizuju kao interpretatorski, dok se za pisanje programa koji treba da se izvršavaju u realnom vremenu kada su efikasnost i pouzdanost primarni faktori, koriste se kompilatorski jezici.

Danas sve više postaju aktuelni i hibridni sistemi kod kojih se u prvoj fazi vrši prevodenje programa u celini do nivoa nekog međukoda, a zatim taj međukod interpretira. Ovaj koncept dobija sve više na značaju u mrežnim okruženjima gde je potrebno da se programi pisani na jednoj mašini mogu izvršavati na različitim mašinama u mreži, Slika 1.3.5.



Sedamdesetih godina kao metodologija programiranja prihvaćen je koncept strukturnog programiranja u okviru kojeg se algoritam programa gradi od standardnih algoritamskih struktura koje se ugrađuju jedna u drugu. Ovaj koncept je takode prihvaćen u okviru viših programskih jezika tako da se danas u klasifikacijama često koristi i termin strukturni jezici čime se izdvajaju viši programski jezici koji podržavaju strukturno programiranje.

Sledeći korak u razvoju programskih jezika su problemu orjentisani jezici. To su deklarativni jezici veoma visokog nivoa u okviru kojih se odstupa od eksplicitnog definisanja algoritma prema kome se izvršava program i sredstvima višeg nivoa definišu zahtevi koje program treba da realizuje. U ovu grupu jezika se obično svrstavaju jezici veštačke inteligencije u okviru kojih su zastupljeni različiti koncepti mada se u nekim klasifikacijama programski jezici veštačke inteligencije izdvajaju kao posebna grupa jezika.

Najznačajniji iz ove grupe jezika danas su objektno orjentisani jezici kod kojih je osnovni koncept objekat kojim se objedinjuju podaci i metodi kao jedini interfejs prema tim podacima. Programiranje pomocu objektnih jezika se svodi na definisanje objekata i događaja na koje ti objekti reaguju. Odgovor objekta na određeni događaj je izvršavanje unapred definisanog metoda.



*Primer 1.3. programa na programskom jeziku FORTRAN 77:*

```
PROGRAM SREDVR
INTEGER INTLIST (99)
INTEGER DUZ, SUM, SVR
REZ = 0
SUM = 0
READ *, DUZ
IF (( DUZ. GT. 0 ). AND. ( DUZ. LT. 100 )) THEN
DO 10 I = 1, DUZ
READ *, INTLIST ( BR)
SUM = SUM + INTLIST ( I)
10 CONTINUE
SVRED = SUM/DUZ
DO 20 I = 1, DUZ
IF ( INTLIST ( I) . GT. SVRED ) THEN
REZ = REZ + 1
END IF
20 CONTINUE
PRINT *, ' BROJ VECIH OD VSRED JE : ', REZ
ELSE
PRINT *, ' GRESKA - DUZINA NIZA NIJE DOBRA. '
END IF
STOP
END
```

Primer 1.3 je ilustracija programa napisanog na programskom jeziku FORTRAN 77. Programom se učitava niz celih brojeva, izračunava srednja vrednost niza i određuje i štampa ukupan broj elemenata niza čija je vrednost veća od srednje vrednosti niza.

Programski jezik COBOL (*Common Business Oriented Language*) pojavljuje se 1959. godine i od prvih dana razvija se kao jezik namenjen pisanju poslovnih aplikacija. Zahvaljujuci tome što je već tada osnovan i međunarodni COBOL komitet, u koji su ušle najveće kompanije iz oblasti računarstva (IBM, Honeywell, Flow-Matic i dr.) razvoj ovog programskog jezika je tekao kontrolisano i uvek u okvirima osnovnog koncepta i njegove namene. COBOL je u jednom periodu bio možda i najznačajniji programski jezik jer je oko 60% svih aplikacija koje su imale komercijalni značaj bilo napisano na ovom jeziku i smatralo se da u velikim računskim centrima zauzima pozicije koje ničim ne mogu biti poljuljane. Primer 1.4. je ilustracija programa napisanog na programskom jeziku COBOL. Program omogućava štampanje izveštaja o nabavnim i prodajnim cenama nekih artikala koji se prodaju u jednoj prodavnici.

*Primer 1.4. programa na programskom jeziku COBOL:*

```
IDENTIFICATION DIVISION.
PROGRAM - ID. LISTA.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE - COMPUTER. DPS6.
OBJECT - COMPUTER. DPS6.
INPUT - OUTPUT SECTION.
FILE - CONTROL.
SELECT STAMPA ASSIGN TO AO - MSD.
DATA DIVISION.
FILE SECTION.
FD STAMPA.
LABEL RECORD STANDARD DATA RECORD
```

```

STAM - SL.
    02 SNAZDZ PIC X (12)
    02 F PIC X (10)
    02 SDOB PIC X (20)
    02 F PIC X (6)
    02 SNABCEN PIC 9 (9)
    02 F PIC X (8)
    02 SPRODCEN PIC 9 (9)
    02 F PIC X (6)
WORKING - STORAGE SECTION.
01 WNAZDZ PIC X (12)
01 WDOB PIC X (20)
01 WNABCEN PIC 9 (9)
01 WPRODCEN PIC 9 (9)
01 CRTA PIC X (80) VALUE ALL "-".
01 ZAGLAVLJE.
    02 F PIC X (22) VALUE " NAZIV ARTIKLA ".
    02 F PIC X (24) VALUE " NAZIV DOBAVLJACA ".
    02 F PIC X (16) VALUE " NABAVNA CENA ".
    02 F PIC X (18) VALUE " PRODAJNA CENA ".
PROCEDURE DIVISION.
P1.
    OPEN OUTPUT STAMRA.
    MOVE ZAGLAVLJE TO STAM - SL.
    WRITE STAM - SL.
    MOVE CRTA TO STAM - SL.
    WRITE STAM - SL.
P2.
    MOVE SPACES TO STAM - SL.
    MOVE SPACES TO WNAZDZ WDOB.
    MOVE ZEROES TO WNABCEN WPRODCEN.
    DISPLAY " UNESITE NAZIV ARTIKLA, ZA IZLAZ UNESITE KRAJ".
    ACCEPT WNAZDZ.
    IF WNAZDZ = "KRAJ" GO TO KRAJ.
    DISPLAY "UNESITE NAZIV DOBAVLJACA".
    ACCEPT WDOB.
    DISPLAY "UNESITE NABAVNU CENU".
    ACCEPT WNABCEN.
    DISPLAY "UNESITE PRODAJNU CENU".
    ACCEPT WPRODCEN.
    MOVE WNAZDZ TO SNAZDZ.
    MOVE WDOB TO SDOB.
    MOVE WNABCEN TO SNABCEN.
    MOVE WPEODCEN TO SPRODCEN.
    WRITE STAM - SL.
    GO TO P2.
KRAJ
    CLOSE STAMPA.
    STOP RUN.

```

Možda najznačajniji iz prve grupe programskih jezika je Algol 60 (*Algorithmic Language*). To je jezik koji je definisan od strane grupe evropskih stručnjaka i prvi put predstavljen na Svetskoj izložbi u Parizu 1960. godine (broj 60 u nazivu ovog jezika odnosi se na godinu njegovog nastanka). U praktičnim primenama Algol nije nikada dostigao nivo koji odgovara njegovim karakteristikama. Razloga za to ima nekoliko. Jedan od bitnih je sigurno i to da Algol kao evropski jezik nikada nije imao jakog sponzora kakav je za FORTRAN bio IBM, DIGITAL ili COBOL komitet za COBOL. Međutim, Algol je jezik koji je imao najviše uticaja na razvoj teorije programskih jezika, a koncepte ugrađene u ovaj jezik prepoznavamo u najsavremenijim jezicima kakav je na primer Ada. Primer 1.5 je ilustracija programa napisanog na programskom jeziku Algo160.

Iznenaduje možda i činjenica da se veoma rano, već sa prvom grupom programskih jezika, 1961. godine pojavljuje i programski jezik BASIC (*Beginners Allpurpose Symbolic Instruction Code*), zamišljen kao jednostavnija varijanta programskog jezika FORTRAN i namenjen jednostavnom obučavanju kadrova. Intezivna primena i velika popularnost ovog programskog jezika su usledili znatno kasnije, osamdesetih godina, sa pojavom kućnih i personalnih računara. BASIC je i danas popularan programski jezik i njegove varijante VB (*Visual Basic*) i VBA (*Visual Basic for Application*) su danas savremeni jezici sa mogućnostima objektnog programiranja i rada u mrežnom okruženju.

*Primer 1.5. programa na programskom jeziku ALGOL 60:*

**begin**

```
integer i ; rel x, y, h, pi; array a,b [1: 10] ;
input ( a,b ) ;
pi := 3.1415926; H := pi/12;
for i := 1 step 1 until 10 do
  begin
    x := 0;
    m : if x <= pi/2 then y := (a [i]*x +b[i] * sin (x)/(x^2+1)
      else
    y := (b[i] *x +a[i] * cos (x)/(x^ 2-1)
    output (x,y);
    x := x + h; if x <= pi then go to M;
  end;
stop;
```

**end;**

Pojava viših programskih jezika je veoma mnogo uticala na sve širu primenu računara, proširila oblasti njihove primene i krug stručnjaka koji računar koriste u svom radu. Zbog svega toga u ovom periodu postala je aktuelna ideja o univerzalnom programskom jeziku, jeziku koji bi zadovoljio potrebe poslovne obrade, ali i omogućio rešavanje numeričkih problema, odnosno, koji bi bio kombinacija jezika FORTRAN, COBOL i Algol. Sa ovom idejom razvijen je PL/1 (*Programming Language One*). Sa istom idejom univerzalnog i najboljeg jezika i otprilike u isto vreme, razvija se i programski jezik Algol 68. Međutim, pokazuje se da ovakve ideje nisu u skladu sa nivoom razvoja hardvera računara i da tako kompleksni jezici toliko opterećuju sistem da nije moguća njihova praktična primena. Smatra se da su ovi jezici bili toliko kompleksni da kompilator za programski jezik Algol 68 nikada nije ni realizovan u punom obliku, a programski jezik PL/1, iako je ambiciozno bio zamišljen kao najbolji ( jedinstven ) programski jezik, nikada nije dostigao taj nivo popularnosti.

*Primer 1.6. programa na programskom jeziku PASCAL:*

**program recenica ( input, output );**

**(\* program za konvertovanje slova re~enice \*)**

**type slova = set of char;**

**var**

```
tekst : array [1..80] of char ;
i, n, pomak : 0..80 ;
mala, velika : slova ;
```

**begin**

```
mala := [ 'a'..'z'];
velika := [ 'A'..'Z'];
```

```

n := 0;
  repeat (* petlja kojom se ~ita re~enica *)
    n := n + 1;
    read ( tekst[n] );
  until (tekst[n] = '.') or (n = 80 );
  pomak := ord ('a') - ord ('A');
  for i := 1 to n do
    if tekst [i] in velika then
      write (chr(ord(tekst[i]) - pomak))
    else
      if tekst[i] in velika then
        write (chr(ord(tekst[i]) + pomak))
      else write ( tekst[i]);
  writeln;
end.

```

Kao odgovor na ove neuspešne ideje o projektovanju univerzalnog programskog jezika sa kompleksnim mogućnostima, početkom sedamdesetih godina pojavljuje se programski jezik Pascal. Autor Pascal-a je prof. N. Wirt koji ideji kompleksnog programskog jezika suprostavlja ideju strukturnog programiranja i jednostavnog programskog jezika koji se lako uči i lako primenjuje. Pascal je pre svega zamišljen kao jednostavan jezik namenjen obuci kadrova i učenju metodologije programiranja. Njegovi koreni su u jeziku Algo160.

Iako je bilo potrebno da prođe skoro cela jedna decenija da ideja strukturnog programiranja bude široko prihvaćena, pojava Pascal-a je jedna od najznačajnijih tačaka u razvoju programskih jezika. Sa Pascal-om u programske jezike ulazi još čitav niz novih koncepata kao što su jaki tipovi podataka i njihove eksplicitne definicije, prebrojivi i strukturni tipovi podataka. Radi se o konceptima koji su standardno prisutni u savremenim jezicima tako da se u klasifikaciji programskih jezika često koristi i termin Pascal-ski jezici ili jezici nalik Pascal-u (*Pascallike*). Takode se koristi i klasifikacija na predpaskalske i postpaskalske jezike. Pascal je takode jezik koji je doprineo širokoj popularizaciji PC računara jer je kroz mnoge verzije Turbo Pascal-a dugo bio najznačajniji programski jezik za razvoj aplikacija na PC računarima. Sa Turbo Pascalom prvi put je široko prihvaćen i koncept implementacije programskog jezika kao integrisanog interaktivnog okruženja koje objedinjuje editor, kompilator, linker i debager kao osnovne alate za razvoj aplikacija, što je takode jedan od danas široko prihvaćenih koncepata. Primer 1.6 ilustruje mogućnosti programskog jezika Pascal. Datim programom sa tastature računara učitava se rečenica koja se završava tačkom ili krajem reda. Mala slova u učitanj rečenici zamenjuju se velikim, a velika malim i tako konvertovan tekst štampa.

Ime profesora Wirta vezano je za još jedan značajan programski jezik i koncepte koje on unosi u teoriju programskih jezika. To je programski jezik Modula sa kojim prodire koncept programskih modula kao struktura koje objedinjavaju podatke i funkcije i procedure za obradu tih podataka. Sa modulima plasirana je i ideja o metodologiji modularnog programiranja gde se aplikacija generiše sintezom unapred definisanih modula u celinu koja zadovoljava funkcionalne zahteve.

Modula je programski jezik koji je imao svoj razvoj kroz varijante Modula 2 i Modula 3. To je danas moderan jezik sa mogućnostima konkurentnog programiranja.

Sedamdesete godine su period u kome se naglo šire oblasti primene računara i dramatično rastu potrebe za efikasnim i jeftinim softverom. Jedna analiza koja je za

potrebe američke vojske izvršena početkom sedamdesetih pokazala je da cena softvera nesrazmerno raste u odnosu na cenu hardvera. U tom trenutku predviđanja su bila da će ona u periodu između 1983. i 1999. dostići cifru od 24 biliona dolara. To je takođe period u kome se pojavljuje puno novih programskih jezika (u pomenutoj analizi pominje se broj od 450) u okviru kojih se eksperimentišu sa mnogim novim konceptima. Postaje očigledno da postoji potreba za savremenim programskim jezikom u okviru kojeg bi bili obuhvaćeni i usklađeni svi oni koncepti koji omogućavaju efikasno i pouzdano programiranje. Sa jednom takvom idejom, od strane Američkog ministarstva vojske (*United State Oepartment of Oefense* -DOD) raspisan je konkurs za projekat novog programskog jezika. Rezultat je programski jezik koji je dobio ime Ada po grofici Adi Augusti (*Augusta Ada Byron, the Countess of Lovelace*) ćerki lorda Bajrona (*Lord Byron*) koja je pisala prve programe namenjene mehaničkoj mašini Čarlsa Bebidža (*Charls Babbage*). Iako je Ada programski jezik projektovan sa idejom da preuzme primat i obeleži kraj dvadesetog veka i početak novog, danas je očigledno da je tok događaja bio nešto drugačiji od očekivanog. Ada nije doživela praktičnu primenu koja joj je bila predviđena, međutim ostaje jedan od najvažnijih jezika

*Primer 1.7. programa na programskom jeziku ADA:*

**generic**

```
type elem is private ;
type vector is array (integer range <> ) of elem;
with funkcija ">" (x,y: elem) return boolean;
```

**procedure sort ( v : in out vector );**

**procedure sort ( v : in out vector ) is**

```
  i: integer := v' first;
  j: integer := v' last;
  x: elem := v ((i+j)/2);
```

**procedure swap is new exchange (elem);**

**begin**

```
  while i <= j loop
    while x > v (i) loop
      i := i + 1;
    end loop;
    while v(j) > x loop
      j := j - 1;
    end loop;
    if i <= j then
      swap (v(i), v(j));
      i := i + 1; j := j - 1;
    end if;
  end loop ;
  if v' first < j then
    sort (v(v'first . . j));
  end if ;
  if i < v'last then
    sort (v(i . . v'last));
  end if ;
```

**end sort;**



Može se reći da je u Adi izvršena jedna revizija u razvoju programskih jezika veoma značajna za sva događanja u ovoj oblasti zadnjih godina. Ada ostaje model jezika po kome se danas postavljaju svi značajniji koncepti programskih jezika, posebno u domenu tipova podataka, modularizacije, apstrakcije podataka, konkurentnog programiranja, obrade izuzetaka i mnogi drugi.

Razvoj programskog jezika C imao je sasvim drugačiju hronologiju i može se reći sasvim suprotan tok od razvoja programskog jezika Ada. Još jednom je potvrđeno da su predviđanja u ovoj oblasti nezahvalna i da često u praksi zažive mnogi koncepti koji teorijski nemaju opravdanja.

Programski jezik C je projektovan 1970. godine kao sastavni deo projekta operativnog sistema UNIX, kao jezik koji je korišćen u razvoju samog operativnog sistema. Međutim značaj jezika C je rastao paralelno sa sve širom primenom UNIX operativnog sistema. Pojavom knjige *The C programming Language*, 1978. godine, autora C-a D. M. Ritchi i B. W. Kernighan, C je definisan kao jezik opšte namene. Tek pet godina kasnije pojavljuje se ANSI standard ovog jezika u kome su mnogi koncepti i mnoge nedorečenosti C-a morale da budu revidirane. Prvobitno namenjen razvoju operativnog sistema i drugih komponenti sistemskog softvera C je viši programski jezik sa nekim specifičnostima koje ga približavaju asemblerskim jezicima. To su njegove prednosti ali i njegovi nedostaci. C je sigurno jezik koji ne može da izdrži poređenje sa jezicima kakvi su Pascal, Modula i Ada kada se ocenjuju neki elementi koji se danas smatraju standardnim u okviru jezika. Njegov najveći nedostatak je to što dozvoljava implicitne intervencije niskog nivoa čime slabi pouzdanost sistema. Međutim to je i njegova najveća prednost. Posebno je popularan baš kod stručnjaka iz računarske tehnike i pogodan za razvoj komponenti sistemskog softvera u okviru kojih su česte potrebe za spregom niskog nivoa sa resursima sistema za koji se softver i projektuje. Primer 1.8 je ilustracija programa napisanog na programskom jeziku C. To je jednostavan program kojim se razlomak konvertuje u decimalan broj.

*Primer 1.8. na programskom jeziku C.*

```
/* PRIMERC.C - Primer programa u C -u */
```

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    char inbaf[100];
```

```
    float gornji, donji ; /*elementi razlomka */
```

```
    float vred /*vrednost broja*/
```

```
    printf ( " Konvertovanje razlomka u decimalni " );
```

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

```
    printf ( "Broilac : ");
```

```
    gets (inbaf);
```

```
    sscanf (inbaf, "%f", & gornji);
```

```
    printf ("Imenilac : ");
```

```
    gets (inbaf);
```

```
    sscanf (inbaf, "%f". &donji);  
    vred = gornji /donji; /* konvertovanje razlomka u decimalni broj */  
    printf ("\n %f / %f = %f", gornji, donji, vred);  
    return 0;  
}
```

Sa sve širim prihvatanjem koncepta objektno-orijentisanog projektovanja softvera i koncepta objektnog programiranja programski jezik C, odnosno njegova objektna nadgradnja C++, postaje posebno popularan. Posebno su značajni i široko korišćene varijante Borland C++ i Microsoft Visual C++.

Nesporno je da zadnju deceniju 20. veka u razvoju programskih jezika obeležava nagli prodor objektnih jezika. Objektni koncept se prirodno nadovezuje na koncept apstrakcije podataka koji postoji kod modularnog programiranja i tome dodaje nove kvalitete kao što su nasleđivanje i polimorfizam. Prvim objektnim jezikom danas se smatra programski jezik Smalltalk, odnosno njegova varijanta Smalltalk-80, mada se koreni objektnih konceptata javljaju još u jeziku Simula 67. Smalltalk je primer potpuno objektno-orijentisanog jezika kod kojeg se objektni koncept proteže kroz sve elemente jezika. Nastao je kao softverska polovina ambiciozno zamišljenog projekta *DYNABOOK* idealističkog vizionara Alana Kaya. DYNABOOK je zamišljen kao prvi pravi personalni računar sa jednakim mogućnostima za upravljanjem svim vrstama informacija, pri čemu je jezik Smalltalk trebalo da iskaže izražajnost i upotrebljivost. Fundamentalne ideje objekata, poruka i klasa preuzete su iz jezika Simula. Jezik Simula omogućava kreiranje objektno orijentisanih sistema ali koristi standardno procedurno orijentisani jezik Algol za upravljačke strukture kao i osnovne strukture podataka.

## 1.5. Karakteristike programskih jezika

Skup pravila kojima se definišu pravilne konstrukcije u nekom jeziku nazivaju se **gramatika** toga jezika.

Svaku nauku o jeziku karakteriše:

- **sintaksa** (od grčkog syntexis - sastavljanje), nauka o sklopu jezičkih konstrukcija
- **struktura podataka**
- **semantika** (od grčkog sematikas - značajni), nauka o značenju (smislu) tih konstrukcija, i
- **vezano vreme** (bindig time).

Ove karakteristike programskih jezika određuju širinu primene i korisnost jezika.

**1.5.1. Sintaksa jezika.** Sintaksa programskog jezika obuhvata strukturu jezika i pravila za korišćenje jezika. Ustvari, to je gramatika jezika strogo definisana nad određenim skupom simbola, kako bi prevodioc (kompajler) bio u stanju da prevede

program sa izvorog jezika na izvršni jezik. Skup dozvoljenih simbola formira azbuku jezika koja se malo razlikuje kod raznih programskih jezika.

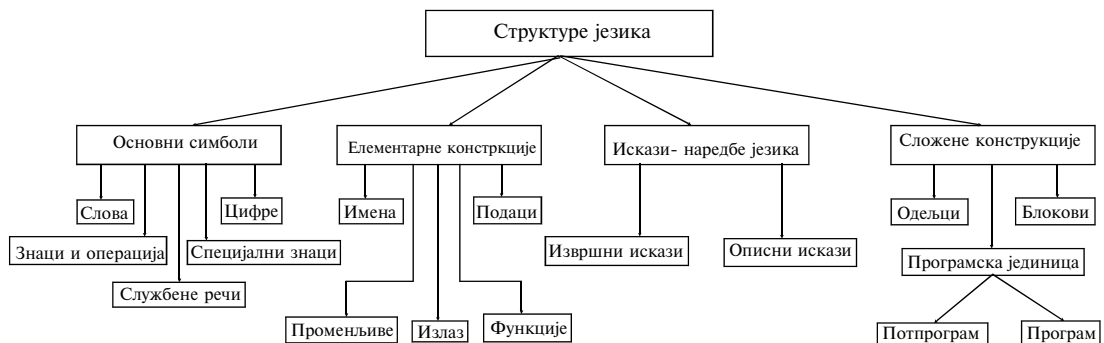
Sintaksa jezika obezbeđuje pravila za kombinovanje simbola azbuke u imena koja čine operantski i programski deo strukture podataka. Sintaksa većine programskih jezika zahteva da imena podataka ne prelaze određenu dužinu, da počinju alfabetskim simbolima (slovima), da mogu sadržati brojeve, ali ne i specijalne znake.

Sintaksa jezika takođe detaljno specificira pravila šta i kako računar treba da radi. Određene kombinacije tih pravila rezultuju u naredbe kojima je definisana vrsta operacije i adrese podataka nad kojim dotična operacija treba da se primeni. Kod nižeg nivoa jezika oblik naredbe vrlo je blizak mašinskom jeziku i načinu na koji računar funkcioniše. Kod jezika viših nivoa, kao što su FORTRAN, COBOL, PL/1, sintaksa jezika omogućava prevodiocu da jednu izvornu naredbu razbije na niz naredbi u izvršnom jeziku. Time se omogućava neposredno programiranje cikličkih algoritamskih struktura, korišćenje podograma i drugih programa.

Izračunavanje jednog višeg programskog jezika svodi se na izračunavanje njegove gramatike, pri čemu se za opis konstrukcije jezika koriste tzv. meta jezici od kojih je najpoznatiji Bakusov jezik metalingvističkih formula.

U jednom višem programskom jeziku možemo strukturno izdvojiti sledeće konstrukcije različite složenosti (sl. 1.5.1.)

- osnovni simboli
- elementarne konstrukcije
- iskazi
- složene konstrukcije.



Sl. 1. 5.1.

*Osnovni simboli* ili azbuka jezika čine skup nedeljivih znakova - simbola od kojih se obrazuju ostale jedinice. Najčešće osnovne simbole čini skup tipografskih znakova: slova (obično velika slova engleske azbuke), cifre dekadnog brojnog sistema, interpunkcijski i specijalni znaci. U osnovne simbole ubrajaju se i skup pomoćnih (službenih) reči, kao i grupa znakova koja se uvek javlja u nepromenljivom obliku. Po sematici osnovni simbol uvek označava samog sebe.

*Elementarne konstrukcije* su najmanje smislaone konstrukcije jezika koje ne mogu samostalno da postoje u okviru jednog jezičkog štiva - programa za slučaj programskih jezika. To su na primer razni nazivi - imena promenljivih, funkcija i nizova, dozvoljeni oblici podataka kako numeričkih tako i nenumeričkih, odgovarajući formati za ulazno-izlazne podatke itd. Ovde spadaju i složenije nesamostalne konstrukcije kao što su razne vrste izraza (aritmetički, logički), spiskovi, liste itd.

*Iskazi* rečenice jednog jezika su takve konstrukcije koje mogu samostalno da postoje, jer po značenju predstavljaju jednu potpunu završenu misao. Za slučaj viših programskih jezika iskazi su osnovne jedinice pomoću kojih se obrazuje program. Iskaz može da ukazuje na neko dejstvo ili da sadrži informaciju potrebnu pri prevođenju ili izvršavanju programa. Upravo po ovim osobinama svi iskazi dele se na:

*izvršne iskaze*- naredbe koje ukazuju na neko dejstvo i  
*opisne iskaze*- obaveštenja.

*Složene konstrukcije* predstavljaju nadgradnju jednog jezika koji čine nezavisnu celinu, a obrazovane su od posebnih iskaza. Kakva će biti ova celina zavisi od konkretnog programskog jezika. Najčešće su to posebne programske jedinice, glavni program i potdprogram kao u slučaju FORTRAN jezika, ili odeljci, na primer kod COBOL-a, ili blokovi kod ALGOL-a i PL/1.

Da bi postojala potpuna kontrola nad izvršenjem naredbi, sintaksa jezika obezbeđuje i sredstva za testiranje i brojanje operacija. To je naročito pogodno kod iterativnih petlji.

Poznavanje sintakse jezika od strane programera je neophodno. Radi toga se izučavanje sintakse bilo kog jezika mora obaviti na formalan i precizan način, kako se to obično i izlaže u literaturi.

**1.5.2. Struktura podataka.** Programski jezik je projektovan da radi sa strukturom podataka koja može imati i vertikalnu i horizontalnu dimenziju. Neki od jezika više raspoznaju razne tipove struktura podataka od drugih jezika, čime se određuje klasa i opseg poslova za koje je određeni jezik pogodniji od drugog jezika. Važan aspekt strukture podataka je u povezivanju sintakse sa sematikom programskog jezika.

**1.5.3. Semantika jezika.** Semantika programskog jezika bavi se transformacijom strukture podataka, naročito između ulazno-izlaznih operanada. Pod sematikom jezika može se shvatiti promena u operantnoj strukturi podataka u toku vremena, izvršenjem niza operacija definisanih naredbama programa. Sematika jezika obuhvata različiti repertoar operacija, koje mogu biti na primitivnom i makro nivou. *Primitivne* operacije su elementarne operacije koje se ne mogu dalje deliti. *Makro* operacije sastoje se od niza primitivnih operacija. One su sastavni deo jezika i mogu automatski obavljati niz operacija koje se često sreću u praksi. Prema tome, makro operacija može se shvatiti kao paket primitivnih operacija, koji se poziva na izvršenje, slično podprogramu. Makro podprogram je efikasniji od podprograma, jer uzima u obzir specifičnost hardvera računara za koji je makro podprogram i pisan.

Jedna ista operacija može biti makro za jedan nivo jezika, a primitivna za drugi nivo jezika. Npr. čitanje podataka sa magnetne trake je makro operacija za niži nivo jezika i poziva se sa GET makro. Ista operacija je primitivna u PL/1 jeziku, gde GET ima smisao primitivne operacije.

**1.5.4. Vezano (predodređeno) vreme-** Binding time. Vezano vreme je trenutak kada vreme izvršenja neke operacije od strane računara postoje fiksirano (vezano) i ne može više da varira. Jedna od karakteristika programskih jezika je i u načinu manipulisanja vezanim vremenom. Vezano vreme može se desiti u bilo kom trenutku, počev od pisanja programa na izvornom jeziku od strane programera pa do izvršenja programa u izvršnom jeziku. Vezano vreme može se dogoditi za vreme pisanja programa na izvornom jeziku, kod prevođenja izvornog u izvršni jezik, kod punjenja izvršnog

jezika u cilju njegovog izvršenja, ili u toku stvarnog izvršenja programa. Vezana vremena koja se dešavaju ranije, npr. za vreme pisanja ili prevođenja izvornog jezika nazivaju se rana vezana vremenom, za razliku od posponiranih (zakašnjenih) vezanih vremena. Programer može imati znatnog uticanja na trenutak pojave vezanog vremena. Taj trenutak je od bitnog značaja za fleksibilnost korišćenja programskih jezika. Što je više zakašnjeno vezano vreme, veća je i fleksibilnost. To je naročito važno kod onih poslova kod kojih programer ne poseduje dovoljno informacija da bi unapred mogao postaviti fiksni algoritam koji bi mogao rešiti sve slučajeve na tačno isti način. Možda on mora obezbediti neke alternative u programu koje mogu izaći na kraj sa nekim slučajevima koje će on u potpunosti poznavati tek u trenutku punjenja programa za izvršenje. Da bi to bilo moguće ostvariti, zakašnjeno vezano vreme je neophodno.

Važan uticaj na vezano vreme imaju programi za prevođenje koji su deo programskog sistema. To su: asembleri, kompajleri, inkrementalni kompajleri i interpreteri.

*Asembleri* obezbeđuju najranije vezano vreme, pri čemu obično vrše prevođenje instrukcija sa izvornog jezika na izvršni u odnosu jedan prema jedan (jedna linija instrukcija na izvršnom jeziku generira se od jedne linije instrukcija sa izvornog jezika). Makro naredbe su retko prisutne u ovom slučaju. Može se reći da je u slučaju asemblera i rano vezanog vremena sve predodređeno u vreme pisanja programa od strane programera.

*Kompajler* (prevodilac) poseduje sve osobine asemblera, s tim što može prihvatiti i makro instrukcije. Vezano vreme za programe koji se prevode obično pada u vreme prevođenja programa (vreme kompilacije).

Često operativni sistem može zakasniti vezano vreme do početka izvršenja programa na izvršnom jeziku. Međutim, ovo može da ima i negativnih posledica da program, koji je radio na jednom sistemu, ne radi na drugom, radi toga što se njihovi operativni sistemi razlikuju.

*Inkrementalni kompajler*, koji se najčešće koristi kod konverzionog i tajmšering rada računara, prevodi po jednu naredbu sa izvornog jezika i to u redosledu kako ih programer i piše. Ovaj način se bitno razlikuje od asemblera ili kompajler programa, koji vrše prevođenje jednog programa odjednom u celosti.

Inkrementalni kompajler razmatra posebno svaku liniju instrukcija na izvornom jeziku. Ukoliko se dotična instrukcija odnosi na neku programsku ili operantsku strukturu podataka koja nije ranije definisana u programu, prevodilac ostavlja vezano vreme otvoreno dok programer ne obezbedi potrebne informacije i time omogući pojavu vezanog vremena. Pri tome mu operativni sistem ukazuje na tip i redosled potrebnih informacija. Programer može izvršiti izmenu bilo koje instrukcije u svom izvršnom programu, u bilo koje vreme, čak i prekidanjem programa u izvršnom jeziku.

*Interpreter* može zakasniti vezano vreme sve do trenutka kada počinje izvršenje programa. On prevodi po jednu instrukciju, nakon čega se ista izvršava. Radi toga, vreme izvršenja programa uz korišćenje interpretera je najduže, za razliku od korišćenja asemblera, kada je to vreme najkraće.

## 1.6. Mašinski jezici

Mašinski jezici su u upotrebi od pojave prvog računara. Svaki model računara ima svoj mašinski jezik. Prema tome za njih se može reći da su jezici samih računara. Oni su bliski računaru, ali su složeni programeru, radi toga što su apstraktni, detaljni i zahtevaju da se svaka adresa u računaru specificira u apsolutnoj formi.

Računar uvek neposredno izvršava program na mašinskom jeziku, koji predstavlja skup naredbi (instrukcija) računara. Preciznije mašinski jezik je način prikazivanja informacija (brojnih podataka, teksta i naredbi), unutar računara. Naredba računara je tačno specificirano dejstvo koje računar obavlja za određeni vremenski interval. Ako želimo da se program neposredno izvršava onda moramo da ga izrazimo na mašinskom jeziku konkretnog računara. O specifičnosti programiranja na mašinskom jeziku biće više reči u posebnom prilogu.

Pisanje programa na mašinskom jeziku ima sledeće osobine:

- potrebno je izvršiti detaljizaciju algoritma, tako da elementarni koraci odgovaraju pojedinim naredbama, računara,
- uvek je potrebno poznavati skup naredbi datog računara- konkretni mašinski jezik. Program napisan za jedan računar obično se može obavljati na drugom,
- sastavljanje i pisanje programa- kodiranje kao i testiranje programa i podataka u memoriji računara zahteva detaljno poznavanje računara tako da to obavlja odgovarajući stručni kadar.

Sve ovo ukazuje da je programiranje na mašinskom jeziku najteži način programiranja uz mogućnost daleko veće pojave grešaka.

Danas se programiranje na mašinskom jeziku vrši uglavnom u sledećim okolnostima:

- na računarima koji nemaju drugu mogućnost programiranja,
- za dobijanje osnovnih sistemskih programa za računare u razvoju.

### 1.7. Simbolički jezici

Ovi jezici su poznati kao mašinsko- orjentisani jezici ili jezici niskog nivoa. Naziv *simbolički jezici* dobili su zato što se mašinska naredba predstavlja u simboličkom obliku. Pri tome su omogućeni različiti nivoi ovih jezika prema tome da li je cela naredba simbolički prikazana u vidu *mnemo* koda (operativni i adresni deo) ili je to slučaj samo sa operativnim delom. Neki simbolički jezici, pored postojećih naredbi računara, sadrže izvedene naredbe čija dejstva mogu znatno da prevaziđu dejstva osnovnih naredbi računara. Ove izvedene naredbe odgovaraju skupu naredbi na mašinskom jeziku i nazivaju se makro naredbe. Program na simboličkom jeziku ne izvršava se neposredno u računaru već se prevodi putem specijalnog programa, koji se zove autokod ili assembler, u mašinski radni program. Pri tome je prevodenje najčešće 1:1, tj. jednoj simboličkoj naredbi odgovara jedna mašinska naredba. Izuzetak su makro naredbe, kod kojih se jedna makro naredba zamenjuje ili grupom mašinskih naredbi ili pozivom na određeni podprogram za slučaj složenih makro naredbi. Često se simbolički jezik naziva imenom programa- prevodioca. Sem izvršnih naredbi (simbolički naredbi i makro naredbi) postoje i tzv. opisne naredbe ili pseudo naredbe koje se ne prevode u mašinske naredbe, već sadrže dodatne informacije potrebne programu prevodioca. Tipične pseudonaredbe su početak i kraj programa, definisanje konstanti- podataka, specificiranje memorijskog prostora, itd.

Programiranje na simboličkom jeziku daleko je lakše i pouzdanije nego na

mašinskom. Pisanje programa je jednostavnije jer nije potrebno voditi računa o konkretnim fizičkim mestima- adresama kako naredbi tako i podataka. Osim toga umanjuju se teškoće oko testiranja i ispitivanja programa. Umetanje novih i izbacivanje starih naredbi se automatski obavlja ponovnim prevodenjem ispravljanog programa.

I pored ovih poboljšanja, programiranje na simboličkom jeziku ima niz nedostataka. Navedimo najvažnije:

- potrebno je vršiti detaljizaciju algoritma tako da elementarnim algoritamskim koracima odgovaraju dejstva simboličkih naredbi,
- raznovrsnost računara dovela je do raznovrsnosti simboličkih jezika, tako da svaki konkretni računar ili familija računara ima svoj poseban simbolički jezik. Program napisan na simboličkom jeziku za jedan računar ne može bez veće ili manje prerade da se izvršava na drugom.

Zbog ovoga je programiranje na simboličkim jezicima sve više potisnuto programiranjem na višim programskim jezicima i primenjuje se u slučajevima:

- Kada nema druge mogućnosti.
- Kada je postojeći viši programski jezik nepogodan za programiranje datog zadatka.

Ako su postavljeni ograničavajući faktori u pogledu brzine izvršavanja i raspoloživog memorijskog prostora:

- u slučaju programa za procesne računare koji rade u realnom vremenu ograničavajući faktor je vreme,
- kod složenijih programa koji se često eksploatišu u računskom sistemu ograničavajući faktori su vreme izvršavanja i zauzeće prostora operativne memorije. Programiranje na simboličkom jeziku u ovim slučajevima može da da optimalnije rešenje.
- Za pisanje sistemskih programa i biblioteke programa datog računskog sistema.

## 1.8. Viši programski jezici

Univerzalnost računara omogućila je njegov prodor u sve oblasti čovekove delatnosti tako da se broj korisnika računara znatno uvećao, a istovremeno promenio i njihov sastav. Dok su to ranije bili specijalisti programeri, sada su to stručnjaci iz raznih oblasti.

U cilju rešavanja nastalog problema tj. pojave korisnika koji ne poznaju tehniku mašinskog (simboličkog) programiranja stvoreni su viši programski jezici ili kako se danas još nazivaju problemsko-orjentisani jezici.

Osnovna ideja je da se program za računar pripremi na jednom jeziku koji je bliži čovekovom načinu izražavanja i na kome se za određenu klasu problema program može kraće ili lakše napisati. Tako napisan program ne izvršava se neposredno u računaru, već se slično programu napisanom na simboličkom jeziku, automatski prevodi putem specijalnog programa u program na mašinskom jeziku konkretnog računara. Za

razliku od simboličkog jezika, gde je prevođenje bilo 1:1, dejstva koja se zadaju u višem programskom jeziku su daleko složenija i raznovrsnija tako da formulu za prevođenje možemo da izrazimo kao 1:N, gde 1 označava dejstvo na višem programskom jeziku, a N broj naredbi računara koje po značenju (sematici) odgovaraju ovom dejstvu. Program na višem programskom jeziku zove se izvorni ili ulazni program, prevedeni mašinski program zove se radni ili izvršni program, a program sistema kojim se prevodi izvorni program naziva se prevodilac ili kompajler.

Pojava viših programskih jezika doprinela je povećanju broja korisnika i još više proširila polja primene računara.

Brzo prihvatanje i masovno korišćenje ovih jezika može se objasniti sledećim činjenicama:

- programiranje je znatno olakšano. Ovi jezici su bliski čovekovom pisanom jeziku i operativnoj terminologiji iz odgovarajuće oblasti, skraćeno je vreme obuke u programiranju i izbegnute su teškoće oko detalja u vezi sa programiranjem na mašinskom, odnosno simboličkom jeziku,
- ovi jezici su nezavisni od strukture samog računara. Program napisan na ovom jeziku može da se izvršava na svakom računaru koji ima prevodilac (kompajler) za ovaj jezik,
- postoji mogućnost izmene programa i iskustva između korisnika jednog problemsko- orjentisanog jezika.

Sve ovo je doprinelo razvitku ovih jezika dako da danas imamo nekoliko hiljada jezika za par stotina različitih delatnosti. Postoje različiti nazivi i klasifikacije viših programskih jezika.

Tako su nazivi: viši programski jezik, algoritamski jezik, problematsko-orjentisani jezik ili prosto programski jezik sinonimi za programske jezike visokog nivoa. Postoji podela na proceduralne i neproceduralne jezike prema tome da li su operativna dejstva i njihov redosled unapred dati ili ne. Postoji klasifikacija prema pristupu problemu: jezici za definisanje problema, opis problema ili njegovo rešenje. Možda je najprirodnija klasifikacija prema tipičnim oblastima primene što i opravdava naziv problemsko- orjentisani jezik. Tako možemo izdvojiti sledeće osnovne grupe jezika:

- jezici za opis naučno- tehničkih problema, kao što su: ALGOL, FORTRAN, NELIAC i dr.
- jezici za opis problema iz poslovne obrade podataka na primer COBOL, TABSOL
- jezici za opisivanje informaciono- logičkih zadataka: COMIT, IPL, LIPS i dr.
- jezici za opisivanje algoritma za upravljanje tehnološkim procesima u realnom vremenu, na primer ART, i simulaciju sistema, na primer GPSS (General Purpose System Simulatio)
- u poslednje vreme javljaju se jezici koji obuhvataju nekoliko oblasti primene, na primer PL/1 ili NPL (Nenj Program Language), GPL (General Purpose Language) i dr.

I pored velikih prednosti ovih jezika možemo navesti i nekoliko nedostataka:



- prisustvo procesa translacije, tj. vremena i angažovanja računara potrebnih za prevođenje izvornog programa na izvršni. Sa tačke gledišta korisnika računara ovo predstavlja gubitak vremena i novca,
- nejednoznačnost i neusklađenost optimalnog izvornog programa i odgovarajućeg izvršnog programa na mašinskom jeziku. Optimalan program na izvornom jeziku ne mora da bude optimalan i na mašinskom jeziku,
- otežano je komuniciranje sa računarom u toku izvršavanja programa, jer se u računaru izvršava radni program na mašinskom jeziku, koji korisnik najčešće ne poznaje.

### **1.9. Podela programskih jezika prema oblasti primene**

Danas računari nalaze primenu u različitim oblastima ljudske delatnosti. Određeni programski jezici se mogu koristiti kao univerzalni jezici za razvoj aplikacija u različitim oblastima primene dok su drugi prvenstveno namenjeni primenama u određenim oblastima. Zbog toga se često vrši podela programskih jezika prema oblasti primene. Tako možemo govoriti o programskim jezicima za:

- naučne aplikacije,
- poslovnu obradu,
- veštačku inteligenciju,
- projektovanje sistemskog softvera,
- projektovanje pomocu računara (CAD),
- upravljahje pomocu računara (CAM),
- opis hardvera računara,
- simboličko programiranje,
- linearno programiranje,
- simulaciju,
- računarske komunikacije,
- prostorno planiranje,
- stono izdavaštvo,
- obradu teksta ( tekst procesori )

Osvrnućemo se na neke od najvažnijih ovde nabrojanih grupa programskih jezika.

#### **1.9.1. Jezici za naučne aplikacije:**

Rešavanje različitih numeričkih problema i brzo izvršavanje obimnih izračunavanja u naučno-tehničkim aplikacijama bila je i ostala jedna od najvažnijih oblasti primene računarskih sistema. Radi se o programima koji obično ne zahtevaju rad sa složenim strukturama podataka ali su to često iterativna izračunavanja sa velikim

brojem ponavljanja raznovrsnih operacija. Efikasnost programa je stoga osnovni zahtev kod ovog tipa aplikacija. U početnim fazama razvoja računarskih sistema za razvoj naučnih aplikacija korišćeni su isključivo asemblerski jezici, ali su već neki od prvih viših programskih jezika (FORTRAN, Algol) razvijeni sa idejom da se primenjuju u te svrhe. Iako je danas broj novih programskih jezika jako veliki u domenu naučno-tehničkih aplikacija, kada je efikasnost programa primarni zahtev, FORTRAN je ostao jedan od najznačajnijih i najviše primenjivanih jezika.

### 1.9.2. Jezici za poslovne aplikacije

Korišćenje računara za obradu podataka u okviru poslovnih aplikacija počinje već pedesetih godina u ranim fazama njihovog razvoja. Po broju aplikacija to je sigurno jedna od najznačajnijih oblasti primene. Za razliku od naučnih aplikacija ovde je akcent na strukturama podataka. Dugo vremena u ovoj oblasti najznačajniji programski jezik bio je COBOL. U novije vreme sa razvojem novih koncepata u okviru poslovnih aplikacija koji se oslanjaju na korišćenje baza podataka COBOL gubi svoj primat i ustupa mesto specijalizovanim sistemima za rad sa tabelama (*Spread Sheet Systems*), sistemima za upravljanje bazama podataka (*Database Management Systems*), kao i novim objektno orijentisanim jezicima koji nalaze primenu u upravljanju objektno orijentisanim bazama podataka.

### 1.9.3. Jezici veštačke inteligencije

Razvoj računarskih sistema sa elementima inteligencije po uzoru na ljudsku inteligenciju je ideja koja se permanentno razvija već od same pojave prvih računara. Jezici koji nalaze primenu u ovoj oblasti omogućavaju predstavljanje znanja i mehanizama za zaključivanje kao osnovnih koncepata na kojima se zasniva veštačka inteligencija. Prvi programski jezik razvijen za primenu u oblasti veštačke inteligencije bio je funkcionalni jezik Lisp u okviru kojeg koriste liste kao osnovne strukture podataka. U ovoj oblasti tokom niza godina eksperimentisano je sa različitim konceptima tako da je i raznovrsnost jezika iz ove oblasti veoma velika. Uz Lisp značajni jezici su Prolog i Plazma za logičko programiranje, Smalltalk i drugi.

### 1.9.4. Jezici za razvoj sistemskog softvera

Operativni sistemi, programski prevodioci i svi softverski alati koji su podrška radu računarskog sistema kao što su editori, linker, debageri i slično, svrstavaju se u kategoriju sistemskog softvera. Osnovni zahtevi kod projektovanja komponenti sistemskog softvera su pouzdanost i efikasnost. Takode, kod ovih programa se često zahteva jača, neposrednija veza sa hardverom sistema tako da su u ovoj oblasti dugo korišćeni asemblerski jezici. Mnogi veliki proizvođači računarskih sistema razvili su svoje specijalizovane mašini-orjentisane više programske jezike za razvoj sistemskog softvera. Takvi su na primer IBM-ov jezik PL/S i Digital-ov jezik BLISS. Najznačajniji jezik u ovoj oblasti danas je C, nastao kao jezik za razvoj operativnog sistema UNIX. To je efikasan jezik sa mogućnostima niske veze sa hardverom računara u kome je programer oslobođen mnogih restrikacija koje postoje kod drugih jezika opšte namene

radi povećanja njihove pouzdanosti. Međutim, to je jezik namenjen pre svega stručnjacima koji su svesni svih opasnosti koje postoje kada se programira na tom nivou i nepouzdan je za razvoj aplikacija opšte namene.

### 1.9.5. Jezici za računarske komunikacije

Povezivanje računara i problemi komunikacije među njima su takođe jedan od koncepata prisutnih u funkcionisanju računarskih sistema već dugi niz godina. Međutim, ovi problemi postaju posebno aktuelni sa intezivnim razvojem Interneta kao globalne računarske mreže tako da se može govoriti o posebnim programskim jezicima namenjenim računarskim komunikacijama i radu u mrežnim Internet i Intranet okruženjima. To su pre svega jezici za upravljanje računarskim mrežama i razvoj Web (*World Wide Web*) aplikacija u Internet okruženjima. Problemi upravljanja računarskim mrežama su samo segment u okviru sistemskog softvera i u ovoj oblasti najznačajniji programski jezik je svakako C. Međutim, u ovu oblast sve više prodiru Web tehnologije koje se oslanjaju na nove jezike projektovane u cilju primene u mrežnom okruženju. Takvi su jezici HTML, VRML, JavaScript i Java, jezik koji preuzima primat u oblasti mrežnih aplikacija.

### 1.9.6. Jezici specijalne namene

U kategoriju jezika specijalne namene svrstaćemo sve jezike projektovane u cilju primene u određenoj oblasti ili za rešavanje nekih problema iz uskog područja u određenoj oblasti. Razvijaju se od samog početka razvoja računarskih sistema i njihov broj je ogroman. Pomenućemo samo dva od prvih jezika iz ove grupe: RPG -jezik za pripremu izveštaja i GPSS jezik za simulaciju sistema. Radi se obično o jezicima sa puno specifičnosti koji su rezultat ili podrška istraživanju u određenoj oblasti koji se teško mogu klasifikovati i porediti.

## 1.10. Kriterijumi ocene jezika

Da li je jedan programski jezik dobar ili loš i da li je jedan jezik bolji ili gori od drugog su pitanja koja zahtevaju dublju analizu i pre svega postavljanje kriterijuma po kojima se jedna takva ocena donosi. Kako je cilj ovog udžbenika upoznavanje sa ključnim mehanizmima u savremenim programskim jezicima, njihova evaluacija i poređenje sličnih mehanizama u različitim jezicima neophodno je da izvršimo klasifikaciju kriterijuma po kojima će se ta ocena vršiti.

### 1.10.1. Čitljivost

Čitljivost, razumljivost programa pisanih na određenom programskom jeziku je jedan od važnih kriterijuma pri oceni određenog programskog jezika. Potreba za čitljivošću programa je očigledna. Boljom čitljivošću smanjuje se potreba za opsežnom dokumentacijom programa, program sam sebe dokumentuje. Takoće, ispravke i izmene programa su mnogo jednostavnije kada je njegova čitljivost veća. Na čitljivost utiče čitav niz faktora, od skupa ključnih reči programa do stepena modularizacije koji može da se postigne. Navešćemo neke od osnovnih koncepata koji utiču na čitljivost jezika:

➤ Jednostavnost jezika

Jezik koji ima veliki broj osnovnih koncepata je obično mnogo teži za prihvatanje od jednostavnog jezika sa malim brojem različitih koncepata.

Programeri koji treba da koriste kompleksan jezik obično imaju naviku da uče podskup jezika i ignorišu složenije koncepte. Čitljivost jezika može da bude narušena i ako se u jeziku za jedan koncept mogu koristiti različiti načini definisanja. Npr. u jeziku C inkrementiranje celobrojne promenljive može da se predstavi na četiri različita načina:

```
br = br + 1; br += 1; ++br; br++;
```

Svaka od ovih naredbi ima drugačiju semantiku, ali je efekat isti u odnosu na vrednost promenljive `br`. Nekada i veliko pojednostavljenje koncepata može negativno da utiče na čitljivost programa. Kao primer mogu se uzeti asemblerski jezici kod kojih se program piše preko sasvim pojednostavljenih naredbi, ali je čitljivost programa loša.

➤ Ortogonalnost

Ortogonalnost programskog jezika sastoji se u tome da se jezik zasniva na relativno malom skupu koncepata koji se mogu kombinovati na mali broj različitih načina.

➤ Upravljačke strukture

Broj i raznovrsnost upravljačkih struktura takode bitno utiču na čitljivost programa pisanih u određenom programskom jeziku. Pojava strukturnih jezika sedamdesetih godina bila je odgovor na lošu čitljivost i mali broj upravljačkih naredbi jezika iz prethodnog perioda.

➤ Strukture podataka

Postojanje mehanizama za adekvatno definisanje tipova i struktura podataka je takode jedan od bitnih faktora čitljivosti jezika. Na primer zapisi kao strukturni tipovi omogućavaju da se mnogo preglednije opišu raznorodne strukture podataka nego kada se za to koriste vektori i numerički tipovi podataka.

➤ Sintaksa jezika

Sama sintaksa jezika je možda jedan od najbitnijih faktora njegove čitljivosti. Na primer koncept korišćenja zagrada `begin` i `end` za definisanje skupa naredbi u okviru upravljačkih struktura (Pascal), u novijim strukturnim jezicima (FORTRAN 77, Ada) zamenjen je konceptom ključnih reči kojima se otvaraju ili zatvaraju pojedini delovi strukture (`if` i `end if`, `loop` i `end loop` i sl.).

Time se znatno povećava čitljivost jezika posebno u slučaju kada se upravljačke strukture umeću jedna u drugu.

➤ Jednostavnost pisanja programa

Mnogi od faktora koji utiču na čitljivost programa utiču i na to koliko se na nekom jeziku lako mogu pisati programi. Međutim, ovaj problem mora da se razmatra u kontekstu problema za koji se određeni program piše. Postoje programski jezici više ili manje prilagođeni određenoj oblasti tako da se većom ili manjom lakoćom mogu koristiti

za pisanje određenih programa. Na primer jednostavnost pisanja programa u COBOL-u je mnogo veća kada se radi o programima za poslovnu obradu u odnosu na programe za neka numerička izračunavanja. Ipak, izdvojićemo neke od najznačajnijih faktora od koji generalno zavisi lakoća pisanja programa na jednom programskom jeziku.

➤ Jednostavnost i ortogonalnost

Ako u programskom jeziku postoji mnogo različitih koncepata obično se događa da neke od njih programeri jednostavno ignorišu i ne koriste. Mali broj jednostavnih koncepata i određeni skup usklađenih pravila za njihovo kombinovanje daju mnogo bolje rezultate od jezika sa mnogim raznovrsnostima.

➤ Podrška apstrakciji podataka

Apstrakcija, kao opšti princip, podrazumeva isticanje globalnih karakteristika, a skrivanje specifičnih. To je jedan od ključnih koncepata u savremenim metodologijama modularnog i objektnog programiranja, a takođe i programskim jezicima. Jedan nivo apstrakcije u programskim jezicima postiže se pomoću potprograma. Ovaj koncept se kod programskih jezika dalje razvija preko bloka, modula i klasa kao sredstva apstrakcije. Jednostavnost pisanja programa u direktnoj je zavisnosti od toga koliko su određenom programskom jeziku prisutni koncepti za apstrakciju podataka.

➤ Izražajnost

Kod različitih programskih jezika kada se govori o izražajnosti to može da podrazumeva više različitih koncepata. Obično se pri tome ocenjuju operatori samog jezika. Na primer u FORTRAN-u 90 postoje naredbe za vektorska izračunavanja koje omogućavaju da se veoma jednostavno napišu programi koji u drugim jezicima zahtevaju složene procedure. Ili, u C-u je notacija `br++` mnogo jednostavnija za korišćenje i kraća od naredbe `br = br + 1`. Takođe, `for` naredba je mnogo jednostavniji način da se napiše petlja sa određenim brojem prolaza od `while` strukture. Sve su to faktori koji doprinose lakšem pisanju programa.

### 1.10.2. Pouzdanost jezika

Pouzdanost jezika je koncept koji je dosta teško definisati. Obično se ovde svrstavaju one karakteristike jezika koje utiču na to da se što je moguće više smanji broj grešaka koje mogu da se pojave u fazi izvršavanja programa. Zbog toga savremeni jezici podržavaju takozvano rano povezivanje, odnosno povezivanje u fazi kompilacije programa. Naime, nastoji se da se što je moguće više grešaka otkrije u fazi kompiliranja programa i time izbegnu greške u fazi izvršavanja programa. Osnovni mehanizam koji se pri tome koristi je koncept tipova podataka.

#### Tipovi podataka

U okviru savremenih programskih jezika zastupljen je koncept jakih tipova podataka koji podrazumeva eksplicitne definicije tipova i tipove zatvorene u odnosu na operacije koje se mogu izvršavati nad elementima tipa. Ovaj koncept takođe

podrazumeva javne (eksplicitne) konverzije tipova podataka. Atributi vezani za tip koriste se u fazi prevođenja programa i omogućavaju otkrivanje grešaka na nivou dodeljivanja, konverzija i operacija u okviru izraza. Na taj način koncept tipova podataka postaje faktor pouzdanosti jezika.

### Obrada izuzetaka

Pod obradom izuzetaka u okviru programskih jezika podrazumevaju se mehanizmi koji omogućavaju odgovor programa na greške u fazi izvršavanja bilo da se radi o softverskim greškama ili prekidnim signalima samog sistema. U savremenim jezicima zastupljeni su strukturni mehanizmi za obradu izuzetaka koji omogućavaju da paralelno sa kodom koji se izvršava u regularnom režimu rada sistema definišemo i njegovo ponašanje u slučaju izuzetaka. Ovi mehanizmi su očigledno u funkciji pouzdanosti jezika i nivo na kome su postavljeni određuje i kvalitet jezika.

### Pseudonimi (*Aliasing*)

Kažemo da programski jezik dozvoljava definisanje pseudonima, odnosno aliasing, u slučaju kada postoji mogućnost da se ista memorijska lokacija referencira na više različitih načina u istom programu. U mnogim jezicima su prisutni ovakvi koncepti. Na primer u FORTRAN-u opisom equivalence mogu se istoj memorijskoj lokaciji dodeliti različita imena. Takode se u Pascal-u preko različitih pokazivača može referencirati ista memorijska lokacija. Očigledno je da aliasing negativno utiče na pouzdanost jezika i zbog toga se u mnogim jezicima svodi na najmanju moguću meru.

### 1.10.3. Efikasnost jezika

U mnogim slučajevima odluka o kvalitetu određenog programskog jezika zavisi od njegove efikasnosti. Efikasnost može biti razmatrana sa više različitih aspekata. Možemo govoriti o efikasnosti programa napisanih na određenom programskom jeziku, o efikasnosti samog kompilatora kao i o tome koliko je efikasan ceo proces razvoja aplikacija na određenom programskom jeziku. Koji će od nabrojanih aspekata biti primaran u mnogome zavisi od toga čemu je namenjen program koji se piše u određenom jeziku. U slučaju programa namenjenih radu u realnom vremenu najbitniji faktor je efikasnost i pouzdanost samog programa koja može da bude plaćena sporijim i skupljim procesom pripreme programa. Efikasnost samog kompilatora može da bude primarna kada se radi u nekom razvojnom okruženju kada je potrebno često menjati programe i ponovo ih prevoditi. U savremenim uslovima rada kada se softver plasira kao veoma skup proizvod efikasnost samog razvojnog okruženja i to da li koncept jezika omogućava modularnost i apstrakciju, odnosno višestruko korišćenje već definisanog softvera, postaje veoma bitan faktor. Mnogi novi koncepti prisutni kod savremenih programskih jezika su u funkciji efikasnosti celokupnog procesa projektovanja aplikacija. Zbog toga se danas obično ne govori samo o programskom jeziku već o celokupnim integrisanim razvojnim okruženjima koja obuhvataju sve softverske alate potrebne za brzi razvoj aplikacija. Ova okruženja obično objedinjuju editor, kompilator, linker i debager i uz jedan vizuelni korisnički interfejs omogućavaju efikasno programiranje.

### 1.11. Karakteristike programskih jezika

Savremeni programski jezici opšte namene odlikuju se nekim zajedničkim karakteristikama čija će nam sistematizacija poslužiti da ujedno izvršimo i sistematizaciju osnovnih koncepata koji će biti razmatrani u narednim poglavljima ove knjige.

#### *Formalno definisana sintaksa programskog jezika*

Formalni opis sintakse programskog jezika je osnova za njegovu standardizaciju, kontrolisani razvoj jezika, njegovo razumevanje i primenu. Formalni opis jezika je osnova za projektovanje prevodilaca i razvoj standardizovanih procedura i metoda za leksičku, sintaksnu i semantičku analizu.

#### *Jaki tipovi podataka*

Tipovi podataka su jedan od najvažnijih elemenata programskih jezika. Kod savremenih jezika zastupljen je koncept jakih, zatvorenih tipova podataka koji podrazumeva eksplicitne definicije tipova podataka. Informacija o tipu koristi se još u fazi prevođenja programa i postaje sredstvo pouzdanosti jezika. Sistem tipova podataka zasniva se na skupu elementarnih tipova podataka od kojih se dodatnim mehanizmima definišu novi, korisnički tipovi podataka.

#### *Strukturni tipovi podataka*

Rad sa osnovnim strukturama podataka kod savremenih jezika podveden je pod koncept strukturnih tipova podataka. Obično postoje mehanizmi za definisanje tipova podataka za rad sa jednodimenzionalnim i višedimenzionalnim jednorodnim strukturama kakvi su vektori i matrice (*array*), slogovima kao višerodnim strukturama podataka (*record*), datotekama (*file*) i skupovima (*set*).

#### *Upravljačke strukture*

U domenu definisanja toka programa danas je u potpunosti prihvaćen koncept strukturnog programiranja tako da u okviru svakog od programskih jezika obično postoje standardne upravljačke strukture za definisanje selekcija (*if*, *case*) i iteracija (*for*, *while*).

#### *Potprogrami*

Koncept potprograma omogućava razbijanje programa na manje, funkcionalno kompaktne celine koje se mogu višestruko koristiti u istom programu, a nekad čak i u različitim programima. U okviru programskih jezika obično postoje koncepti za definisanje funkcija i procedura kao potprograma opšteg tipa.

### *Moduli*

Moduli kojima se obuhvataju strukture podataka i procedure i funkcije kojima se realizuju osnovni algoritmi za rad sa tim podacima su osnovno sredstvo apstrakcije podataka u savremenim programskim jezicima. Koncept modula obično podrazumeva i mehanizme za zatvaranje kojima se postiže da se podacima u okviru modula može pristupiti samo preko funkcija i procedura samog modula. Funkcije i procedure su jedini interfejs sa podacima modula. Ovako zatvoreni moduli imaju sve attribute tipa podataka pa je koncept poznat pod imenom apstraktni tipovi podataka.

#### *Mehanizmi za konkurentno programiranje*

Mehanizmi za definisanje više tokova programa u okviru jedne aplikacije kao i sredstva za njihovu sinhronizaciju i komunikaciju su danas jedan od osnovnih koncepata u okviru savremenih programskih jezika.

#### *Mehanizmi niskog nivoa*

Od savremenih računara se često zahteva rad u okruženju drugih sistema i uređaja. To su takozvani ugrađeni (*embedded*) sistemi u okviru kojih je potrebna sprega niskog nivoa računara i drugih uređaja. Savremeni programski jezici, namenjeni ovakvim primenama, raspolažu mehanizmima niskog nivoa za pristup registrima procesora, upravljanje formatima podataka i pristup određenim memorijskim lokacijama. Može se reci da jezici visokog nivoa imaju neke karakteristike asemblerskih jezika koje im omogućavaju bolju spregu sa drugim uređajima u sistemu.

#### *Mehanizmi za obradu grešaka i izuzetaka (Exception Handling)*

Novi programski jezici obično raspolažu strukturnim mehanizama za obradu izuzetaka. Ovi koncepti omogućavaju da se paralelno uz kod koji se regularno izvršava definiše i kod koji se izvršava u slučaju pojave određenog izuzetka. Pored standardnih hardverskih izuzetaka koji su obično odgovor na standardne prekidne signale procesora kod novijih jezika postoji mogućnost i obrade softverskih izuzetaka koje opisuje sam programer.

#### *Standardni skup u/i procedura*

Mehanizmi za ostvarivanje osnovnih ulazno-izlaznih aktivnosti obično nisu deo standarda programskog jezika. Međutim, realizacija savremenih programskih jezika podrazumeva i definisanje skupa modula koji obuhvataju osnovne procedure za ostvarivanje ulazno-izlaznih aktivnosti.

## **2. Elementi programskih jezika**

### **2.1. Uvod u elemente programskih jezika:**

Prilikom definicije jezika polazi se od osnovnog skupa znakova, *azbuke* jezika koja sadrži sve završne simbole (terminalne simbole) jezika. Nad azbukom jezika definišu se ostali elementi jezika, konstante, rezervisane reči, identifikatori od kojih se dalje grade druge složene sintaksne kategorije kao što su npr. opisi, upravljačke naredbe i sl.



### 2.1.1. Azbuka jezika

Azbuka jezika predstavlja osnovni skup simbola (znakova) od kojih se grade sve sintaksne kategorije jezika. Broj znakova azbuke se obično razlikuje od jezika do jezika i kreće se od najmanje 48 do 90 znakova. Azbuka programskog jezika obično obuhvata skup velikih i malih slova engleske abecede, skup dekadnih cifara i određeni skup specijalnih znakova. Dok je kod starijih programskih jezika bilo uobičajeno da se koriste samo velika slova (npr. FORTRAN IV), danas se skoro redovno dozvoljava i ravnopravno korišćenje malih slova abecede. Azbuke programskih jezika se najviše razlikuju po skupu specijalnih znakova koje obuhvataju. Narednih nekoliko primera azbuka određenih programskih jezika to najbolje ilustruje.

```

Azbuka jezika Algol 60
slova: A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
      a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
cifre: 0|1|2|3|4|5|6|7|8|9
logičke vrednosti: true|false
specijalni znaci:
  znaci aritmetičkih operacija: +, -, /, *, %, ^
  znaci relacija: <, >, =, <=, >=, ≠
  znaci logičkih operacija: ∧, ∨, ¬
  razdelnici: ;, ,, ,, , , , :=
    
```

```

zagrade: (, ), [, ], begin, end
    
```

#### Azbuka jezika PL/1:

```

slova: A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|_|$#@
cifre: 0|1|2|3|4|5|6|7|8|9
specijalni znaci: b | = | + | - | * | / | ( | ) | . | | $ |
                : | : | _ | & | ! | > | < | ? | _
    
```

#### Azbuka jezika Pascal

```

slova: "A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|"
      "N"|"O"|"P"|"Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"|"
      "a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"|"
      "n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"
cifra: "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
specijalni znak: " " | "-" | "." | ":" | ";" | "<" | ">" | "<=" | ">=" | "." | ":" | ";" |
                " " | "-" | "." | ":" | ";" | "<" | ">" | "<=" | ">=" | "." | ":" | ";" |
rezervisana reč: "AND"|"ARRAY"|"BEGIN"|"CASE"|"CONST"|"DIV"|"DOWNTO"|"
                "DO"|"ELSE"|"END"|"FILE"|"FOR"|"FUNCTION"|"GOTO"|"IF"|"IN"|"LABEL"|"
                "MOD"|"NIL"|"NOT"|"OF"|"OR"|"PACKED"|"PROCEDURE"|"PROGRAM"|"RECORD"|"
                "REPEAT"|"SET"|"THEN"|"TO"|"TYPE"|"UNTIL"|"VAR"|"WHILE"|"WITH"
    
```

#### Azbuka jezika Ada

```

velika slova: A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
cifre: 0|1|2|3|4|5|6|7|8|9
specijalni znaci: "#|$|_|'(|)|*|+|.|-|/|:|<|=|>|_|/|
znak blanko
mala slova: a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
drugi specijalni znaci: !|!|$|_|@|_|(|)|^|'|(|)|~
    
```

U skladu sa opštim zahtevima za standardizacijom danas se obično skup specijalnih znakova azbuke programskog jezika standardizuje i svodi na skup specijalnih znakova međunarodnog standardnog koda ISO7 ( ASCII kod dodatak F ).

ISO7		JUS	
@	^	Ž	ž
[	ç	Š	š
]	)	Č	č
\		Đ	đ
^	~	Ć	ć

! @ # \$ % ^ & \* ( ) + ? / > < { | } ! = - | \ ~ . " . ' !

često se pored osnovnog skupa specijalnih znakova koriste i složeni simboli, obično dvoznaci kao npr:

**\*\* <= >= << >> <> >< := /\* \*/**

U nekim programskim jezicima (FORTRAN), zbog nedovoljnog broja odgovarajućih znakova umesto specijalnih znakova koriste se posebne simboličke oznake kao

**.EQ. , .NE. , .GT. , .GE. , .LT. , .LE.**

### 2.1.2. Rezervisane reči

Nizovi znakova azbuke koji u programu imaju određeni smisao nazivaju se *leksema*. Leksema predstavlja leksičku jedinicu koja odgovara jednoj reči ili grupi reči koja obrazuje gramatičku ili frazeološku celinu (npr. goto). Leksema može da bude i samo jedan znak.

Reč jezika, čije značenje je utvrđeno pravilima tog jezika i ne može se menjati ni u jednom programu pisanom na tom jeziku, naziva se *rezervisana reč*.

Rezervisane reči mogu da budu *zabranjene*, kada se ne mogu koristiti kao identifikatori u programu. Međutim, i u jezicima u kojima je to dozvoljeno ne preporučuje se korišćenje ključnih reči kao identifikatora jer može da smanji preglednost programa, a u nekim slučajevima da dovede i do ozbiljnih grešaka u programu. Poznat je, na primer, slučaj greške sa DO naredbom koji je doveo do pada letilice iz satelitskog programa *Geminy* 19. U programu za upravljanje letilicom stajala je DO naredba napisana kao:

**DO 10 I =1. 10**

umesto ispravnog koda

**DO 10 I =1. 10.**

Greška pri prevođenju međutim nije otkrivena jer je leksički analizator ovu liniju koda protumačio kao naredbu dodeljivanja u kojoj se promenljivoj DO10I dodeljuje vrednost 1.10.

### Rezervisane reči jezika Ada

abort	abs	access	all	and	array
at					
begin	body	case	constant	declare	delay
do	else	elsif	end	entry	digits
exit					exception
for	function	generic	goto	if	in
limited	loop	mod	new	not	is
of					null
or	others	out	package	pragma	private
raise	range	record	rem	renomes	return
select	separate	subtype	task	terminate	then
use	when	while	with	xor	type
					procedure
					reverse

**Rezervisane reči jezika C**

int	char	float	double	struct	union
long					
short	unsigned	auto	extern	register	typedef
static					
goto	return	sizeof	break	continue if	else
for	do	while	switch	case	default
entry					

Kod nekih programskih jezika ( COBOL ) postoje kategorije obaveznih reči i neobaveznih reči. Obavezne reči ne smeju da budu izostavljene iz naredbe u kojoj je po sintaksi definisana njihova upotreba i na osnovu njih kompilator analizira i prevodi naredbu, dok neobavezne reči imaju samo dokumentacioni karakter i upotrebljavaju se da dopune naredbu i njen tekst približe govornom jeziku. Razmotrimo sledeći primer definicije naredbe READ kojom se čita slog datoteke:

**READ ime-datoteke [NEXT RECORD [INTO ime-podatka] AT END iskaz**

Reči READ, NEXT, INTO i END su obavezne prilikom pisanja odgovarajućih delova naredbe, njima je određena sintaksa naredbe, odnosno koristi ih sintaksni analizator pri prevodenju naredbe. Reči RECORD i AT su neobavezne, ali se mogu koristiti da bi se povećala jasnoća naredbe.

**2.1.3.Konstante**

Bilo koji niz znakova u programu, posmatran nezavisno od njegovog logičkog značenja, nad kojim se mogu izvršavati određena dejstva (operacije) naziva se *podatak*. Deo podatka nad kojim se mogu izvršavati elementarne operacije naziva se *element podatka*. Elementu podatka u matematici približno odgovara pojam skalarne veličine. Podatak je uređeni niz znakova kojim se izražava vrednost određene veličine. Veličina koja u toku izvršavanja programa uvek ima samo jednu vrednost, koja se ne može menjati, naziva se *konstanta*. Kao oznaka konstante koristi se ona sama. U nekim programskim jezicima (Pascal, Ada, C) postoji mogućnost imenovanja konstante. Konstantama se dodeljuju imena koja se u programu koriste umesto njih.

Tipovi konstanti koje se mogu koristiti u određenom programskom jeziku određeni su tipovima podataka koje jezik predviđa. Slede neki primeri:

Celobrojne dekadne konstante:

**1; 50; 153; +55; -55**

Realne konstante u fiksnom zarezu:

**3.14; 3.0; -0.314; -.314; +.314**

Realne konstante u pokretnom zarezu:

**3.14 E 0; -0.314 E 1**

**-.314 E +0; +.314 E -2 (FORTRAN, Pascal, Ada, C)**

Realne konstante dvostruke tačnosti:

**3.14 D 0; -3.14 D1; +.314 D2 (FORTRAN)**

Kompleksne konstante:

**(3.14, 0.13); (0, -23) (FORTRAN)**

**3.13, 013 I; 0. -23 I (PL/I)**

Binarne konstante (PL/I):

**10111B; -0.0001B (u fiksnom zarezu)**

**11011 E -4B; 111.001 E 2 B (u pokretnom zarezu)**

Oktalne konstante (C):

**0567; 0753; 0104**

Heksadecimalne konstante (C):

**0XIF; 0XAA; 0XII**

Long konstante (C):

**123l; 527l; 321l; +147l**

Logičke konstante:

**true; false (Pascal, Ada)**

**.TRUE.; .FALSE. (FORTRAN)**

Znakovne konstante:

**' A ' ; ' B ' (Pascal, Ada, C)**

String konstante:

**"Beograd"; "Alfa l"(Ada,C)**

Binarne string konstante:

**'10001111'B, '11000011'B (PL/I)**

Simboličke konstante:

**ZERO; ZEROS; SPACE (COBOL)**

#### **2.1.4. Promenljive**

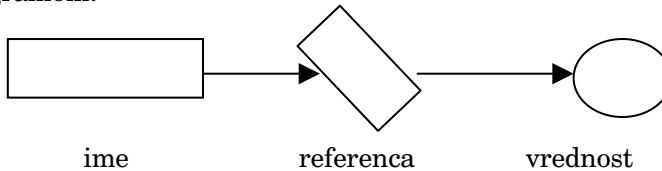
Veličine čije se vrednosti menjaju u toku izvršavanja programa nazivaju se *promenljive*. Promenljivoj se u programu dodeljuje ime, u svakom trenutku ona je definisana svojom vrednošću. Kažemo da je svaka promenljiva u programu povezana sa tri pojma:

*imenom* -identifikatorom promenljive.

*referencom* -pokazivačem koji određuje mesto promenljive u memoriji i

*vrednošću* -podatkom nad kojim se izvršavaju operacije.

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



Na primer naredbu dodeljivanja  $x := 3.141592$  čitamo:  $x$  dobija vrednost 3.141592, pri čemu imamo u vidu da je  $x$  simboličko ime memorijske lokacije gde se u toku izvršavanja programa pamti vrednost 3.141592. Takođe možemo reći:  $X$  ima vrednost broja  $Pi$ , pri čemu razlikujemo:

- vrednost broja 3.141592.
- uobičajeno ime broja  $Pi$ ,
- adresu memorijske lokacije u kojoj se pamti vrednost broja  $Pi$ ,
- ime promenljive  $X$ , identifikator koji se u datom programu koristi kao ime promenljive koja ima datu brojnu vrednost.

#### 2.4.1.1. Identifikatori

Identifikatori su uvedene reči kojima se imenuju promenljive ali i druge sintaksne kategorije kao što su na primer potprogrami, programski moduli, klase i sl. U svim programskim jezicima postoji slična konvencija za pisanje identifikatora. Identifikatori su nizovi koji se obično sastoje od slova i cifara i obavezno započinju slovom. Ovo ograničenje omogućava jednostavniju implementaciju leksičkog analizatora i razdvajanje identifikatora od drugih sintakasnih kategorija (numeričkih i znakovnih konstanti npr.). U nekim jezicima dozvoljeno je da se i neki specijalni znaci pojave u okviru identifikatora. Najčešće je to crtica za povezivanje "-" kojom se postiže povezivanje više reči u jedan identifikator. Na primer, u jeziku C crtica za povezivanje se može koristiti na isti način kao i slova, što znači da identifikator može da započne ovim znakom. Slede primeri nekih identifikatora:

ALFA    A    B1223    Max\_vrednost    PrimerPrograma

### 2.1.5. Komentari

U svim programskim jezicima postoji mogućnost proširenja izvršnog koda programa komentarima kojima se kod dodatno pojašnjava. Ovi komentari se u toku prevođenja programa ignorišu od strane kompilatora i ne ulaze u sastav izvršnog koda koji se generiše prevođenjem programa. Međutim komentari su veoma važan deo programa kojim se podešava njegova dokumentarnost, i bitno utiču na efikasnost analize programa. Konvencija za zapisivanje komentara se razlikuje od jezika od jezika. Slede primeri komentara u nekim programskim jezicima :

Komentari se u Pascal-u zapisuju između velikih zagrada.

Komentari u jeziku Ada započinje sa dve crtice i mogu da se nađu bilo gde u programu.

```
/* Ovo je primer komentara u jeziku C */  
// Kratak C komentar u okviru jednog reda  
<!-- Ovo je komentar u HTML kodu -->
```

U jeziku Algol 60 komentari se eksplicitno naglašavaju pomoću ključne reči comment.

Sledeći primer ilustruje ovaj drugi način upotrebe komentara u Algo1-u:

```
for i := 1 step 1 unt1 N do  
begin comment Pocetak tela petlje;  
Y[i] := Y[i] -m;  
Y[i] :=X[i] * Y[i];  
end kraj tela petlje;
```

## 2.2. TIPOVI PODATAKA

Jedan od najznačajnijih pojmova u okviru programskih jezika je pojam tipa podataka. Sve promenljive koje se koriste u jednom programu imaju atribut tipa, što određuje skup vrednosti koji im se može dodeljivati, format predstavljanja ovih vrednosti u memoriji računara, skup osnovnih operacija koje se nad njima mogu izvršavati i veze sa drugim tipovima podataka. Na primer, promenljivoj koja pripada celobrojnom tipu mogu se kao vrednosti dodeljivati samo celi brojevi iz određenog skupa. Ovi brojevi se u računaru registruju po unapred definisanom formatu koji zahteva jednu polureč, reč ili dvostruku reč za predstavljanje jednog broja. Nad tako definisanim podacima mogu se izvršavati osnovne aritmetičke operacije sabiranja, oduzimanja, množenja, deljenja, stepenovanja, kao i neke specifične operacije kao što je određivanje vrednosti jednog broja po modulu drugog.

Koncept tipova podataka prisutan je, na neki način, već kod simboličkih asemblerskih jezika gde se za definiciju tipa koriste implicitne definicije preko skupa specijalnih znakova kojima se određuju podaci različitog tipa.

Kod viših programskih jezika tip podataka može da bude definisan implicitno po određenom automatizmu (dogovoru), preko implicitnog opisa tipa ili preko eksplicitnih definicija tipova. Kao primer može da se uzme jezik FORTRAN kod kojeg postoje sva tri tipa definicija.

Na osnovu toga kako je postavljen koncept tipova podataka programski jezici mogu da se svrstaju u dve grupe, na programske jezike sa slabim tipovima podataka i na jezike sa jakim tipovima podataka.

### 2.2.1. Koncept slabih tipova

U slučaju jezika sa slabim tipovima podataka informacija o tipu koristi se, i korektna je samo na mašinskom nivou. Ovako postavljen koncept podrazumeva sledeće mogućnosti:

(1) Operacija koja se od strane mašine prihvati kao korektna, na nivou izvornog koda programa može da bude potpuno nekorektna. Razmotrimo sledeći primer:

```
var
c : char;
c := 4;
```

Promenljiva *c* definisana je da pripada tipu *char*, što podrazumeva da joj se kao vrednosti dodeljuju znaci kao podaci. Međutim umesto korektnog dodeljivanja *c := '4'*, promenljivoj *c* je dodeljena vrednost broja 4 kao konstante celobrojnog tipa. Kod jezika sa slabim tipovima podataka kompilator ne otkriva ovu grešku i informaciju o tipu koristi samo na mašinskom nivou kada promenljivoj *c* dodeljuje vrednost jednog bajta memoriske lokacije u kojoj je zapisana konstanta 4. Očigledno je da ovako postavljen koncept tipova može da dovede do veoma ozbiljnih grešaka u izvršavanju programa koje se ne otkrivaju u fazi kompiliranja programa.

(2) Koncept slabih tipova podrazumeva određeni automatizam u transformaciji tipova podataka u slučaju kada se elementi različitih tipova nalaze u jednom izrazu čija se vrednost dodeljuje promenljivoj određenog tipa. Razmotrimo sledeći primer:

```
var
x, y : real ;
i, j, k : integer;
i := x;
k := x - j ;
```

Promenljive *x* i *y* su realne (tipa *real*), a *i*, *j* i *k* celobrojne (integer tipa). Naredbom

*i := x* ; vrši se dodeljivanje vrednosti tipa *real* promenljivoj celobrojnog tipa. Kod jezika sa slabim tipovima ovo dodeljivanje je dozvoljeno iako se pri tome *x* svodi na drugi format i pravi greška u predstavljanju njegove vrednosti. Kod ovako postavljenog koncepta tipova da bi se napisao korektan program potrebno je tačno poznavati mehanizme transformacije tipova. U drugoj naredbi iz primera (*k := x - j* ; ) od broja *x* koji je tipa *real* treba oduzeti broj *j*, tipa *integer* i rezultat operacije dodeliti promenljivoj tipa *integer*. Da bi smo bili sigurni u korektnost rezultata potrebno je da znamo redosled transformacija koje se pri tome izvršavaju, odnosno da li se prvo *x* prevodi u *integer* i onda izvršava oduzimanje u skupu celih brojeva i vrednost rezultata dodeljuje promenljivoj tipa *integer* ili se *j* prevodi u tip *real*, izvršava sabiranje u skupu realnih brojeva, a zatim rezultat prevodi u tip *integer* i dodeljuje promenljivoj *k*.

(3) Koncept slabih tipova nekad omogućava postizanje određenih efekata naredbama koje imaju sasvim drugu namenu. Npr .naredbom

**i:=(k sh 12) or 1);**

promenljivoj i se dodeljuje vrednost koja se dobija tako što se binarni ekvivalent vrednosti k pomera za 12 mesta ulevo (na taj način izdvajaju se samo četiri cifre binarnog broja) i u krajnju desnu poziciju upisuje vrednost 1, što je rezultat logičke or operacije sa binarnim ekvivalentom broja 1.

Očigledno je da koncept slabih tipova podataka dopušta puno sloboda kod zapisivanja izraza u naredbama dodeljivanja, međutim cena te slobode je nejasan program sa skrivenim transformacijama, bez mogućnosti kontrole i korišćenja informacije o tipu u fazi kompiliranja programa.

### 2.2.2. Koncept jakih tipova podataka

Koncept jakih tipova podataka obuhvata nekoliko osnovnih principa: .Tip podataka određuju sledeći elementi:

- skup vrednosti,
  - format registrovanja podataka,
  - skup operacija koje se nad podacima mogu izvršavati,
  - skup funkcija za uspostavljanje veza sa drugim tipovima podataka.
- ◆ Sve definicije tipa moraju da budu javne, eksplicitne. Nisu dozvoljene implicitne definicije tipova.
  - ◆ Objektu se definiše samo jedan tip.
  - ◆ Dozvoljeno je dodeljivanje vrednosti samo odgovarajućeg tipa.
  - ◆ Dozvoljene su samo operacije obuhvaćene tipom.
  - ◆ Tip je zatvoren u odnosu na skup operacija koji obuhvata. Ove operacije se mogu primenjivati samo nad operandima istog tipa. Mešoviti izrazi nisu dozvoljeni.
  - ◆ Dodeljivanje vrednosti raznorodnih tipova i operacije nad raznorodnim operandima moguće je samo uz javnu upotrebu funkcija za transformaciju tipa.

U sledećim primerima date su naredbe koje po ovom konceptu nisu dozvoljene:

```
var
x: real ; --x je tipa real ;
i : integer; --i je celobrojnog tipa;
b: Boolean; --b je logickog tipa;
c: char; --c je znakovnog tipa;
i := 'A'; --nekorektno, jer se promenljivoj
           --celobrojnog tipa dodeljuje vrednost
           --znakovnog tipa;
x := i + 5.0; --nekorektno, jer su operandi
             --razlicitog t;pa;
c := 10;
i := i or 7; --nekorektno, jer je logicka operacija
            --upotrebljena nad vrednostima
            --celobrojnog tipa;
```



Koncept jakih tipova povećava pouzdanost, dokumentarnost i jasnoću programa. Korektnost programa obezbeđena je na apstraktnom nivou. Kako se zahteva eksplicitna upotreba operacija za transformaciju tipa, onda je nedvosmisleno je jasno da je određena transformacija na nekom mestu namerna i potrebna. Ovaj koncept omogućava da se informacija o tipu može koristiti u fazi kompiliranja programa i na taj način postaje faktor pouzdanosti programa.

### 2.2.3. Ekvivalentnost tipova

Kada kompilator jezika sa jakim tipovima podataka treba da obradi naredbu dodeljivanja oblika:

```
x := izraz
```

on vrši dodeljivanje samo u slučaju ekvivalentnosti tipa promenljive sa leve strane dodeljivanja i rezultata izraza na desnoj strani naredbe. Pri tome koriste se pravila po kojima je definisana ekvivalentnost tipova. Razlikuju se

- ◆ strukturna ekvivalentnost i
- ◆ imenovana (eksplicitna) ekvivalentnost.

Strukturna ekvivalentnost podrazumeva da su leva i desna strana u naredbi i dodeljivanja ekvivalentne strukturno - svode se na isti broj memorijskih lokacija.

Na primer, dodeljivanje definisano sa:

```
var  
i : integer;  
u : real ;  
u :- i ;
```

je dozvoljeno. U krajnjem slučaju obe promenljive pripadaju tipovima podataka definisanim nad istim elementarnim tipom i obuhvataju vrednosti koje su strukturno jednake, jer se radi o numeričkim podacima koji zauzimaju četiri bajta u memoriji.

#### Imenovana ekvivalentnost

U ovom slučaju zahteva se da promenljiva na levoj strani naredbe dodeljivanja pripada eksplicitno istom tipu kome i izraz na desnoj strani naredbe. U slučaju eksplicitne ekvivalentnosti vrednost jedne promenljive se može dodeliti drugoj samo ako obe eksplicitno pripadaju istom tipu, što ilustruje sledeći primer:

```
var  
i .U : real ;  
u :- i ;
```

Eksplicitnom ekvivalentnošću tipova postiže se veća pouzdanost jezika. U ovom slučaju nisu potrebne posebne procedure po kojima bi se ispitivala strukturna ekvivalentnost. Međutim, kada je potrebno vrednost promenljive ili izraza dodeliti promenljivoj koja mu ne odgovara po tipu ovaj koncept zahteva korišćenje funkcija za transformisanje tipova.

Strukturalna ekvivalentnost dozvoljava slobodnije korišćenje koncepta jakih tipova i jednostavniji prelazak na ovaj koncept sa koncepta slabih tipova. U ovom slučaju koncept jakih tipova samo omogućava veću jasnoću programa, a ne povećava njegovu pouzdanost. Zbog toga kod novijih jezika uz koncept jakih tipova podataka ide i eksplicitna ekvivalentnost.

#### 2.2.4. Elementarni tipovi podataka

U okviru svakog programskog jezika, sistem tipova podataka zasniva se na skupu osnovnih tipova podataka nad kojima se dalje definišu izvedeni tipovi, podtipovi, strukturni tipovi i specifični apstraktni tipovi podataka. Skup osnovnih tipova podataka se obično svodi na tipove podataka za rad sa elementarnim numeričkim podacima (celi i realni brojevi), znakovnim podacima (pojedinačni znaci ASCII koda) i logičkim vrednostima (*true* i *false*).

##### *Celobrojni tipovi (INTEGER)*

Skoro bez izuzetaka, u svim programskim jezicima postoji jedan ili više tipova podataka za rad sa celim brojevima koji obično u nazivu imaju reč INTEGER.

Obično postoji osnovni (standardni) tip INTEGER koji se zavisno od realizacije odnosi na određeni opseg celih brojeva. Nekad je to opseg koji odgovara formatu jedne polureči ili formatu jedne reči. U odnosu na ovaj osnovni celobrojni tip često postoji mogućnost definisanja i drugih celobrojnih tipova koji se odnose na neki kraći ili prošireni format.

Tip operacije	Operacija	Operator
Multiplikovane operacije	stepenovanje	**
	množenje	*
	deljenje	/
	ostatak deljenja	mod
Aditivne operacije	sabiranje	+
	oduzimanje	-
Unarne operacije	plus	+
	minus	-
Relacije	manje	<
	manje ili jednako	<=
	veće	>
	veće ili jednako	>=
	jednako	=
	nejednako	!=

##### *Logički tip podataka LOGICAL*

Logički tipovi podataka postoje kao osnovni tipovi podataka u svim novijim jezicima. Obično nose naziv LOGICAL ili BOOLEAN (Ada). Obuhvataju samo dve vrednosti *true* i *false*, nad kojima su definisane osnovne logičke operacije *not*, *and*, *or* i *xor*. Značajno je pomenuti da se u programskom jeziku Ada ovaj tip podataka svrstava u širu kategoriju diskretnih tipova tako da raspolaže istim skupom atributa i funkcija kao i svaki drugi diskretni tip podataka. Takođe važi i uređenost skupa vrednosti ovog tipa tako da je *false* < *true*. Sledeći primeri ilustruju neke od funkcija koje se mogu koristiti u ovom tipu podataka:

---

**BOOLEAN'FIRST =FALSE**  
**BOOLEAN'SUCC(FALSE) =TRUE**

### *Znakovni tipovi*

Za rad sa pojedinačnim znacima u okviru programskih jezika koriste se znakovni tipovi koji obično nose naziv CHARACTER (Ada) ili CHAR (Pascal). Ovi tipovi obuhvataju standardni skup znakova ASCII koda i određene operacije nad ovako definisanim podacima. Podrazumeva se uređenost skupa znakova u skladu sa uređenošću tablice ASCII koda tako da se kao osnovne operacije u ovim tipovima koriste relacije poređenja. U jeziku Ada tip CHARACTER se takođe svrstava u širu kategoriju diskretnih tipova podataka što podrazumeva i mogućnost upotrebe svih standardnih atributa i funkcija diskretnih tipova. U okviru znakovnih tipova se obično mogu koristiti funkcije kojima se uspostavlja veza između određenog znaka i njegove dekadne vrednosti u ASCII kodu. U jeziku Ada to je funkcija ORD. Npr:

**CHARACTER'ORD('A') =65**

**CHARACTER'ORD('O') =49**

### *Tip realnih brojeva (Real)*

Standardni tip za rad sa realnim brojevima u Programskim jezicima je tip Real. Obuhvata brojeve za čije se predstavljanje koristi standardni format pokretne tačke, a dozvoljene su standardne aritmetičke operacije :

+ sabiranje  
-oduzimanje  
\* množenje  
/ deljenje

Standardne funkcije ABS ( X ) i SQR ( X ) u slučaju argumenata tipa Real daju rezultat tipa Real, dok su rezultati standardnih funkcija: SIN(X), COS(X), TAN ( X ), EXP ( X ) , ARCTAN ( X ) i SQRT ( X ) tipa Real za bilo koje vrednosti argumenta X.

### 3. PROGRAMIRANJE

#### 3.1. PROCES PROGRAMIRANJA

U prethodnom poglavlju - *Uvod u programske jezike*, bilo je reči o istorijatu, tipovima i karakteristikama određenih poznatijih programskih jezika a na dalje ćemo

definisati pojam programiranja u širem smislu reči i to kao proces koji obuhvata sledeće faze: sastavljanje algoritamskih šema, pisanje i razradu programa, testiranje programa i izradu dokumentacije sa uputstvom za korišćenje programa.

U ovom poglavlju istaknute su faze koje se javljaju u procesu programiranja i preporuke za njihovo uspešno obavljanje. Treba istaći da je programiranje u širem smislu složen i raznovrstan proces, uslovljen tipom problema koji se rešava. I pored toga, moguće je navesti izvesne metode i principe koji se koriste u procesu programiranja kod većine problema.

Osnovni elementi o kojima treba voditi računa u toku programiranja su:

- greške u programiranju,
- brzina izvršavanja programa,
- ekonomično korišćenje memorijskog prostora,
- brzina programiranja.

Ovi elementi su nažalost dobrim delom protivurečni, pa se često moraju činiti kompromisi.

##### 3.1.1. Greške u programiranju.

U procesu programiranja treba voditi računa o raznim izvorima grešaka koje mogu onemogućiti dobijanje zadovoljavajućih rezultata. Izvori grešaka mogu biti:

- greške u postavljanju problema,
- greške u matematičkom modelu,
- nisu uzete u obzir sve moguće kombinacije ulaznih podataka,
- programer pripada timu i radi jedan deo velikog programa koji ne poznaje u celini. Neusaglašenost sa ostalim delovima programa može biti izvor grešaka u programu,
- greške programiranja u toku pisanja programa,
- greške kod prenošenja programa i podataka na kartice ili drugi ulazni medijum. (Programi koji obrađuju veliki broj ulaznih podataka moraju obuhvatiti kontrolu podataka, iako se time obim programa povećava i do 30%)
- greške operatora u toku propuštanja programa,
- greške računara.

##### 3.1.2. Brzina izvršavanja programa

Brzina izvršavanja programa zavisi od broja instrukcija u programu i od broja ponavljanja pojedinih delova programa (broja ciklusa). Brzina izvršavanja programa takođe zavisi od stepena angažovanosti centralne jedinice računara i njegove periferije. Multiprogramiranje i multiprocesiranje mogu znatno uticati na brzinu izvršavanja programa. Obučenosť operatora i vreme potrebno za njegove intervencije kod izvršavanja programa moraju se takođe uzeti u obzir.

Povećanje brzine izvršavanja programa može se ostvariti na više načina:

- pisanjem programa, sa što je moguće manjim brojem instrukcija,
- upotrebom instrukcija za čije je izvršavanje potrebno kraće vreme,
- pamćenjem pojedinih međurezultata, umesto njihovog ponovnog izračunavanja,
- minimalnim korišćenjem spoljnih memorija,
- pogodnom organizacijom podataka,
- korišćenjem odgovarajućih ulazno-izlaznih uređaja,
- korišćenjem namensko-orijentisanih sredstava za preliiminarnu obradu ulaznih podataka.

### 3.1.3. Ekonomično korišćenje memorijskog prostora

Postoji više načina za postizanje veće ekonomičnosti u korišćenju memorije:

- razbijanje programa na više nezavisnih delova (modula), ukoliko za to postoje mogućnosti. Pojedini moduli programa unose se pojedinačno u operativnu memoriju, izvršavaju se, a zatim se na njihova mesta unose drugi moduli,
- korišćenje zajedničkih zona memorije za razne grupe podataka,
- korišćenje cikličnih programskih struktura,
- pakovanje podataka.

### 3.1.4. Brzina programiranja

Brzina programiranja zavisi od više faktora: izbora programskog jezika, tipa i konfiguracije računara, a najviše od obučenosti programera.

Kako se jedan isti program može programirati na više načina, složenost pristupa u rešavanju takođe utiče na brzinu programiranja. Što je pristup logički jednostavniji, brzina programiranja je veća, jer se program lakše testira, koristi i razumljiviji je, mada se u tom slučaju dovodi u pitanje ekonomičnost korišćenja memorije, kao i brzina računanja.

Uporedo sa razvojem računске tehnike razvijaju se metode i postupci koji se koriste u pojedinim fazama procesa programiranja. Većina ovih metoda nezavisna je od jezika na kome se programiranje vrši, ali neke zavise od tipa i konfiguracije računarskog sistema.

### 3.1.5. Sastavljanje algoritama i programa

Obično sistem analitičar specificira u glavnim crtama problem koji treba da se reši uz pomoć računara. On formira globalni algoritam, bez razrade detalja, služeći se sistemskim dijagramima, tablicama odlučivanja i uopštenim grafičkim šemama. Tako formulisan problem preuzimaju programeri. Njihov zadatak je da razrade algoritam i sastave program za rešavanje postavljenog problema.

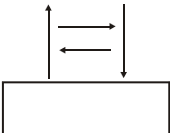
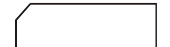





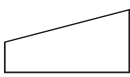


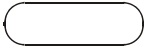
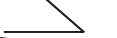

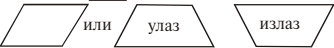
Ovde ćemo spomenuti samo najvažnije metode i principe koji se koriste u fazama sastavljanja algoritma i pisanja programa.

### 3.1.6. Sistemski dijagrami

Glavna svrha sistemskih dijagrama je definisanje ulaza i izlaza programa, odgovarajućih ulazno-izlaznih medijuma, kao i ukazivanje na funkciju programa, bez ulazanja u problem kako to računar radi. Sistemski dijagram se odnosi na jedan prolaz

(run) programa kroz računar. Pod tim se podrazumeva vremenski period u kome računar kontroliše jedan program i njime upravlja da prihvati određenu strukturu ulaznih podataka i da na svom izlazu da određenu strukturu izlaznih podataka. Kod crtanja sistemskih dijagrama koriste se standardni grafički simboli (Tabela 1.2).

Tabela 1.2. Elementi sistemskog dijagrama

	Smer kretanja podataka (informacioni tokovi)
	Obrada podataka
	Podaci na bušenim karticama
	Podaci na dokumentu (dokumenti na papiru i sve vrste izveštaja)
	Podaci na magnetnoj traci (indirektni pristup podacima)
	Podaci na spoljnom nosiocu informacije (OF LINE pamćenje podataka, na papiru, kartici, magnetnoj ili perforiranoj traci)
	Podaci na pisaču ili ekranu katodne cevi
	Podaci na ON LINE terminalu (podaci se unose u računar ili dobijaju sa računara preko ON LINE terminala)
	Podaci na papirnoj traci
	Sortiranje
	Tastatura (operacija koja koristi tastaturu ili prekidač)
	Prenos podataka preko komunikacionih uređaja
	Podaci na disku ili dobošu (poludirektni pristup podacima)
	Ulaz ili izlaz podataka

Kod crtanja sistemskih dijagrama treba voditi računa o:

- preglednosti,
- jasnom pripremanju ulaznih podataka sa odgovarajućim sažetim opisima datoteka,
- vremenskoj obradi i njenoj sinhronizaciji sa ostalim delovima sistemskog dijagrama,
- smeštajnim kapacitetima i količini polaznih podataka.

### 3.1.7. Organizacija programa

O organizaciji programa treba voditi računa već kod izrade uopštene grafičke šeme. Dalja razrada organizacije programa vrši se paralelno sa razradom grafičke šeme i pisanjem programa. Svaki program sadri tri faze koje čine posebne organizacione celine.

Prva faza priprema računar za obradu podataka. Druga faza se bavi obradom ulaznih podataka radi dobijanja izlaznih rezultata. U poslednjoj fazi završava se obrada i štampaju krajnji rezultati.

Sa stanovišta organizacije programa i redosleda učitavanja i štampanja ulaznih podataka i izlaznih rezultata, mogući su sledeći slučajevi:

*Faze su izolovane.* Učitavanje svih podataka obavlja se u prvoj fazi, obrada podataka je u drugoj, a štampanje svih izlaznih rezultata viši se u okviru treće faze.

*Prve dve faze se preklapaju.* Učitavanje ulaznih podataka obavlja se u nekoliko tačaka programa, između kojih se nalaze koraci obrade. Izlazni rezultati štampaju se u okviru treće faze. Slična organizacija programa je u slučaju kada se druga i treća faza preklapaju, a učitavanje podataka se vrši u okviru prve faze.

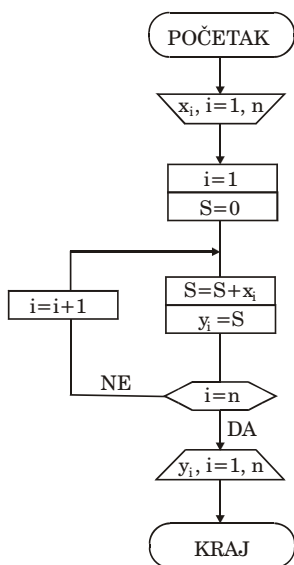
*Sve tri faze se preklapaju.* Program naizmenično učitava ulazne podatke i generiše izlazne rezultate, tj. faze čine ciklus koji se ponavlja i sastoji se iz čitanja, obrade i štampanja.

Pomenute tri faze u organizaciji programa ilustrovaćemo jednim primerom, koji ćemo rešiti u više varijanti.

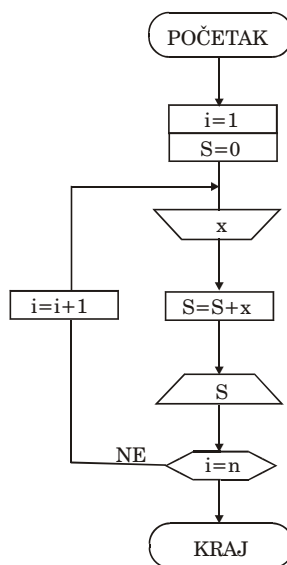
Dat je niz  $x = \{x_i\}$  od  $n$  elementa. Treba izračunati i štampati sve sume definisane relacijom

$$S_j = \sum_{i=1}^j x_i, \quad j = 1, \dots, n$$

Na sl. 2. ÷ 5. prikazane su grafičke šeme kojima se ovaj zadatak može rešiti, uz ilustraciju kada su faze izolovane ( sl. 2. ), kada se sve tri faze preklapaju ( sl. 2. ), i kada se po dve faze preklepaju ( sl. 3. i sl. 4.).



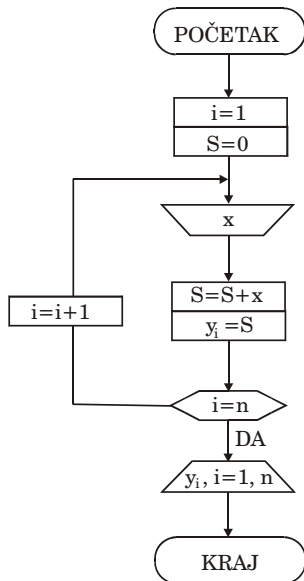
Sl. 2. Faze su izolovane



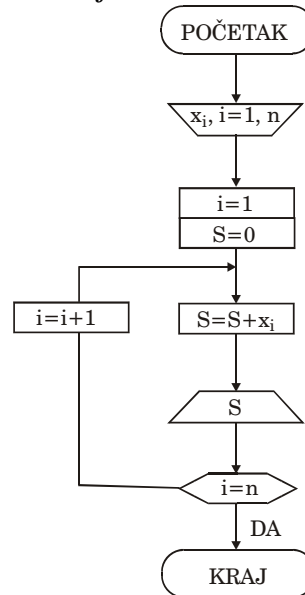
Sl. 3. Faze se poklapaju

Prednosti i nedostaci raznih varijanti organizacije programa su očigledne. U slučaju prve varijante (faze su izolovane) najviše se zauzima memorijski prostor, ali je vreme potrebno za učitavanje, obradu i štampanje najkraće u odnosu na ostale varijante. Kod drugog slučaja (faze se preklapaju) je obrnuto, dok su ostale dve varijante kombinacije predhodnih varijanti. Često se pomenutim varijantama organizacije programa dodeljuju nazivi:

- paralelan ulaz, paralelna obrada, paralelan izlaz,
- sekvencijalan ulaz, sekvencijalna obrada, sekvencijalan izlaz,
- sekvencijalan ulaz, sekvencijalna obrada, paralelan izlaz,
- paralelan ulaz, sekvencijalna obrada, sekvencijalan izlaz.



Sl. 4. I i II faza preklapanja



Sl. 5. II i III faza preklapanja

### 3.1.8. Modularnost programa

Za program se kaže da je modularan ako je organizovan u više nezavisnih delova (modula), s tim što se potrebna kombinacija između modula ostvaruje i organizuje preko operandske strukture. Da bi se ovo ostvarilo treba poznavati principe *koncentracije*, *razdvajanja* i *povezivanja* programa.

Koncentracija se sastoji u fizičkom grupisanju naredbi koje obavljaju određenu funkciju. Na primer, ako se u programu zahteva čitanje ulaznih podataka koji se odnose na određenu datoteku, operacije učitavanja mogu se koncentrisati u jednu sekciju programa, što se postiže, na primer, pisanjem potprograma koji se poziva kad god postoji potreba za čitanjem podataka.

Da bi se ostvarilo razdvajanje naredbi moraju se uočiti logičke celine i razdvojiti u posebne module. Treći aspekt modularnosti je povezivanje. Da bi program mogao pravilno da funkcioniše i koristi pojedine komponente (module), mora postojati povezanost između modula. Prvo, mora se obezbediti odgovarajući redosled izvršenja operacija, i drugo, mora se obezbediti komunikacija međurezultata između pojedinih modula programa. Pomenuta komunikacija najbolje se postiže korišćenjem modula tipa potprograma, kao i korišćenjem makroinstrukcija. Na primeru zatvorenih i otvorenih potprograma objasnićemo ulogu komunikacije. Pozivom zatvorenog potprograma obezbeđuje se prenos upravljanja na potprogram i njegov povratak posle izvršenja potprograma. Poziv takođe uključuje odgovarajuće dostavljanje ili identifikaciju podataka koji su potrebni potprogramu. Ovi podaci su ulaz programa i nazivaju se parametri. Oni mogu sadržati i identifikaciju ili adresu izlaznih podataka iz potprograma.



Otvoreni potprogrami zahtevaju sličnu vrstu povezivanja, ali je kod njih poziv manje formalan, a povezivanje je manje očigledno. Operativni sistem obično vodi računa o povezivanju zatvorenih potprograma, dok je kod otvorenih potprograma to redak slučaj. Razlog tome je što zatvoreni potprogrami mogu biti pozvani na izvršenje u bilo kom trenutku, dok se prenos upravljanja radi izvršenja otvorenog programa obično vrši jednom, i to na određenu tačku programa.

Korišćenjem makro programa, kada se jednom naredbom definiše određena funkcija, postiže se takođe modularnost programa. U praksi, makro program ima strukturu otvorenog potprograma, ali funkcionalno njegov poziv se ne razlikuje od poziva zatvorenog potprograma. Razlog ovome je što prevodilac obezbeđuje makro program u izvršnom jeziku onoliko puta koliko se on poziva u izvornom jeziku.

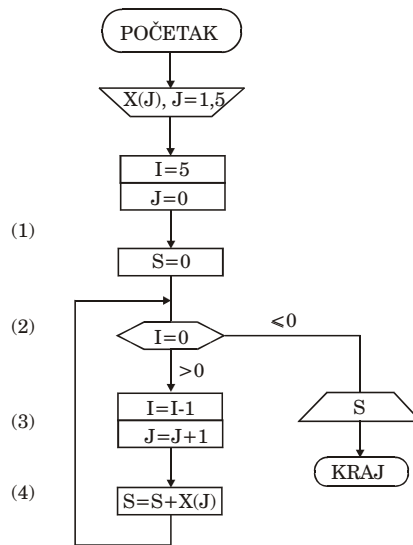
### 3.1.9. Cikličnost

Cikličnost se često koristi u procesu programiranja i time se postiže ušteda u vremenu programiranja i memorijskom prostoru, Dok je vreme izvršenja programa nešto duže. Cikličnost obuhvata sledeće korake:

1. Postavljanje početnih vrednosti ciklusa, i to:
  - postavljanje brojača, koji određuje broj ponavljanja ciklusa,
  - postavljanje početne vrednosti pokazivača (pointera) koji ukazuju na podatke. Često brojač može istovremeno imati i ulogu pokazivača,
  - postavljanje početnih vrednosti promenljivih koje se menjaju unutar ciklusa.
2. Testiranje određenog uslova koji predstavlja izlazni kriterijum iz ciklusa. Taj uslov može biti zadati broj ponavljanja ciklusa, zadata tačnost računara određene veličine ili maksimalno dozvoljeni broj iteracija kombinovan sa traženom tačnošću, itd.
3. Promenu (smanjenje ili povećanje) sadržaja brojača i postavljanje pokazivača na novu vrednost.
4. Obradu podataka u ciklusu.

Razradom algoritma kod cikličkih struktura treba izbeći kombinacije koje mogu dovesti do pogrešnog izvršenja programa. Npr. to se postiže izbegavanjem provere jednakosti dveju veličina u programu. Izlazni kriterijum u konstantnoj cikličnoj petlji treba napisati u obliku veće ili jednako od maksimalnog broja iteracija umesto jednako. Razlog tome je što brojač ciklusa može u principu preskočiti odbrojavanje maksimalnog broja iteracija, usled čega dolazi do formiranja beskonačne petlje.

Na sl. 6.naznačeni su pomenuti ciklički koraci u slučaju jedne algoritamske strukture kojom je predviđeno sabiranje jednog niza od pet elemenata. U tabeli 1.3. prikazano je stanje pojedinih veličina u raznim iteracionim ciklusima.



Sl. 6. Ciklička struktura

Tabela 1.3.

Iteraci oni ciklus	Brojač I	Pokazivač J	X (J)	Suma (S)	Objašnjenje
0	5	0	-	0	postavljanje poč. vrednosti
1	4	1	7	7	7=0+7
2	3	2	10	17	17=7+10
3	2	3	4	21	21=17+4
4	1	4	8	29	29=21+8
5	0	5	3	32	32=29+3

U praksi se obično najpre izvrši određena operacija, zatim testiranje, pa tek onda promena sadržaja brojača. Takav način je nešto brži, ali je manje pouzdan.

### 3.1.10. Indeksiranje

Indeksiranje se često koristi kod struktura kod kojih su adrese podataka u funkcionalnoj zavisnosti. Ta zavisnost je najčešće linearna. Znači, suština indeksiranja je u funkcionalnom generisanju adresa na osnovu kojih se pristupa podacima. Kod viših programskih jezika obezbeđeno je jednostavno indeksiranje pomoću indeksa nizova.

### 3.1.11. Testiranje programa (debugging)

Testiranje programa je faza u procesu programiranja kojom se otkrivaju i ispravljaju greške učinjene u toku programiranja. Greške su rezultat nepažnje, omaške, nepotpunosti ili nelogičnosti algoritma, neadekvatnosti ili nepreciznosti numeričke metode, netačnosti matematičkog modela procesa koji se rešava na računaru, a mogu biti i posledica neispravnosti računara. Na testiranju programa programer utroši trećinu, možda i polovinu vremena potrebnog za rešavanje određenog problema, te se radi toga mora posvetiti odgovarajuća pažnja postupcima za testiranje. Može se reći da je uspešno testiranje jedina garancija da je program upotrebljiv.

Proces testiranja počinje prevođenjem programa sa izvornog na izvršni jezik. Ukoliko ima sintaksnih grešaka, računar štampa dijagnostičke poruke kojima se greške svrstavaju u dve grupe: nedopustive i tolerantne.

Ukoliko se otkriju greške iz prve grupe prekida se rad računara, kako bi programer izvršio ispravke izvornog programa. Greške iz druge grupe, kao što je na primer preklapanje polja u memoriji itd., se tolerišu, jer ne moraju uticati na tačnost izvršavanja programa.

Sledeći korak je testiranje programa ili delova programa ili delova programa na računaru sa specijalno odabranim ulaznim podacima. Preporučljivo je najpre odvojeno testiranje modulskih delova programa, a zatim i testiranje programa kao celine. Otklanjanje grešaka iz modulskih delova programa ne garantuje pravilan rad celine, kod koje se mogu pojaviti greške zbog prenošenja podataka i uzajamne neusaglašenosti pojedinih delova programa.

Zatim sledi testiranje celokupnog programa na računaru, sa stvarnim ulaznim podacima.

Na kraju se pristupa testiranju i uhodavanju celokupnog sistema programa koji se može sastojati iz više već testiranih programa. Takav je slučaj kod informacionog sistema u jednom preduzeću ili kod skupa preduzeća ili ustanova, gde postoji povezanost između pojedinih programa koji koriste iste podatke (datoteke ili banke podataka).

Navešćemo neke postupke koji se koriste u procesu testiranja:

➤ -Izbor test podataka. Test podaci treba da poseduju izvesne osobine koje omogućavaju lako i pouzdano testiranje programa. Podaci treba da budu reprezentativni, ali jednostavni i malobrojni. Takođe je poželjno da test podaci obezbede unapred poznate izlazne rezultate ili rezultate koji se mogu lako ručno proveriti.

➤ -Testiranje cikličnih petlji. Iskustvo nas uči da će ciklična petlja raditi tačno, ako se proverom utvrdi da nekoliko početnih ciklusa radi ispravno.

➤ -Promena redosleda izvršenja programa privremeno umetnutim instrukcijama. Da bi ovaj postupak dao rezultate programer mora voditi računa o štampanju odgovarajućih podataka pre i posle prebacivanja upravljanja na izvršenje određenog dela programa.

➤ -Kontrola sadržaja memorije u toku izvršenja programa. Izvršenje programa može se pratiti korak po korak analiziranjem sadržaja memorije. Takvo testiranje programa je zametno, ali može biti od koristi u pojedinim slučajevima.

➤ -Privremeno ubacivanje štampanja specijalnih poruka i pozicionih indikatora u pogodno izabranim tačkama programa. Prilikom testiranja (uhodavanja) programa ove specijalne poruke su od velike pomoći, jer ukazuju na mesto grešaka, a često i na njihovo poreklo.

➤ -Proširenje programa dodatnim instrukcijama čija je uloga provera nekih međurezultata, ili konačnih rezultata koristeći unapred poznate rezultate funkcionalnih relacija između ovih međurezultata (rezultata). Na primer kod programa koja izračunava *sinus* i *kosinus* jednog ugla  $x$ , pogodna funkcionalna veza je  $\sin^2 x + \cos^2 x = 1$ .

U procesu testiranja od velikog su značaja pomoćni programi koji su deo računskog sistema. Najvažniji pomoćni programi u testiranju je program za izbacivanje sadržaja memorije (dump). Ono omogućava dobijanje sadržaja (sloga) iz unutrašnje memorije, koji se zatekao u posmatranom trenutku na određenom mestu za vreme izvršavanja programa. Sadržaj može ukazivati na programske ili operantske podatke. Da bi se program mogao poslužiti dobijenim sadržajem, on mora znati u kojoj tački se nalazilo upravljanje redosledom izvršenja programa. Do te informacije može se doći

preko indikatora na kontrolnoj konzoli, ukoliko operativni sistem računara ne daje odgovarajuću poruku. Kod tumačenja dobijenog sloga, programer se mora pomoći i informacijama koje se odnose na prevođenje, smeštanje u memoriju i povezivanje posmatranog sloga (edit map, load map, linkage map).

Sledeći pomoćni program za testiranje programa su dijagnostičke rutine. Sa njima se može pretraživati memorija u cilju nalaženja određenog podatka, ili se može verifikovati ispravnost funkcionisanja pojedinih delova računara.

Operativni sistem računara takođe ima veliki značaj kod testiranja programa. On javlja dijagnostičke poruke i daje izveštaj o događajima koji su se desili u toku izvršavanja programa, a koji su uzrok pojavi grešaka. Na primer kod pokušaja adresiranja polja koje je deo zaštićenog dela memorije, operativni sistem reaguje i javlja odgovarajuću dijagnostičku poruku.

### 3.1.12. Dokumentacija

Sređivanje dokumentacije programa je neophodno i može se slobodno reći da bez odgovarajuće dokumentacije mnogi programi postaju neupotrebljivi, čak i za same autore posle izvesnog vremena. Bez dobre dokumentacije ne može se zamisliti korišćenje, ispravka, dopuna i održavanje bilo kog složenijeg programa.

Finalna dokumentacija obično se formira od strane programera kada je određeni program već u eksploataciji. Dalja revizija dokumentacije prepuštena je onima koji eksploatišu i održavaju program.

Komponente programske dokumentacije su sledeći:

- sistemski dijagram,
- opis algoritma (rečima, matematičkim simbolima, formulama, tablicama odlučivanja),
- Grafička šema (obično se pojavljuje bar u dve forme: jedna je opšta, a druga detaljna, orjentisana ka jeziku na kome se programira),
- opis organizacije (forme i formata) ulaznih i izlaznih podataka,
- neophodna konfiguracija računara i identifikacija programa,
- uputstva operatoru za propuštanje programa na računaru (spisak potprograma koji se koriste iz biblioteke programa, sadržaj pojedinih poruka, pauza u programu zbog određenih intervencija nad ulaznim podacima, format izlaza i broj kopija, zauzeće memorije, vreme trajanja programa, itd.),
- kopija (listing) programa, ulaznih podataka i izlaznih rezultata,
- objašnjenje kritičnih tačaka u programu koje mogu uticati na korišćenje i dalju nadgradnju programa,
- uputstvo za korišćenje programa, sa priloženim ilustrativnim primerom, istorija programa, koja prati sve izmene od početka eksploatacije .

## 3.2. ALGORITMI I OPERACIJE

### 3.2.1. Algoritmi

Algoritam je plan (strategija) koji sadrži skup operacija, čije izvršavanje u odgovarajućem redosledu daje željenu promenu određene strukture podataka, Npr. aritmetika je algoritamski proces za manipulisanje brojevima. Za rešenje jednog aritmetičkog zadatka mora postojati plan sa aritmetičkim operacijama čijim se izvršenjem nad polaznim podacima, dolazi do željenih rezultata. U okviru algoritma često se koriste pojmovi *operator* i *operand*. Pod operatorom se podrazumeva bilo koja operacija koja se koristi u formiranju algoritma, dok se operandom naziva podatak na koji operator deluje.

### 3.2.2. Osobine algoritma

Glavne osobine algoritma su sledeće:

- svaki algoritam ima početak i kraj,
- algoritam koristi određeni skup operacija, o kojima će biti više reči u daljem tekstu,
- algoritmom se transformiše i kreira određena struktura podataka,
- algoritam je usmeren, determinisan i obavlja se u diskretnim vremenskim intervalima,
- algoritam mora obuhvatati sve alternative rešenja kako bi mogao prihvatiti razne vrednosti ulaznih podataka,
- Algoritam sadrži plan operacija, dok je upravljanje računarnom kod izvršavanja operacija zadatak programa, koji u interakciji sa hardverom (tehnička realizacija računara) i softverom (programska realizacija računara) realizuje plan operacija, odnosno algoritam.

Drugim rečima, za program se može reći da je realizacija algoritma na računaru.

### 3.2.3. Formiranje i zapis algoritma

Formiranje (pripremanje) algoritma je najkreativniji deo u procesu korišćenja računara. Osnovni izvori kod formiranja algoritma su ideje ljudi. Obično se kod rešavanja zadataka počinje sa analizom uloge računara u upravljačkoj petlji. Odatle se dolazi do skupa pravila koja omogućavaju formiranje algoritma.

Kod formiranja algoritma koriste se sledeći izvori:

*Analiza postojeće ručne obrade podataka jednog posla.* Treba voditi računa o ograničenosti i nepotpunosti takvog pristupa, jer se obrada podataka na računaru konceptijski razlikuje od ručne obrade.

*Poznavanje tehnološkog procesa.* Razumevanje raznih fizičkih pojava i procesa, načina njihovog odvijanja, rada mašina i ponašanje materijala, omogućavaju uspostavljanje relacija između veličina koje karakterišu određene pojave ili procese u matematičkoj ili logičkoj formi. Te relacije se koriste kod formiranja algoritma.

*Logičke tablice i tablice odlučivanja. Numerička analiza.* Sa stanovišta formiranja algoritma ovo je jedna od najvažnijih grana matematike. Njena specijalnost je upravo u razradi iterativno- rekurzivnih mehanizama koji omogućavaju algoritmizaciju složenih procesa i metoda.

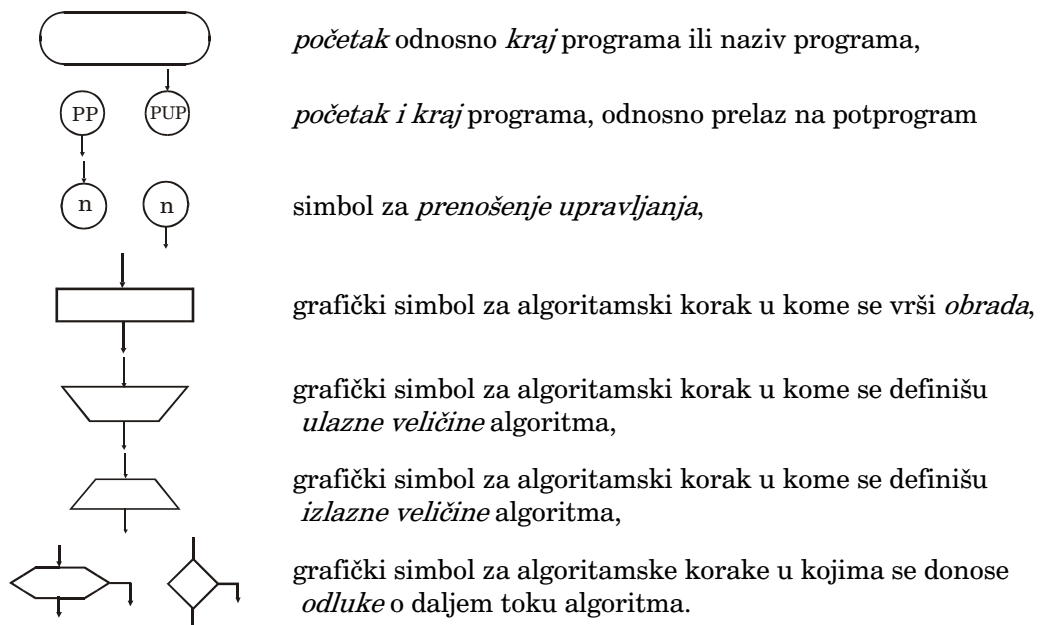
### 3.2.4. Metode linearnog i nelinearnog programiranja.

Najpogodniji način zapisa algoritma je uz pomoć grafičke šeme koja svojim simbolima ukazuje na prirodu pojedinih algoritamskih koraka, kao što su npr:

- početak i kraj algoritma
- unošenje (čitanje) ulaznih podataka i izdavanje rezultata
- operacije koje se izvršavaju u algoritmu i
- veza između pojedinih algoritamskih koraka.

Algoritam može biti realizovan pomoću jedinstvene grafičke šeme (glavni program), a može sadržati još jednu ili više dodatnih nezavisnih grafičkih šema (potprogrami). U slučaju kada ima potprograma treba definisati i njihove ulazno- izlazne veličine.

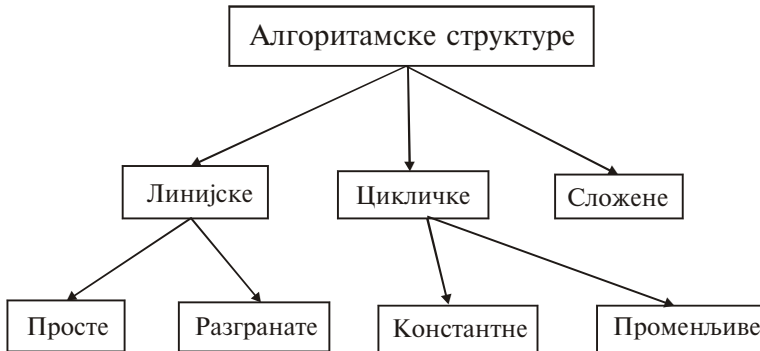
Pri grafičkom predstavljanju algoritma mogu se koristiti grafički simboli:



Korisno je poznavati hardverske i softverske mogućnosti računara, koji se koristi za rešavanje određenog zadatka, pre nego što se pristupi formiranju algoritma. Time se znatno olakšava posao oko formiranja i zapisa algoritma u smislu nivoa detaljizacije pojedinih algoritamskih koraka i njegove programske realizacije.

### 3.2.5. Algoritamske strukture

Sa stanovišta upravljanja kojima se ostvaruje (izvršava) algoritam, moguće je realizovati razne tipove algoritamskih struktura (sl. 7.).



Sl. 7. Algoritamske strukture

Kod linijskih algoritamskih struktura moguće je samo jednom izvršavanje nekog algoritamskog koraka, tj. nakon izvršavanja jednog algoritamskog koraka upravljanje izvršenjem algoritamskih koraka može se preneti samo na algoritamski korak koji nije još nijedanput izvršen. Kod prostih linijskih struktura redosled izvršavanja algoritamskih koraka je unapred definisan i ne dolazi do prenošenja upravljanja na bazi

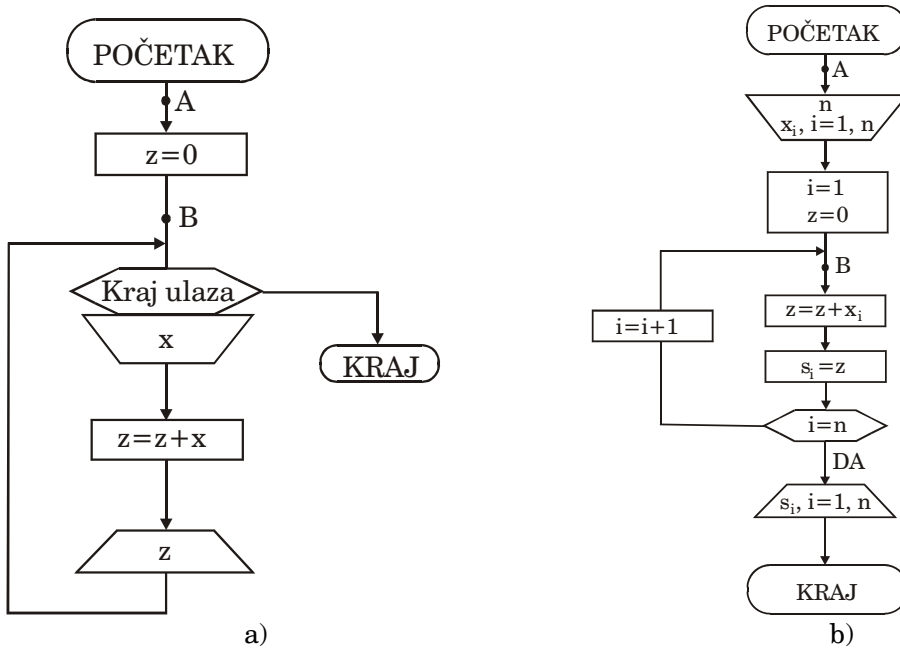
ispitivanja nekog uslova, kao što je to slučaj kod razgranatih linijskih struktura.

Karakteristika cikličkih algoritamskih struktura je u višestrukome izvršavanju jednog ili više algoritamskih koraka, pri čemu može doći do promene zakona obrade kod pojedinih algoritamskih koraka (promenljiva ciklička struktura), ili ne (konstantna ciklička struktura). Promena zakona obrade najčešće se dešava nad operacijama koje povezuju pojedine promenljive (modifikacija algoritma), ili nad samim promenljivima koje učestvuju u obradi. Sложене algoritamske strukture su kombinacija pomenutih struktura i često sadrže veći broj povezanih cikličkih struktura. Sledeći primer ilustruje sve tipove algoritamskih struktura.

**Primer.** Sastaviti algoritam za dobijanje svih međusuma od elemenata niza  $x_i$ ,  $i = 1, \dots, n$

$$S_j = \sum_{i=1}^j x_i, \quad j = 1, \dots, n$$

1. kada  $n$  nije poznato
2. kada je  $n$  poznato



Sl. 8. algoritamske strukture

Oba algoritma na sl. 8. poseduju prostu linijsku strukturu (od tačke A do V), razgranatu linijsku strukturu (postoji algoritamski korak za ispitivanje uslova) i cikličku strukturu, s tim što je kod prvog algoritma ciklička struktura konstantna, a kod drugog promenljiva (algoritamski korak  $s_i = z$ ).



## 4. OSNOVE C JEZIKA

Programski jezik C se ubraja u standardne programske jezike višeg nivoa za profesionalni razvoj softvera. Godina nastanka C jezika 1972. poklapa se sa vremenskim određenjem pojma " strukturno programiranje " na kome se ovaj jezik i bazira. Ipak, za izlaganje istorije C jezika neophodno je pomenuti operativni sistem UNIX, jer je taj operativni sistem i programi na njemu počeli da rade kada su bili napisani u programskom jeziku C.

U Bell laboratorijama se razvijao UNIX, operativni sistem, koji je trebao biti prenosiv, opremljen korisnim razvojnim alatima, kratkim komandama i relativno otvorenim okruženjem. Ovakav operativni sistem je napisan za mali računar **DEC PDP-7**, u potpunosti u njegovom asembleru.

UNIX je već bio u funkcionalnoj upotrebi, kada je Ken Tompson ( *Ken Thompson* ) implementirao na PDP-7 kompajler za novi programski jezik, nazvan *jezik B*. Razvoj ovog jezika je bio pod značajnim uticajem **BCPL-a** Martina Ričardsa ( *Martin Richards* ), jezika za pisanje sistemskog softvera, a jezici BCPL i B su imali snažan uticaj na razvoj jezika C, pogotovo što je B jezik bio zasnovan na radu sa rečima.

Vremenom je za laboratoriju nabavljen novi računar, DEC PDP-11 (sa 24 Kilobajta memorije, od koje je operativni sistem koristio 16 Kilobajta, i tvrdim diskom od 512 Kilobajta), na koji su preneti UNIX i B jezik. Razmatrajući ideju da se UNIX prepiše na B jezik, došlo se do zaključka da to nije najbolje rešenje, pošto je jezik B bio interpreter i zbog toga spor, i što je još važnije, PDP-11 je bio orijentisan prema radu sa bitovima za razliku od B jezika, koji je bio orijentisan prema radu sa rečima. Zato je počeo razvoj na nasledniku B jezika, pogodno nazvanom C jezik. Ono što se dalje odvijalo je zapanjujuće jednostavno. Godine 1973. UNIX je proradio pod C jezikom, zahvaljujui Denisu Ričiju ( *Dennis Ritchie* ). Danas, zbog svojih izuzetnih osobina C jezik je jedan od najvažnijih i najpopularnijih programskih jezika, koji se veoma široko primenjuje, o čemu govori sledeća slika.

Mnoge softverske kuće koriste C jezik za pisanje tekst - procesora, programa za tabelarne proračune, kompajlera, ljske ekspertnih sistema, itd. Njima je poznato da su C programi kompaktni i efikasni, i što je najvažnije, lako prilagodljivi novim hardverskim i softverskim platformama. Tako je i većina programa iz operativnog sistema Windows, pisana baš u ovom programskom jeziku.

## Karakteristike programskog jezika C

Svi jezici, uključujući i prirodne, imaju svoj karakterističan izgled, stil i moguću prihvaćenu primenu. Mnogi programeri, prilikom prvog susreta sa C jezikom, ne mogu da se priviknu na mnoštvo operatora neobičnog izgleda i ekstenzivnog korišćenja pokazivača, obiljem elemenata strukturnog programiranja sa jasno izraženim kontrolnim upravljačkim strukturama i sledećim osnovnim karakteristikama:

♦ **C je jezik sa malim skupom komandi** koji omogućuje da se programska rešenja izraze sa samo nekoliko ključnih reči. Za razliku od nekih drugih jezika npr. Ada, Cobol, itd., kod kojih većina programera može da se nada da će naučiti samo delove programskog jezika kao celine, programski jezik C se shvata u potpunosti za veoma kratko vreme. ANSI C koristi samo 32 ključne reči kao 32 standardna identifikatora, a

♦ zahvaljujući ANSI standardu, praktično su rezervisana i imena funkcija iz standardne biblioteke funkcija i makroa. Takav pristup znači de je skup opšte prihvaćenih imena bibliotečkih funkcija skoro deo samog jezika, to umnogome olakšava rad na stvaranju softverskih rešenja na različitim platformama.

♦ **Struktuiranost programskih rešenja na visokom nivou** čine da programski jezik C poseduje mnoge osobine jezika visokog nivoa kao što su Ada, Modula 2 i Pascal. Ove osobine pre svega omogućuju da se pri razvoju programskih rešenja puna koncentracija posveti logičkom toku problema bez razmišljanja o pojedinim specifičnim mašinskim instrukcijama. Programski jezik C sadrži sve kontrolne strukture i podržava metodološke pristupe što se i očekuje od savremenih programskih jezika:

- za kontrolu toka, koristi tri vrste petlji za rukovanje ponavljajućim operacijama i tri strukture za izbor alternativnog puta izvršavanja programa;

- raznoliki tipovi podataka (osnovni i izvedeni) omogućuju da se predstavi širok opseg informacija;

- upotreba top-down planiranja, strukturnog programiranja i modularnog dizajna je sastavni deo kreiranja programa na C jeziku, usled čega je program u većoj meri pouzdan i razumljiv.

♦ **C je široko prenosiv** između različitih okruženja, pa čak i između samostalnih mikroprocesora i najbržih superračunara, jer skup tipova podataka jezika C odražava arhitekturu koja je u osnovi mnogih modernih računarskih sistema. C programi napisani u jednom sistemu se mogu sa vrlo malo ili nimalo modifikacija izvršavati i na drugim sistemima. Ako su modifikacije i potrebne, onda se one odnose na datoteke zaglavlja i ulazno/izlazne operacije koje program koristi. Naravno, prenosivost je moguća i kod drugih programskih jezika, ali lidersko mesto ipak pripada C jeziku. Sa ovim će se sigurno složiti svi oni koji su konvertovali (ili barem pokušali) programe na programskim jezicima BASIC ili FORTRAN.

♦ **C omogućava modularnost rešenja** što predstavlja jedno od osnovnih načela strukturnog programiranja: program treba podeliti na skup potprograma, od kojih svaki obavlja jedan, pažljivo definisan zadatak. U programskom jeziku C, potprogrami imaju jedan od sledeća dva oblika: funkcije i makroi.

-*Funkcije* predstavljaju osnovnu jedinicu za programsko rešenje u C jeziku i blisko su povezane sa upotrebom pokazivača i nizova. Osobina funkcija je da izvršavaju tačno jedan određen zadatak. Funkcije, kao osnovna podrška modularnosti u programiranju, mogu vraćati jednu ili više vrednosti ili pak ne moraju imati definisanu vraćenu vrednost. Funkcije jezika C imaju dve veoma važne osobine: prva, da imamo

kontrolu pristupa do podataka i onemogućavanje neovlašćenog menjanja podataka, i druga, da se može sakriti način realizacije konkretnih rešenja.

*Makrodefinicije* predstavljatu definicije simbola, imaju jedan ili više argumenata i tip argumenta makroa može biti proizvoljan. Najčešće se koriste umesto jednostavnih funkcija.

♦ **C omogućava modularno projektovanje rešenja:** U osnovnom konceptu kreiranja izvršnog koda u programskom jeziku C su odvojeni postupci prevođenja i povezivanja. Ovakav pristup omogućuje programerima da prevode samo delove programa koji su

izmenjeni u toku razvoja ili testiranja. Ovako modularno projektovanje rešenja je od veoma velikog značaja pri realizaciji kompleksnih softverskih projekata koji se obično sastoje od velikog broja programa i modula.

♦ **C se koristi za širok opseg programerskih zadataka** iako je u početku korišćen za sistemsko programiranje (pisanje operativnih sistema i kompajlera ) pogodan je i za aplikativno programiranje, pošto poseduje operatore niskog nivoa za rad sa podacima na nivou bita. Nema potrebe da se koristi mašinski zavisno programiranje na assembleru.

♦ **C je snažan i fleksibilan jezik** zahvaljujući obimnom skupu operatora (ukupno 40). Jezik C ima sve očekivane operatore za aritmetičke operacije, poređenja, logičke operacije i za manipulaciju podacima na razne načine. Operativni sistem UNIX, koji se smatra takođe snažnim i fleksibilnim, napisan je u C jeziku. Na UNIX-u se izvršavaju mnogi programi, počev od tekst editora do kompajlera i interpretera za druge jezike: kao to su FORTRAN, APL, Pascal, LIST, Logo i BASIC. Kada se koristi npr. FORTRAN na UNIX-u, na kraju ipak C program generiše izvršni FORTRAN program.

♦ **C podržava rekurziju** što omogućava elegantna rešenja mnogih problema kod kojih je rešavanje rekurzijom prirodnije nego upotreba iteracija, naročito u različitim oblastima veštačke inteligencije.

♦ **C podržava pokazivačke promenljive:** Operativni sistemi novije generacije su prema razvojnim softverskim alatima postavili zahtev za adresiranjem određenih delova memorije. Programski jezik C je svakako jedan od jezika koji u punoj meri zadovoljava ovakav zahtev, jer su pokazivači, pokazivačke promenljive i aritmetika sa pokazivačima utkani u samu strukturu programskog jezika C.

♦ **C obezbeuje dobro definisanu programersku spregu** zahvaljujući standardizovanoj i raznovrsnoj biblioteci često korišćenih rutina za vreme izvršavanja (run-time-library).

Kompajleri koji u ovom trenutku stoje na raspolaganju za prevođenje C izvornih kodova su:

cc u UNIX - u ili gcc u LINUX operativnom sistemu  
Borland C ili Turbo C u DOS operativnom sistemu  
Visual C++ ili Borland C u WINDOWS operativnom sistemu  
C++ Builder u WINDOWS operativnom sistemu

U sledećem poglavlju predstavljani su osnovni sastavni elementi C programa. To su: skup karaktera, ključne ili rezervisane reči, identifikatori, izrazi, iskazi, komentari, operatori, funkcije i konstante. Navedeni su jednostavni i kompletni programski primeri, koji sadrže definisane elemente. Opisane su i dve funkcije, printf i scanf, iz

sistemske biblioteke, koje se intenzivno koriste u narednim programskim primerima kao osnovne funkcije ulaza - izlaza.

#### 4.1 Skup karaktera C jezika

Dozvoljeni karakteri za formiranje osnovnih elemenata C programa su alfanumerički karakteri (velika i mala slova i cifre), specijalni karakteri i nevidljivi (ili neštampajući) karakteri (blank, nova linija i tab). Skup svih karaktera je prikazan u tabeli 3.1. U većini programskih jezika (FORTRAN, COBOL, BASIC, itd.) velika i mala slova se identično tretiraju (SIMBOL i simbol su identični simboli). Međutim, u C jeziku velika i mala slova tretiraju se različito (int i INT su različiti simboli).

#### 4.2 Ključne reči

U tabeli 3.2 prikazane su ključne ili rezervisane reči. One imaju unapred definisano značenje. Drugim rečima, ključne reči se ne mogu koristiti u drugom kontekstu. Osobnost C jezika je korišćenje isključivo malih slova za formiranje ključnih reči.

<b>Mala slova</b>	<b>a b c ...z</b>
<b>Velika slova</b>	<b>A B C ...Z</b>
<b>Cifre</b>	<b>0 1 2 3 4 5 6 7 8 9</b>
<b>Specijalni karakteri</b>	<b>+ = - _ ( ) * &amp; % \$ # ! &lt; &gt;   ! . " ' / ? } { ~ \ [ ]</b>
<b>Nevidljivi karakteri blank (prazno mesto),</b>	
<b>nova linija, tab ( prazna mesta )</b>	

Tabela 3.1 Skup karaktera C jezika

Ključna reč `entry` se trenutno ne koristi u C prevodiocima, ali je rezervisana za buduću upotrebu. Slično tome, `ada`, `pascal`, `enum`, `void` i `fortran` se ne koriste u svim prevodiocima. U poređenju sa drugim jezicima C ima mali broj ključnih reči (Ada, na primer, ima 62), ali time njegove mogućnosti nisu umanjene.

<code>ada</code>	<code>double</code>	<code>goto</code>	<code>static</code>	<code>asm</code>	<code>else</code>	<code>if</code>	<code>struct</code>
<code>auto</code>	<code>entry</code>	<code>int</code>	<code>switch</code>	<code>break</code>	<code>enum</code>	<code>long</code>	<code>typedef</code>
<code>case</code>	<code>extern</code>	<code>pascal</code>	<code>union</code>	<code>char</code>	<code>float</code>	<code>register</code>	<code>unsigned</code>
<code>continue</code>	<code>for</code>	<code>do</code>	<code>return</code>	<code>void</code>	<code>default</code>	<code>fortran</code>	<code>short</code>
						<code>while</code>	

### 4.3 Osnovni elementi C jezika

C program se sastoji od osnovnih gradivnih elemenata: identifikatori, iskazi, izrazi, operatori, separatori, itd. Napišimo jednostavan C program, koji na terminalu ili CRT ekranu ispisuje poruku "Zdravo, Kako ste?" I uočimo osnovne gradivne elemente (Program 1). Razumevanje ovih pojmova olakšaće čitanje narednih poglavlja.

```
main()
{printf("Zdravo, Kako ste? \n");}
```

main i printf su identifikatori. Identifikator je sekvenca velikih ili malih slova, cifara i karaktera "\_" ( donja crta ). Slovo ili donja crta moraju biti prvi karakter identifikatora. Na primer, ispravni identifikatori su:

x	z_256	_suma	_iznos
imel	prezime	tabela_1	slika_256
a pogrešni:			
max + 1	2_main	\$krug	235
ime 1	int	main	-zapremina

max + 1 nije identifikator, već izraz, jer sadrži karakter +, 2\_main počinje cifrom, a \$krug karakterom '\$', 235 je broj, a ne identifikator, ime 1 sadrži blank, int je ključna reč, main je rezervisan za sistemsku funkciju, a -zapremina je izraz.

Identifikatori se koriste za dodeljivanje imena objektima u programu (funkcije, promenljive, itd.). Dobra programerska praksa je izabrati mnemoničke identifikatore objekata (npr., identifikator površina ukazuje na površinu nekog kruga), u cilju povećanja čitljivosti i dokumentovanosti programa. Broj karaktera u identifikatoru nije ograničen, ali je u mnogim sistemima broj značajnih karaktera ograničen, npr. na 8. To znači, da su identifikatori temperatura1 i temperatura2 identični, jer je identično prvih 8 karaktera. Na primer, C prevodilac na sistemu DEC VAX-11 VMS\* ograničava broj značajnih cifara na 31, a na sistemu UNIX System V Release 2 identifikatori mogu biti proizvoljne dužine. Identifikatori main i printf su imena sistemskih funkcija. main informiše sistem o tome gde treba započeti izvršavanje programa, a "()" označava da main nema argumenata. U C programu obavezna je jedna i samo jedna funkcija main, jer program ne može započeti izvršavanje sa dva ili više mesta istovremeno.

*printf* je funkcija za štampanje poruke "Zdravo. Kako ste?". "Zdravo. Kako ste?\n" je niz karaktera i u slučaju programa 3.1 predstavlja argument funkcije printf. Karakteri '\ ' (obrnuta kosa crta) i "n" čine jedan karakter koji se zove nova linija i znači da C program sledeće štampanje obavlja na novoj liniji, odmah ispod tekuće.

*printf*("Zdravo. Kako ste?\n"); je iskaz C jezika i predstavlja elementarnu obradu, koju program treba da obavi. Naš program se sastoji od samo jednog iskaza. Svaki iskaz se mora završiti karakterom ';' koji ima ulogu terminatora iskaza.

Karaktereri '{' i '}' (vitičaste zagrade) objedinjuju više pojedinačnih iskaza u jednu programsku celinu, odnosno u složeni iskaz, analogno komandama BEGIN i END u PASCAL jeziku. Pojedinačni iskazi se ne moraju objedinjavati sa '{' i '}' jer već predstavljaju programsku celinu, tj. elementarnu obradu. U našem primeru karakteri '{'

'}', informišu šta program treba da obavi i obavezne su, bez obzira što je u pitanju samo jedan iskaz.

Pogledajmo program 4.2, koji uvodi nove sastavne elemente C jezika.

### Program 4.2

*main()*

```
{
int razlika;
razlika = 100 -90;
printf("Razlika 100 -90= %d\\ n", razlika);
}
```

**Program 4.2 Izlaz**

**Razlika 100 -90= 10**

U programu 4.2 iskaz:

```
int razlika;
```

je deklaracija, kojom se promenljiva razlika deklarise kao tip int, tj., kao celobrojna promenljiva. Celobrojnoj promenljivoj se mogu dodeliti samo celi brojevi, odnosno, brojevi bez decimalnog zareza. Primeri celih brojeva su:

```
0      1      3      -1
766    -52    128    999
```

dok to ne važi za brojeve

```
123.   436 36.
```

Pored celih brojeva, u C jeziku se koriste i brojevi sa pomičnim zarezom ili realni brojevi. To su brojevi, koji sadrže decimalni deo. Primeri brojeva sa pomičnim zarezom su :

```
2.3    0.0    314.  1.
```

```
dok to nisu    0      1      -3      99,9
```

Karakter '-' je aritmetički operator oduzimanja, a karakter '=' operator dodeljivanja vrednosti. 100-90 je izraz, u ovom slučaju konstantan, jer u toku izvršavanja programa ne menja vrednost. Operandi 100 i 90 su konstante. Iskazom

```
razlika = 100 -90;
```

celobrojnoj promenljivoj razlika dodeljuje se vrednost izraza 100-90. Operator dodeljivanja vrednosti je specifično definisan u C jeziku i u narednim poglavljima detaljno opisan. Karakteri '{' i '}', objedinjavaju jednu deklaraciju i dva iskaza formirajući složeni iskaz.

Printf se koristi u složenijem obliku, sa dva argumenta "Razlika 100-90 = %d\n" i razlika. Prvi argument funkcije sadrži %d. Funkcija printf analizirajući niz "Razlika 100- 90 = %d\n", pronalazi karakter "%", pri čemu karakter 'd' prihvata kao konverzioni karakter, odnosno, na tom mestu štampa vrednost odgovarajućeg argumenta iz liste argumenata kao celi broj. printf ima promenljiv broj argumenata. Prvi argument je obavezno niz, koji određuje format, odnosno način na koji se prikazuju argumenti iz liste argumenata. Na ovaj način štampaju se vrednosti programskih promenljivih.

Program 4.3 uvodi pojmove višestruke deklaracije i komentar. Pomoću iskaza

**int razlika, opl, op2;**

deklarišu se tri celobrojne promenljive, razlika, op1 i op2. Deklaracije promenljivih mogu se napisati odvojeno kao

```
int razlika;
int opl ;
int op2;
ili kao
int razlika,
op1,
op2;
```

```
/*Program 4.3 */
```

```
/* Program za izracunavanje razlike dva broja*/
```

```
main()
```

```
{
```

```
int razlika, op1, op2;      /* deklaracija */
```

```
op1 = 100;                 /* 1. operand */
```

```
op2 = 90;                  /* 2. operand */
```

```
razlika = op1 -op2;       /* razlika 1. i 2. operanda */
```

```
printf("Razlika 100 -90 = %d\n", razlika); /* stampanje razlika */}
```

**Program 4.3 Izlaz**

**Razlika 100 -90 = 10**

Tekst napisan između /\* i \*/ je komentar (/\* i \*/ se moraju pisati sastavljeno) Prevodilac ignoriše komentar. Komentar se može pisati u jednoj ili više linija, kao :

```
/* komentar */
/** komentar ***/
/*****
```

**Veoma**

**dugačak**

**komentar**

```
*****/
```

Komentar nije deo izvršnog programa i koristi se za dokumentovanje programa. Dokumentovanje programa ima za cilj da autoru programa, korisniku ili serviseru jasno opiše kako program radi, njegovu namenu i ograničenja. Komentare treba pisati istovremeno sa pisanjem programa. Ako program u početku radi bez vidljivih grešaka, komentari se izostavljaju ili navode u skraćenom i stoga nedovoljnom obliku, što otežava, a ponekad sprečava, naknadno proširivanje programa ili popravku kasnije uočenih grešaka. Pored bolje dokumentovanosti, komentari mogu poslužiti kao korisna provera programske logike i strukture, doprinoseći jasnoći i korektnosti programa.

C jezik nije pozicioni kao FORTRAN. Iskazi se mogu pisati bilo gde u liniji ili u više linija. Program 4.4 je potpuno korektan C program.

```
/*Program 4.4*/  
main(  
    ) {int v1,  
        v2 ;  
            v1 = 100;v2 =  
                90;  
printf("Ovaj program je korektan\n"  
);  
printf("v1 = %d v2 = %d\n",  
    v1,  
    v2)  
; }
```

#### Program 4.4 Izlaz

Ovaj program je korektan  
v1 = 100 v2 = 90

Očigledno da programe nije poželjno pisati u sličnom stilu. Program 4.5 je isti kao i program 4.4, ali je napisan čitljivije i dokumentovanije.

```
Program 4.5  
main( )  
{  
    int v1,v2 ;  
    v1 = 100;  
    v2 = 90;  
    printf("Ovaj program je korektan\n" );  
    printf("v1 = %d v2 = %d\n",v1, v2) ;  
}
```



**Program 4.5 Izlaz**

Ovaj program je korektan

v1 = 100 v2 = 90

**4.4 Elementarni ulaz i izlaz i predprocesori**

U prethodnim primerima korišćena je sistemska funkcija `printf` za ostvarivanje izlazne aktivnosti. Funkcija `scanf` realizuje ulaznu aktivnost. U narednom tekstu, ove funkcije su detaljnije opisane.

`printf` i `scanf` obavljaju formatizovanu izlaznu i ulaznu aktivnost, odnosno, po određenom formatu štampaju i učitavaju specificirane vrednosti. Argumenti obe funkcije su podeljeni u dva dela: kontrolni ili konverzioni niz i lista argumenata. Konverzioni niz određuje format za štampanje ili učitavanje liste argumenata. Specifikacija formata počinje karakterom '%', a završava konverzionim ili kontrolnim karakterom. U prethodnim primerima imali smo %d, gde karakter 'd' predstavlja konverzioni karakter, koji određuje da se argumenti iz liste argumenata štampaju kao dekadni celi brojevi. U tabeli 3.3 prikazan je podskup najčešće korišćenih konverzionih karaktera funkcije `printf` (kompletan skup konverzionih karaktera opisan je u poglavlju, koje opisuje ulaz i izlaz C programa).

Konverzioni karakter	Konverzija izlaznog niza
c	karakter
d	ceo broj
f	realan broj
s	niz karaktera

Tabela 3.3 Konverzioni karakteri funkcije `printf`

Mesto na kome se štampa argument naziva se polje, a broj karaktera u polju je širina polja. U programima 4.6 i 4.7 koriste se konverzioni karakteri c, d, s. Širina polja je specificirana u konverzionom nizu, između karaktera '%' i konverzionog karaktera. Program 4.6 štampa karaktere 'A', 'B' i 'C' u polju širine 3, 5 i 7 karaktera, respektivno. Program 4.7 štampa niz karaktera, karakter i celobrojnu vrednost, demonstrirajući upotrebu konverzionih karaktera s, c i d.

*/\*Program 4.6\*/*

*main()*

{

*printf("Karakter:\n%3c\n%5c\n%7c\n", 'A', 'B', 'C');*

}

**Program 4.6 izlaz****Karakter:**

A  
B  
C

```

/*Program 4.7*/
main()
{
int var;
var = 65;
printf("%s\nkarakter=%3c\nceo broj=%5d\n",
"Upotreba konverzionih karaktera s, c, d", var,var);
}

```

**Program 4.7 Izlaz****Upotreba konverzionih karaktera s, c, d**

karakter= A  
ceo broj= 65

Scanf je slična funkcija printf. Razlika je u listi argumenata. Argumenti funkcije printf prenose se vrednošću, dok se kod scanf prenose svojom adresom, jer se učitana vrednost mora vratiti u pozivajući program. To se postiže navođenjem karaktera '&' (operator indirekcije) ispred argumenata u listi argumenata, koji je opisan u poglavlju o ukazateljima. Primena operatora & je obavezna, u protivnom, scanf ne radi korektno. U tabeli 3.4 prikazani su najčešće korišćeni konverzioni karakteri funkcije scanf.

Program 4.8 učitava i štampa dekadnu celobrojnu vrednost. Iskazom scanf("%d", &x); učitava se celobrojna vrednost i dodeljuje promenljivoj x.

```

/*Program 4.8*/
main()
{
int x;
printf("Upotreba funkcije scanf:\nUkucajte ceo broj\n");
scanf("%d",&x);
printf("Ukucali ste\n%d\n", x); }

```

**Program 4.8 Izlaz****Upotreba funkcije scanf:**

Ukucajte ceo broj

23

Ukucali ste

23

#### 4.4.1. Preprocesori

Ovo poglavlje opisuje jednu mogućnost C jezika, koja ne postoji u mnogim drugim programskim jezicima visokog nivoa. To je C preprocesor. C preprocesor omogućava jednostavniji razvoj i modifikaciju i veću prenosivost programa. Preprocesor je sastavni deo C prevodioca koji analizira tekst programa pre analize iskaza od strane C prevodioca. Procesorski iskazi se identifikuju prisustvom karaktera numerik '#', koji mora biti vidljiv karakter u liniji. Sintaksa ovih iskaza je različita od sintakse iskaza C jezika. Diskusiju o iskazima C preprocesora počinjemo iskazom #define.

##### 4.4.1.1. Simbolička imena i makroi- #define

Najjednostavnija upotreba iskaza #define je dodeljivanje simboličkih imena konstantama. Preprocesorski iskaz:

```
#define PI 3.141592
```

definiše simboličko ime (ili drugačije, simbol) PI za konstantu 3.141592. Simbol PI može biti korišćen u daljem tekstu programa, gde god je potrebna konstanta 3.141592. U tom slučaju, C procesor automatski zamenjuje simbol PI konstantom 3.141592. Na primer, ako su promenljive obim i r tipa float, tada se obim kruga može izračunati iskazom:

```
obim=2*PI*r ;
```

jer se PI zamenjuje sa 3.141592. Simbol PI ne predstavlja promenljivu, pa ne može biti levi operand operatora za dodeljivanje vrednosti. Kad god se simboličko ime, definisano iskazom #define, pojavi u tekstu programa. C preprocesor automatski zamenjuje sa onim što se nalazi desno od imena u iskazu #define.

Sintaksa iskaza #define se razlikuje od sintakse iskaza C jezika. U gornjem iskazu #define, između PI i 3.141592 ne postoji karakter '=', niti na kraju iskaza separator ';' definisanog simbola. Na primer, simbol PI mora biti definisan pre izračunavanja obima kruga u predhodnom iskazu. Ovako definisani simboli nemaju osobinu lokalnosti. Jednom definisan simbol, u funkciji ili van funkcije ili bilo gde drugde, može biti dalje korišćen bilo gde u programu, odnosno, izvornoj datoteci. Radi preglednosti, poželjno je grupisati iskaze #define na početku programa.

Korišćenje iskaza #define ilustrovano je u programu 4.9. u kojem su definisane dve funkcije: (1) o\_krug za izračunavanje obima kruga i (2) p\_krug za izračunavanje površine kruga. Kako se u obe funkcije koristi konstanta 3.141592, iskazom #define konstanti 3.141592 dodeljuje se simboličko ime PI.

#### Program 4.9.

```
/*Program za izracunavanje obima i površine kruga*/
#define PI 3.141592
float o_krug(r) /*F ja za izracunavanje obima kruga*/
    float r;
    {
        return(2.0*r*PI);
    }
```

```

float p_kruga(r)      /*F ja za izracunavanje povrsine kruga*/
    float r;
    {
    return(r*r*PI);
    }

main()
{
    static float r[4]={2.0, 3.0,4.0,5.0};
    int i;

    for (i=0; i<4; ++i)
        printf("Precnik=%f Obim=%f Povrsina=%f\n",
                2.0*r[i], o_krug(r[i]));
}

```

#### Program 4.9. Izlaz

Precnik=4.000000	Obim=12.566368	Povrsina=12.566368
Precnik=6.000000	Obim=18.849552	Povrsina=28.274328
Precnik=8.000000	Obim=25.132736	Povrsina=50.265472
Precnik=10.000000	Obim=31.415920	Povrsina=78.539800

#### 4.4.1.2. Uključivanje datoteke- #include

Operativni sistem UNIX ima veliki broj unapred definisanih makroa. Simbolička imena tih makroa su napisana malim slovima, tako da se ne razlikuju od regularnih funkcija. Na primer, getchar je makro, koji učitava karakter sa standardnog ulaza. Posle dužeg programiranja u C jeziku, većina programera definiše svoje sopstvene makroe. Sistemski i sopstveni makroi mogu biti u posebnim datotekama, takozvani include datotekama. C preprocesor omogućava uključivanje takvih datoteka u izvorni program. Na taj način, izbegnuta je potreba definisanja makroa kad god su potrebni. Sopstveni makroi se definišu u posebnoj datoteci i korišćenjem iskaza #include, uključuju izvorni program. Slično, sistemski makroi su definisani u veći broj posebnih sistemskih datoteka. Ove datoteke se, takođe, uključuju iskazom #include.

Pretpostavićemo da se definicije sledećih makroa nalaze u datoteci macro.h\*.

```

#define TO_UPPER(c) (IS_LOWER(c) ? (c) -'a'+'A' : (c))
#define IS_LOWER(c) (((c)>='a') && ((c)<='z'))
#define MIN(x, y) ((x)<(y)) ? (x) : (y)
#define MAX(x, y) (((x)>(y)) ? (x) : (y))
#define SQUARE (x) * (y)

```

U programu, koji koristi neke od definisanih makroa iz datoteke macro.h, mora se nalaziti preprocesorski iskaz:

```
#include "macro.h"
```

Ovaj iskaz se mora pojaviti pre korišćenja bilo kog makroa iz datoteke macro.h. Procesor traži specificiranu datoteku u sistemu i kopira njen sadržaj na istom mestu na

\* U UNIX-u datoteke sa definicijama sistemskih i sopstvenih makroa najčešće imaju ekstenziju "h".

kojem se pojavljuje iskaz `#include`. Na ovaj način, svi iskazi iz datoteke `macro.h` tretiraju se kao da su napisani direktno u izvorni program, s tom razlikom što se kopiranje odvija automatski. Dvostruke navodnice u iskazu `#include` nalažu sistemu da datoteku `macro.h` traži u istom direktorijumu u kojem se nalazi izvorni program. Ako se datoteka ne pronađe, tada preprocesor pretražuje sve direktorijume, koji su specificirani od strane administratora sistema. Međutim, iskaz

```
#include <stdio.h>
```

ima isti efekat kao i predhodni iskaz `#include`, s tom razlikom što se ne pretražuje direktorijum na kojem se nalazi izvorni program. Zgrade '`<`' i '`>`' se koriste za uključivanje datoteka sa sistemskim makroima.

Program 4.10. ilustruje korišćenje iskaza `#include`. Makroi `SQUARE`, `MAX`, `MIN` i `TO_UPPER` su u datoteci `macro.h` na istom direktorijumu na kojem se nalazi izvorni program. Makro `MIN` određuje manju vrednost od dve zadate.

```
/*Program 4.10.*/
```

```
/*Program za izlustraciju iskaza #include*/
```

```
#include "macro.h"
```

```
main()
```

```
{
float x=2.5, y=0.5;
char *s;

s="Borivoje";
printf("SQUARE(%.2f)=%.2f\n", x, SQUARE(x));
printf("MAX(%.2f, %.2f)=%.2f\n", x, y, MAX(x, y));
printf("MIN(%.2f, %.2f)=%.2f\n", x, y, MIN(x, y));
printf("TO_UPPER(%s)=", s);
while (*s)
{
printf("%c", TO_UPPER(*s));
++s;
}
printf("\n");
}
```

Program 4.10. Izlaz

```
SQUARE(2.50)=6.25
MAX(2.50, 0.50)=2.50
MIN(2.50, 0.50)=0.50
TO_UPPER(Borivoje)=BORIVOJE
```

Ovom diskusijom završava se poglavlje o osnovama C jezika. Sa narednim poglavljem počinjemo sa detaljnim opisom mogućnosti C jezika.

## 4.5. OSNOVNI TIPOVI PODATAKA I ARITMETIČKI IZRAZI

U ovom poglavlju opisane su konstante i promenljive, osnovni tipovi podataka, aritmetički operatori, osnovna pravila za formiranje aritmetičkih izraza, operator dodeljivanja vrednosti i pravila za konverziju tipova podataka.

### 4.5.1. Osnovni tipovi podataka

Promenljive i konstante su objekti kojima program manipuliše. Identifikatori se koriste za dodeljivanje imena promenljivim i konstantama. Tip promenljive određuje tip vrednosti ili podatka, koji se može dodeliti promenljivoj, skup operatora, koji se može primeniti nad tim vrednostima i veličinu memorijskog prostora za smeštaj vrednosti promenljive. Osnovni tipovi podataka su int, float, double i char .

### 4.5.2. Tip int

Podaci tipa int su celobrojne vrednosti. U ovu kategoriju podataka ubrajaju se celobrojne konstante, promenljive, izrazi i funkcije. U daljem tekstu opisana su pravila za pisanje i deklarisanje celobrojnih konstanti i promenljivih .

U C jeziku celobrojne konstante sastoje se od sekvence jedne ili više cifara sa opcionalnim karakterom '-' (minus) ispred sekvence. Prazna mesta u sekvenci nisu dozvoljena. U C jeziku postoje tri notacije ( brojni sistemi ) za predstavljanje celobrojnih konstanti: dekadni ( baza 10 ), oktalni ( baza 8 ) i heksadecimalni ( baza 16 ).

Dekadnu notaciju koristimo u svakodnevnom životu. Legalne cifre su 0,1,...,9. U funkcijama printf i scanf konverzioni karakter za dekadne vrednosti je d.

Oktalnu notaciju dobijamo ako je prva cifra konstante 0. U tom slučaju ostale cifre mogu biti 0,1,...,7. U funkcijama printf i scanf konverzioni karakter za oktalne vrednosti je o. Heksadecimalnu vrednost dobijamo, ako su prve cifre 0 i x ili X (tj.0x ili 0X). Dozvoljene cifre su 0,1,..., 9, a slova a,b,c,d,e,f (ili A,B,C,D,E,F) koristimo za cifre čije su vrednosti 10, 11,12,13,14 i 15 respektivno. U funkcijama printf i scanf konverzioni karakter za heksadecimalne vrednosti je x.

Ispravne celobrojne konstante su :

0001	0xFFf	0777	0Xab05
12345 0	0Xabcd	0123	0X1Ef0

a neispravne

0x10.5	123.4	0X0G1	123.3e2
0 77	.0001	+12.5	0xAF

Ključna reč int deklarise celobrojnu promenljivu. Deklaracije tipa int imali smo u primerima u prethodnom poglavlju. Navešćemo neke :

int x, y, z;

int a, b, c;

Početne vrednosti promenljivih nisu definisane, jer se rezervisane memorijske lokacije ne inicijalizuju automatski prilikom deklaracije. To se može postići primenom operatora dodeljivanja vrednosti na početku programa. Pogledajmo program 4. 11.

```
/*Program 4.11*/
main()
{
  int x,y,z;
  x = 1;
  y = 2;
  z = 3;
  printf(" x = %d\n y = %d\n z = %d\n", x, y, z);
}
Program 4.11 izlaz
```

```
x=1
y=2
z=3
```

U programu 4. 11 iskazi

```
x=1; y=2; z=3;
```

inicijalizuju promenljive x, y i z na vrednosti 1, 2 i 3, respektivno. Upotrebljena su tri iskaza što nije optimalno. U uslovima čestog izvršavanja programa vreme potrebno za inicijalizaciju nije zanemarljivo. C jezik ima mogućnost eksplicitne inicijalizacije u deklaraciji. Pogledajmo program 4.12. Deklarativnim iskazom

```
int x = 11, y = 22, z = 33;
```

deklarišu se promenljive: x kao tip int sa početnom vrednošću 11, y kao tip int sa početnom vrednošću 22 i z kao tip int sa početnom vrednošću 33. Očigledno je da je program 4.12 optimalniji od programa 4. 11.

```
/*Program 4.12*/
main()
{
  int x = 11, y = 22, z = 33;
  printf("x = %d\n y = %d\n z = %d\n", x, y, z);
}
```

**Program 4.12 Izlaz**

```
x = 11
y = 22
z = 33
```

Opseg celobrojnih vrednosti je različit od računara do računara i može se menjati primenom kvalifikatora long i short. Kvalifikator long povećava opseg vrednosti celobrojnih promenljivih. Na primer, na računaru PDP-11 opseg vrednosti promenljivih tipa int je -32768 do + 32767 . Deklaracijom tipa long int opseg vrednosti se povećava na -2147483648 do +2147483647. Međutim na računaru VAX-11 tip int i long int imaju jednake opsege (kao tip long int na računaru PDP-11).

Celobrojne konstante tipa long int se dobijaju dodavanjem karaktera L (ili l) na kraju konstante. Prazno mesto nije dozvoljeno između zadnje cifre konstante i L. Na primer, deklaracija long int adresa = 126256L deklariše celobrojnu promenljivu adresa kao tip long int sa početnom vrednošću 126256. Dodavanje karaktera L je opcionalno. Prevodilac tretira konstantu tipa int veću od maksimalne konstante tipa int kao konstantu tipa long int.

U funkcijama printf i scanf konverzioni karakter za dekadne, oktalne i heksadecimalne vrednosti tipa long int, su ld, lo i lx respektivno .

Kvalifikator short ispred int smanjuje opseg celobrojnih promenljivih. Motiv za njegovo uvođenje je štednja memorijskog prostora računara. Na mnogim računarima tip short int ima isti opseg kao i int. Na primer, na računaru VAX-11 short int memoriše celobrojne vrednosti u opsegu -32768 do + 32767 .

Kvalifikatori unsigned ispred int deklariše promenljivu tako da može memorisati samo pozitivne vrednosti. Na primer, promenljiva tipa int na 16-bitnim računarima ima vrednost između -32768 do +32767 , dok promenljiva tipa unsigned int uzima vrednosti između 0 do 65535.

Deklaracija

```
long int x;
short int y;
unsigned int z;
```

deklariše promenljive x, y i z kao tip long int, short int i unsigned int, respektivno. Ključna reč int se u deklaracijama može izostaviti, tako da gornja deklaracija postaje

```
long x;
short y;
unsigned z;
```

### 4.5.3. Tip float i double

Promenljive tipa float ili double memorišu vrednosti realnih brojeva, odnosno, vrednosti sa decimalnim ( pokretnim ) zarezom. Razlika između tipa float i double je u tačnosti predstavljanja realnih vrednosti. Tačnost predstavljanja realnih brojeva određena je brojem decimalnih cifara, koje se mogu memorisati u rezervisani memorijski

prostor za promenljive tipa float i double. U deklarativnim iskazima koriste se ključne reči float i double. Na primer, iskazima

```
float fx, fy;
double da, db;
```



promenljive  $fx$  i  $fy$  deklariraju se kao promenljive tipa `float`, a promenljive  $da$  i  $db$  kao promenljive tipa `double`. Konstante sa pokretnim zarezom se razlikuju od celobrojnih prisustvom decimalnog zareza.

Konstanta sa pokretnim zarezom se u C jeziku mogu predstaviti u običnoj i naučnoj (ili eksponencijalna) notaciji. Obična notacija je uobičajeno predstavljanje u formi realnih brojeva. Pri pisanju mogu se izostaviti delovi levo ili desno od decimalne tačke ali ne oba. Naučna notacija koristi karaktere `e` i `E`, tako da se vrednost broja  $1.34e3$  dobila kao proizvod  $1.34$  i broja  $10$  stepenovanog sa  $3$ , znači  $1340$ . Vrednost  $1.34$  je mantisa broja, a  $3$  je eksponent.

Ispravne konstante su

<code>5.56</code>	<code>.14</code>	<code>3.</code>
<code>6.64e +3</code>	<code>3E4</code>	<code>0.0</code>
a neispravne		
<code>3.54,8</code>	<code>3.896 8</code>	
<code>.e6</code>	<code>-8.86,</code>	

Ne postoji razlika između konstanti tipa `float` i `double`. C prevodilac sve konstante sa pokretnim zarezom predstavlja kao tip `double`.

Promenljive deklarirane kao `double` mogu memorisati grubo dvaput više decimalnih cifara od promenljivih tipa `float`. Tačan broj cifara, koje mogu biti memorisane, zavisi od računarskog sistema. Na primer, na računaru DEC VAX-11 promenljiva tipa `float` može memorisati aproksimativno 7 decimalnih cifara, dok promenljiva tipa `double` aproksimativno 16 decimalnih cifara. Broj decimalnih cifara igra važnu ulogu kod preciznih matematičkih izračunavanja, npr. u nuklearnoj fizici. Program 4.13 pokazuje tačnost vašeg računara. Iz programa 4.13 je očigledno, da je tačnost računara na kome su rađeni primer 6 i 15 decimalnih mesta (uporedite prvih 6 cifara promenljive  $\pi$  sa štampanim brojem, takođe i prvih 15 cifara promenljive  $\sqrt{3}$ ). U konverzionom nizu nalazi se `%.20f`. Vrednost `20` posle tačke određuje broj decimalnih mesta koje se štampaju. U funkciji `printf` standardan broj decimalnih mesta je `6` i ne zavisi od tipa štampane vrednosti.

*/\*Program 4.13\*/*

*main()*

{

*float pi = 3.14159265358979323845;*

*double sqrt\_3 = 1.7320508756887729356;*

*printf("pi = %.20f\nsqrt\_3 = %.20f\n", pi, sqrt\_3);*

}

**Program 4.13 Izlaz**

**pi = 3.1415927 4101257320000**

**sqrt\_3 = 1.73205087568877260000**

U funkcijama `printf` i `scanf` konverzioni karakteri za tipove `float` i `double` su `f` i `e` za običnu i naučnu notaciju respektivno,

#### 4.5.4. Tip `char`

Promenljive i konstante tipa `char` memorišu karaktere. U deklarativnim iskazima koristi se ključna reč `char`. Na primer, u iskazu

```
char ca, cb, cc;
```

promenljive `ca`, `cb` i `cc` deklarišu se kao promenljive tipa `char`. Konstante tipa `char` dobijaju se stavljanjem karaktera između jednostrukih navodnica, npr. `'A'`, `'1'`, itd. `'A'`

treba razlikovati od `"A"`, jer `"A"` predstavlja konstantni niz karaktera, koji u C jeziku ima posebnu predstavu i tretman .

Mnogi računari i svi personalni računari koriste standard ( ili kod ) ASCII (American Standard Code for Information Interchange) u kojem je svaki karakter kodiran 7-bitnom vrednošću ( 1 bajt ima 8 bita ) što daje mogućnost kodiranja 128 različitih karaktera.

Drugi standard EBCDIC (Extended Binary-Coded Decimal Interchange Code) koristi se u većim IBM računarima. Karakteri se kodiraju 8 bitnom celobrojnom vrednošću što daje 256 različitih vrednosti i nije kompatibilan sa ASCII kodom.

U daljim diskusijama koristi se isključivo kod ASCII. Kako su karakteri kodirani celobrojnim vrednostima, mogu se smatrati " malim " celobrojnim vrednostima i definisati uređenje među njima. Kodna vrednost karaktera `'A'` je manja od kodne vrednosti karaktera `'B'`, pa možemo reći da je `'A'` manje od `'B'`. Ovo uređenje se naziva leksikografsko uređenje i omogućava sortiranje karaktera, reči i linija.

U funkcijama `printf` i `scanf` konverzioni karakter za promenljive i konstante tipa `char` je `c`. Promenljive i konstante tipa `char` učestvuju u izrazima svojom kodnom vrednošću, koja se tretira kao celobrojna vrednost. Program 4.14 ilustruje tretman karaktera u C programima. Promenljiva `kar` sadrži karakter `'a'`. Sukcesivnim inkrementiranjem promenljive `kar` dobijaju se karakteri `'b'` i `'c'`. Poslednja funkcija `printf` pokazuje da se karakteri mogu tretirati kao male celobrojne vrednosti.

*/\*Program 4.14\*/*

```
main()
{
char kar = 'a';

printf("Karakter je %c\n", kar );

printf("Karakter je %c\n", kar +1);
printf("Karakter je %c\n", kar +2);
printf("Kodna vrednost je %d\n", kar+3);

}
```



---

```
/*Program 4.16 */
```

```
main()
{
printf("\tKarakter \t je prihvatio terminal\n\
\tkao svoj kontrolni karakter\n");
}
```

#### Program 4.6 Izlaz

**Karakter \t je prihvatio terminal  
kao svoj kontrolni karakter**

Karakter "" u nizu takođe je neophodno da prethodi karakter '\'. Na primer, iskaz `printf("On je rekao \"Zdravo\")`; štampa poruku `On je rekao "Zdravo"`

Slično, karakter " se može dodeliti promenljivoj tipa `char` iskazom : `c = '\`;

Forma `\nnn` omogućava da u niz karaktera uključimo proizvoljni karakter, gde je `nnn` maksimalno 3-cifren oktalni ceo broj i predstavlja kodnu predstavu karaktera. Na ovaj način omogućeno je da karaktere, koje ne možemo otkucati na tastaturi ipak unesemo u niz karaktera: Na primer, "bell" karakter ima kodnu vrednost 07 i ne može se uneti sa tastature, ali se može uneti u niz karaktera u formi `\7`(ili `\07`, ili `\007`). Program 4.17 generiše 5 kratkih zvučnih signala i štampa poruku o ozbiljnoj grešci u sistemu.

```
/*Program 4.17 */
```

```
main()
{
printf("\7\7\7\7\7 PANIC ERROR!\nSTOP!\n");
}
```

#### Program 4.17 Izlaz

**PANIC ERROR!  
STOP!**

### 4.6. Aritmetički izrazi i operatori

Izrazi su kombinacije konstanti, promenljivih, operatora i funkcija. Aritmetički izrazi su izrazi, koji sadrže aritmetičke operatore.

Aritmetički operatori C jezika `+`, `-`, `*`, `/` i `%` odgovaraju matematičkim operacijama sabiranja, oduzimanja, množenja, deljenja i deljenja po modulu, respektivno. Ovi operatori su binarni, jer zahtevaju dva operanda, sa izuzetkom operatora `-`, koji može biti i unaran ( zahteva samo jedan operand ). Ne postoji unaran operator `+`.

Operator deljenja / primenjen na vrednosti tipa int daje specifične rezultate. Naime, celobrojno deljenje odbacuje decimalni deo i zadržava samo celobrojni deo. Na primer, rezultat deljenja 1/2 je 0, 5/2 je 2, -8/3 je -2.

Operator deljenja po modulu se primenjuje po pravilu na pozitivne cele vrednosti. Izraz a%b primenjen na negativne cele vrednosti daje nepredviđen rezultat, koji se razlikuje od sistema do sistema. Rezultat operatora % je ostatak posle celobrojnog deljenja. Na primer, rezultat izraza 7%2 je 1, 10%4 je 2. Za pozitivne operande važi jednakost:

$$a/b*b+a%b=a$$

U programu 4.18 ilustrovana je upotreba aritmetičkih operatora. U konverzionom nizu zadnje funkcije printf nalaze se dva sukcesivna karaktera %%. Normalno, funkcija printf karakter '%' "shvata" kao početak konverzionog niza. U slučaju dvostrukog karaktera '%', funkcija printf "shvata" kao indikaciju da treba da štampa karakter '%' na mestu dvostrukog '%'.

*/\*Program 4.18\*/*

*main()*

```
{
int a = 202, b = 4, c = 50, d = 8;
printf("a=%d b = %d c = %d d = %d\n",a, b, c, d);
printf("c -d=%d\n", c -d);
printf("b * d =%d\n", b *d );
printf("c /d=%d\n", c/d );
printf("- a=%d\n", -a);
printf("a%%d=%d\n", a % d );
printf("b + d = %d\n", b + d );
printf("a / b * b + a%%b = %d\n", a / b * b + a % b);
}
```

**Program 4.18 Izlaz**

**a = 202 b = 4 c = 50 d = 8**

**c -d = 42**

**b \* d = 32**

**c/d=6**

**-a =-202**

**a%d=2**

**b + d = 12**

**a/b \* b+ a % b = 202**

U C jeziku definisan je prioritet i asocijativnost aritmetičkih operatora. Prioritet operatora određuje redosled izvršavanja operatora u izrazu, a asocijativnost operatora određuje redosled izvršavanja operatora jednakih prioriteta. Tako, na primer, u izrazu a\*b+c/d operatori \* i / imaju viši prioritet u odnosu na operator +. Znači, prvo se izvršavaju operatori \* i /, pa zatim +. Međutim u izrazu a\*b/d operatori \* i / imaju isti

prioritet, pa shodno asocijativnosti ovih operatora (s leva na desno), operatori \* i / se primenjuju s leva na desno. Znači, prvo se primenjuje \*, pa /. U tabeli 4.2 prikazani su prioriteti i asocijativnost aritmetičkih operatora.

Operatori u istoj liniji ( npr. \* / % ) imaju jednake prioritete i svi aritmetički operatori nižeg prioriteta navedeni su ispod te linije. Asocijativnost operatora je prikazana na desnoj strani tabele 4.2. Primenom zagrada ( I ) redosled izvršavanja operatora u izrazu se može menjati.

Operator	Asocijativnost
-(unarni) (tip)	s desna na levo
* / %	s leva na desno
+ -	s leva na desno
= + = -= * = /= % =	s desna na levo

Tabela 4.2 Prioritet aritmetičkih operatora

Program 4.19 ilustruje primenu aritmetičkih operatora. prioritet i asocijativnost izračunavajući navedene aritmetičke izraze.

*/\*Program 4.19 \*/*

*main()*

```
{
int a = 25, b = 4;
float c = 50.0, d = 8.0;
printf("a = %d b = %d c = %f d = %f\n", a, b, c, d);
printf("26 + a * b / c = %f\n", 26 + a * b / c);
printf("d / c + 100 = %f\n", d / c + 100);
printf("c / d * d = %f\n", c / d * d);
printf("a + b / c + d = %f\n", a + b / c + d);
}
```

**Program 4.19 Izlaz**

```
a = 25 b = 4 c = 50.000000 d = 8.000000
26 + a * b / c = 28.000000
d / c + 100 = 100.160000
c / d * d = 50.000000
a + b / c + d = 33.080000
```

#### 4.6.1. Operatori dodeljivanja vrednosti

Iskaz za dodeljivanje vrednosti ima formu:  $\text{promenljiva} = \text{izraz};$  u kojem se promenljivoj *promenljiva* dodeljuje vrednost izraza *izraz*. Karakter '=' , predstavlja operator dodeljivanja vrednosti. Operator i izraz dodeljivanja vrednosti ne treba mešati sa odgovarajućim matematičkim pojmovima. Operator dodeljivanja vrednosti je definisan slično ostalim aritmetičkim operatorima. Na primer, operator +

zahteva dva operanda u izrazu  $a+b$ . Vrednost izraza  $a + b$  je zbir vrednosti  $a$  i  $b$ . Slično ovome, operator  $=$  je binaran i zahteva dva operanda: promenljivu (levi operand), kojoj se dodeljuje vrednost i proizvoljni izraz (desni operand), čija se vrednost dodeljuje promenljivoj. Stoga, izraz za dodeljivanje vrednosti je forme (bez ;)

promenljiva = izraz

Vrednost desnog operanda predstavlja vrednost izraza za dodeljivanje vrednosti. Takva definicija operatora  $=$  dozvoljava da pišemo iskaze

```
int a, b, c;
a = (b = 10) + (c = 20);
```

Izrazi  $b = 10$  i  $c = 20$  moraju biti u zagradi, jer operator  $=$  ima niži prioritet u odnosu na  $+$  (tabela 4.2). Promenljiva  $b$  dobija vrednost 10,  $c$  vrednost 20, a promenljiva  $a$  vrednost 30.

Asocijativnost operatora  $=$  je sa desna na levo, pa možemo pisati

```
int a, b, c;
a = b = c = 10;
```

Sve tri promenljive dobijaju vrednost 10 i to redom  $c$ ,  $b$  i  $a$ .

U programiranju često se javljaju izrazi tipa :

```
I = I + 2 ili opštije
promenljiva = promenljiva op izraz
```

U C jeziku slični izrazi se mogu pisati kao :

```
1+=2 ili opštije
promenljiva op = izraz
```

Svi aritmetički operatori mogu graditi ovakve izraze. Operator  $op$  može biti  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ , pa dobijamo dvojne operatore  $+=$ ,  $-=$ ,  $/=$ ,  $*=$  i  $\%=$ , respektivno. U tabeli

4.2 prikazan je prioritet nekih operatora dodeljivanja vrednosti u odnosu na ostale aritmetičke operatore. Ispravni izrazi su:

$k * = 4 + I$	ekvivalentan $k = k *(4 + I)$
$k \% = m = 7 + . n/3$	ekvivalentan $k = k \% (m = 7 + n/3)$
$2 + 6 * (n + = 9)/8$	ekvivalentan $2 + 6 * (n = n + 9)/ 8$

ali je izraz

```
2 + 6*n+=9/8
```

Ilegalan, zato što leva strana operatora  $+=$  mora biti promenljiva .

#### 4.6.2. Konverzija tipova podataka

Tip rezultata aritmetičkih izraza zavisi od tipova operanada. Pogledajmo izraz  $x + y$ . Ako su oba operanda tipa `int`, onda je i rezultat tipa `int`. Šta se dešava ako je promenljiva  $x$  tipa `short`, a  $y$  tipa `int`. U tom slučaju prevodilac automatski konvertuje vrednost tipa `short` u vrednost tipa `int`. Kako su sada oba operanda tipa `int` i rezultat je tipa `int`. Promenljiva  $x$  i njena vrednost se ne menjaju jer se konvertuje kopija vrednosti, koja učestvuje u izrazu.

Postupak konverzije je sledeći:

1. `char` i `short` se konvertuju u `int`
2. Ako su operandi izraza posle prvog koraka različiti na osnovu hijerarhije:

`int < unsigned < long < unsigned < long < float < double`

operandi nižeg tipa se konvertuju u tip višeg nivoa sve dok svi operandi ne budu istog tipa koji postaje tip rezultata.

Postupak konverzije biće jasniji posle nekoliko primera. Pretpostavimo da imamo sledeće promenljive:

`char c; short s; int i; unsigned u;`  
`long l; float f; double d;`

U tabeli 4.3 dato je više primera izraza, kao i tip dobijenog rezultata. Shodno postupku automatske konverzije u izrazu `c-s/i`, `short` i `char` se konvertuju u `int`, pa je rezultat tipa `int`. U izrazu `u*5-i` celobrojna konstanta `5` se konvertuje u `unsigned`, a i u `unsigned`. Rezultat je `unsigned`. Treba naglasiti da se konstanta `5` memoriše kao `int` a `5.0` kao `double`.

Kod operatora dodeljivanja vrednosti stvari su drugačije. Ako imamo dodeljivanje `d = i` prevodilac vrednost promenljive `i` konvertuje u `double` i dodeljuje promenljivoj `d`. Ali u izrazu `I=d` prevodilac vrednost promenljive `d` konvertuje u `int` pri čemu dolazi do gubitka decimalnih cifara. Takva celobrojna vrednost dodeljuje se promenljivoj `i`.

#### 4.7. KONTROLA TOKA PROGRAMA

U svim dosadašnjim C programima iskazi su se izvršavali jednom i samo jednom, po redosledu pojavljivanja u programu. Programi ovakvog tipa su jednoslavni, jer ne sadrže logičke kontrolne strukture, kao što su: testovi za određivanje da li je određen uslov ispunjen ili nije, ponavljajuće izvršavanje grupe iskaza ili selekcija jedne grupe iskaza od više mogućih. Mnogi programi, koji su od praktičnog interesa, intenzivno koriste takve mogućnosti. U ovom poglavlju opisani su relacioni i logički izrazi i operatori, operatori inkrementiranja i dekrementiranja, iskazi za ostvarivanje petlji (iskazi `while`, `for` i `do-while`) i iskazi za ostvarivanje programskih grananja (iskazi `if`, `if-else`, `else-if` i `switch`), iskazi bezuslovnog grananja (iskazi `break`, `goto` i `continue`) i uslovni operator.



#### 4.7.1. Relacioni operatori i izrazi

Relacioni operatori, prikazani u tabeli 5.1, uspostavljaju relaciju između levog i desnog operanda. Operandi mogu biti proizvoljni izrazi. Svi relacioni operatori su binarni. Svaki od njih zahteva dva operanda i daje kao rezultat vrednost 0 ili 1 tipa int. Ispravni primeri relacionih izraza su :

x < y	3.14 < = 99.99	y = z
a > c	b != d	e > = f

a nekorektni

x < = y	b > = d
a > = d	a > > b

manji od	<
veći od	>
manji ili jednak od	< =
veći ili jednak od	> =
jednak	==
nije jednak	=

Tabela 4.1 Relacioni operatori

Razmotrimo relacioni izraz  $a > c$ . Jasno je da, ako je vrednost izraza  $a$  veća od vrednosti izraza  $c$ , tada je izraz  $a > c$  istinit. U tom slučaju, izraz  $a > c$  dobija vrednost 1 tipa int. U drugom slučaju, ako je vrednost izraza  $a$  manja ili jednaka vrednosti izraza  $c$ , tada je izraz  $a > c$  neistinit. U ovom slučaju, izraz  $a > c$  dobija vrednost 0 tipa int. Znači, moguće su dve vrednosti izraza  $a > c$ , 1 tipa int, ako je istinit ili 0 tipa int, ako je neistinit.

Relacioni operatori slede standardna pravila za konverziju tipova podataka u aritmetičkim izrazima.

Relacioni operatori  $<$ ,  $< =$ ,  $>$  i  $> =$  imaju jednak prioritet i viši su u odnosu na  $=$  i  $| =$  (Operatori  $=$  i  $| =$  su jednakog prioriteta). Aritmetički operatori su višeg prioriteta u odnosu na relacione operatore.

#### 4.7.2. Logički operatori i izrazi

Logički operatori, prikazani u tabeli 5.2, realizuju logičke operacije AND, OR i NOT.

logički AND	&&
logički OR	
logički NOT	

Tabela 5.2 Logički operatori

Operator `|` (logička negacija) je unaran operator i može se primeniti na proizvoljni izraz. Ako izraz ima vrednost 0 bilo kog tipa, primenom operatora negacije dobija se vrednost 1 tip `int`. Ako izraz ima nenultu vrednost bilo kog tipa, dobija se vrednost 0 tipa `int`. Opšta forma logičke negacije je : `| Izraz`

Ispravni primeri su:

$$\begin{array}{ccc} |10 & |x & |'x' \\ |(x+3.14) & & |(5/(x+y)+3+'a') \end{array}$$

Operator logičke negacije u C jeziku se razlikuje od negacije u matematici. Ako je `x` matematički izraz tada za operator matematičke negacije važi jednakost :

$$\text{not}(\text{not } x) = x$$

Međutim, ako izraz `x` ima nenultu vrednost bilo kog tipa tada za operator logičke negacije važi jednakost :

$$|(x) = 1$$

S druge strane, ako izraz `x` ima vrednost 0 tada važi jednakost  $|(x) = 0$ . Operator `|` ima jednak prioritet i asocijativnost (s desna na levo) kao i ostali unarni operatori.

Binarni logički operatori `&&` i `||` realizuju logičke operacije AND i OR, respektivno. Opšta forma izraza sa ovim operatorima je :

$$\text{Izraz1 } \&\& \text{ izraz2}$$

$$\text{Izraz1 } || \text{ izraz2}$$

Operatori `&&` i `||` daju kao rezultat vrednosti 0 ili 1 tipa `int`. Operator `&&` daje kao rezultat vrednost 1, ako oba operanda imaju nenultu vrednost. U protivnom, rezultat je 0. Operator `||` daje kao rezultat vrednost 0 ako oba operanda imaju vrednost 0. U protivnom daje vrednost 1.

Logički operatori slede standardna pravila za konverziju tipova podataka u aritmetičkim izrazima. U logičkim izrazima izračunavanje se prekida čim se rezultat celog izraza može odrediti na osnovu dotadašnjih izračunavanja. Na primer, u izrazu `izraz1 && izraz2` `izraz2` neće biti izračunavan, ako `izraz1` ima vrednost 0. Slično u izrazu `izraz1 || izraz2`

`izraz2` neće biti izračunavan ako `izraz1` ima vrednost 1. Program 5.1 potvrđuje ove navode. Na osnovu izlaza programa 5.1 promenljive `x` i `i` imaju vrednost 0. Kako se u iskazu :

$$x = 0 \&\& (i = 999);$$

zahteva dodeljivanje vrednosti 999 promenljivoj `i`, jasno je da se izraz `i = 999` nije izvršio. Drugim rečima, izračunava se vrednost levog operanda, pa kako taj operand ima vrednost 0, desni operand se ne izračunava. Iz tih razloga promenljiva `i` zadržava početnu vrednost. Slična pravila važe i za operator `||`. Kako levi operand operatora `||` ima vrednost 1, desni operand se ne izračunava i vrednost 9999 ne dodeljuje promenljivoj `i`. Promenljiva `i` zadržava početnu vrednost.

Tabela 5.3 Semantika operatora `&&` i `||`

Izraz1	Izraz2	Izraz1 && Izraz2	Izraz1    Izraz2
0	0	0	0
0	#0	0	1
#0	0	0	1
#0	#0	1	1

---



---

```
/*Program 4.20 */
```

```
main()
{
int i=0,j=0,x,y;
x = 0 && (i = 999);
printf("Operator &&\n");
printf("x=%d i=%d\n", x , i);
y = 1 || ( j = 9999 );
printf("Operator ||\n");
printf("y=%d j=%d\n", y , j);
}
Program 4.20 Izlaz
Operator &&
x=0 i=0
Operator ||
y=0 j=0
```

### 4.7.3. Operator inkrementiranja i dekrementiranja

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

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

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

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

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

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

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

Operatori inkrementiranja i dekrementiranja su predstavljeni u programu 4.21

```
/*Program 4.21 */
```

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

```

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

```

#### Program 4.21 Izlaz

```

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

```

Izraz  $a = ++c + c$  iz programa 5.2 ukazuje na izvor čestih grešaka. Naime, ovaj izraz zavisi od realizacije C prevodioca pa ga stoga treba izbegavati. Zbog čega? C prevodilac može, izračunavajući  $++c + c$  prvo da inkrementira  $++c$  pa tek onda da preuzme vrednost promenljive  $c$  kao vrednost drugog operanda. Drugi C prevodilac može prvo da preuzme vrednost promenljive  $c$  kao vrednost drugog operanda, pa tek potom da inkrementira  $++c$ . Očigledno da ovakve neodređenosti nisu poželjne u programima. Zato izraz  $++c + c$  treba pisati u obliku  $++c$  i  $a=c+c$  čime se eksplicitno navodi redosled izračunavanja gornjih izraza i time izbegava neodređenost redosleda izračunavanja.

Razmotrićemo operatore `,` ( zarez ) i `sizeof`. Operator `,` ima najniži prioritet među operatorima C jezika. To je binarni operator sa izrazima kao operandima: `izraz1,izraz2` što govori da se `izraz1` izračunava kao prvi a `izraz2` kao drugi.

Operator `sizeof` kao rezultat daje vrednost tipa `int`, koja je jednaka broju bajtova potrebnih za memorisanje operanda. Opšta forma ovog operatora je :

`sizeof(tip_podatka)`, na primer `sizeof(int)`

daje kao rezultat broj bajtova koji zauzima tip `int`. Program 4.22 koristi operator `sizeof` određujući broj bajtova za osnovne tipove podataka. Na osnovu izlaza programa 4.22 može se zaključiti da se za tip `char` koristi 1 bajt, za tip `short` 2 bajta, za tip `int` 2 bajta, za tip `long` 4 bajta, za tip `unsigned` 2 bajta, za tip `float` 4 bajta i za tip `double` 8 bajtova (16-bitni računar).

```

/*Program 4.22*/
main()
{
printf("char=%d bajt\n", sizeof( char ));
printf("short=%d bajta\n", sizeof(short));
printf("int=%d bajta\n", sizeof(int));
printf("long=%d bajta\n", sizeof(long));
printf("unsigned=%d bajta\n", sizeof(unsigned ));
printf("float=%d bajta\n", sizeof(float));
printf("double=%d bajtova\n", sizeof(double));
}

```

**Program 4.22 Izlaz**

```
char=1 bajt
short=2 bajta
int=2 bajta
long=4 bajta
unsigned=2 bajta
float=4 bajta
double=8 bajtova
```

**4.7.4. Složeni i bezefektni iskazi**

Složeni iskaz je sekvenca iskaza objedinjena vitičastim zagradama { } . Osnovni razlog definisanja složenog iskaza je grupisanje iskaza u jednu izvršnu celinu. U C jeziku gde god je korektno staviti jedan iskaz, korektno je staviti i složeni iskaz. Drugim rečima, složeni iskaz je sam za sebe iskaz. Primeri složenih iskaza su :

```
    {
      x += 1;
      y = x + 10;
      printf("x + y = %d x-y= %d\n", x + y, x-y);
    }
```

Bezefektni iskaz ne sadrži izraze, pa stoga ne obavlja nikakvu obradu. Bezefektni iskaz sadrži samo terminator iskaza ; (tačka-zarez). Koristi se na mestima gde je iskaz potreban, shodno sintaksi C jezika.

**4.7.5. While iskaz**

Kontrolna struktura while omogućava uslovno ponavljanje jednog ili više iskaza. U konstrukciji forme:

```
while (uslov_petlje)
    while_iskaz;
    sledeći_iskaz;
```

prvo se izračunava uslov\_petlje. Ako uslov\_petlje ima vrednost 0 ( neistinit ), kontrola programa se prenosi na sledeći iskaz. Ako je različit od nule ( istinit ) izvršava se while\_iskaz i kontrola prenosi na početak while strukture i uslov\_petlje ponovo izračunava. Stoga, while\_iskaz se izvršava sve dok je uslov\_petlje različit od nule ( istinit ). Efekat strukture while je da se while\_iskaz izvršava ni jednom ili više puta zavisno od vrednosti uslova\_petlje. Uslov\_petlje može biti izraz proizvoljne složenosti. Strukturom while može se jednostavno realizovati beskonačna petlja. Kao primer korišćenja strukture while za realizaciju uslovne petlje pogledajmo program 4.23 za nalaženje najvećeg zajedničkog delioca (NZD) dva nenegativna broja. NZD je najveći broj koji deli dva broja bez ostatka. Naprimera, za brojeve 12 i 18 NZD je 6. U programu 4.23 korišćen je algoritam, koji je definisao Euclid 300 g.p.n.e.

```
/*Program 4.23 */
/* Program za nalazenje najveceg zajednickog delioca dva nenegativna cela broja */
main()
{
    int prvi, drugi, pomocni;
    printf("Ukucajte dva nenegativna cela broja ?\n");
```

```
scanf("%d%d", &prvi, &drugi);
while (drugi==0)
{
pomocni = prvi % drugi;
prvi = drugi;
drugi = pomocni;
}
printf("Najveci zajednicki delilac ovih brojeva je %d\n", prvi);
}
```

### Program 4.23 Izlaz

Ukucajte dva nenegativna cela broja ?

354 118

Najveci zajednicki delilac ovih brojeva je 118

Dvostruki `%d%d` u ulaznoj funkciji `scanf` omogućava učitavanje dva cela broja. Prvi od njih se memoriše u promenljivu `prvi` a drugi u promenljivu `drugi`. Brojevi moraju biti razdvojeni sa jednim ili više praznih mesta ili novom linijom (ne zarezom ili drugim znakom). Za izračunavanje NZD koristi se `while` petlja. Posle izlaska iz `while` petlje vrednost promenljive `prvi` predstavlja NZD.

Napomenimo da se `while (drugi i = 0)` može napisati kao `while (drugi) U` programu 4.24 takođe koristimo strukturu `while`. Program 4.24 invertuje cifre učitano broja.

```
/*Program 4.24 */
/* Program za permutovanje cifara celog broja - while iskaz */
main()
{
int broj;

printf("Ukucajte ceo broj ? ");
scanf("%d", &broj);
printf("Permutovani broj je ");
while ( broj )
{
printf("%d", broj % 10);
broj = broj/10;
}
printf("\n");
}
```

### Program 4.24 Izlaz

Ukucajte ceo broj? 8612

Permutovani broj je 2168

Algoritam korišćen u programu 4.24 objasnićemo primerom. Uzmimo broj 4321. Treba da dobijemo 1234. Prvu cifru dobijamo kao ostatak celobrojnog deljenja 4321 sa 10, tj.  $4321 \% 10 = 1$ . Zatim delimo  $4321 / 10 = 432$ . Postupak se ponavlja sve dok rezultat celobrojnog deljenja ne bude 0.

U programu 4.25 while struktura se izvršava ako je broj učitanih brojeva manji od specificiranog broja (  $\text{count} < n$  ). Zbir svih brojeva je memorisan u promenljivoj sumi. Srednja vrednost  $\text{suma}/n$  se štampa posle izlaska iz strukture while.

```

/*Program 4.25 */
/* Program za određivanje srednje vrednosti n celih pozitivnih brojeva -while iskaz */
main()
{
int n, brojac = 0;
float suma = 0, x;
printf("Ukupno brojeva ? ");
scanf("%d", &n);

while (brojac < n)
{
printf("Ukucajte %d. broj ? ", brojac+1);
scanf("%f", &x);
suma += x;
brojac += 1;
}
printf("Srednja vrednost ovih brojeva je %f\n", suma/n);
}
Program 4.25 Izlaz

```

```

Ukupno brojeva ? 4
Ukucajte 1. broj ? 34.345
Ukucajte 2. broj ? 234.2876
Ukucajte 3. broj ? 45.98
Ukucajte 4. broj ? 453.32
Srednja vrednost ovih brojeva je 191.983154

```

Struktura while može biti ugnježdjena u drugu while strukturu. Ugnježdenost može biti proizvoljne dubine. Znači da sledeću konstrukciju možemo koristiti u programima :

```

while (...)
{
    while(...)
    {
        while(...)
        {
        }
    }
}

```

**4.7.6. For iskaz**

Opšta forma kontrolne strukture for je

```
for (početni_izraz; uslov_petlje; izraz_petlje)
    programski_iskaz;
    sledeći_iskaz;
```

i semantički je ekvivalentan

```
pocetni_izraz;
while (uslov_petlje)
{
    programski_iskaz; izraz_petlje;
}
sledeci_iskaz;
```

Prva komponenta u for iskazu, početni\_izraz, postavlja početne parametre petlje. U programu 4.26 ovaj deo for iskaza inicijalizuje promenljivu i na 1. Početni\_izraz može biti jedan ili više izraza C jezika. Njihovo povezivanje ( razdvajanje ) se ostvaruje operatorom , ( zarez ), čime se jednoznačno određuje redosled njihovog izvršavanja.

Druga komponenta for iskaza specificira uslov koji je potreban da bi se petlja nastavila. Uslov\_petlje može biti proizvoljni izraz C jezika. U programu 4.26 to je uslov  $i \leq n$ . Ako je uslov\_petlje različit telo petlje može biti jedan ili više iskaza koji su dozvoljeni u C jeziku. Ako je uslov\_petlje jednak 0 (neistinit) telo petlje se ne izvršava i kontrola se prenosi na sledeći\_iskaz.

Treća komponenta for iskaza sadrži iskaz, koji se izvršava na kraju svakog ciklusa petlje, nakon što se izvrši telo petlje. Izraz\_petlje može biti jedan ili više izraza C jezika. Njihovo povezivanje se ostvaruje operatorom zarez, čime se definiše redosled njihovog izvršavanja. Program 4.26 ilustruje primenu iskaza for, izračunavajući faktorijel učitano broja. Početni\_izraz je izraz  $i = 1$  kojim se ažurira promenljiva za kontrolu petlje.

*/\*Program 4.26 \*/*

```
/* Program za izracunavanje faktorijela */
#include <stdio.h>
#include <math.h>
main()
{
    int i, n;
    long fak = 1;
    printf("Izracunavanje n!\nUkucajte broj ? ");
    scanf("%d", &n);
    for ( i=1; i<=n; ++i)
        fak *= i;
    printf("%d! = %ld\n", n, fak);
}
```

**Program 4.26 Izlaz**

**Izracunavanje n!**

**Ukucajte broj ? 6**

**6! = 720 ,**



for iskazi mogu biti ugnježdjeni. U C jeziku konstrukcija

```
for (...)
    for (...)
        for (..)
            iskaz;
```

je potpuno legalna. For i while iskazi se koriste za realizaciju petlje, pa je njihovo korišćenje stvar ukusa. Malu prednost ima for iskaz jer se inicijalizacija, uslov i izraz petlje nalaze na vrhu petlje, što u uslovima ugnježdjenih for iskaza povećava čitljivost programa. Program 4.27 je reprogramiran program 4.24 i koristi for iskaz. Upoređivanjem programa 4.24 i 4.27 evidentna je prednost for iskaza u odnosu na while iskaz u ovom slučaju. Izračunavanje sume brojeva je smešteno u izrazu petlje iskaza for. Izraz petlje sastoji se od dva izraza: ++ brojac i suma+= x. Promenljiva brojac kontroliše izvršavanje petlje :

```
/*Program 4.27 */
```

```
/* Program za odredjivanje srednje vrednosti n celih pozitivnih brojeva -for iskaz */
```

```
main()
{
    int n, brojac =1;

    float suma =0, x ;
    printf("Ukupno brojeva ? ");
    scanf("%d", &n);

    for ( ; brojac<=n; ++brojac, suma+=x )
    {
        printf("Ukucajte %d broj ?",brojac);
        scanf("%f",&x);
    }
    printf("Srednja vrednost ovih brojeva je %f\n", suma/n);
}
```

**Program 4.27 Izlaz**

```
Ukupno brojeva ? 4
Ukucajte 1. broj ? 34.345
Ukucajte 2. broj ? 234.2876
Ukucajte 3. broj ? 45.98
Ukucajte 4. broj ? 453.32
Srednja vrednost ovih brojeva e 191.983154
```

#### 4.7.7. Do-while iskaz

U konstrukciji for i while uslov petlje se izračunava na početku petlje. Posledica je da se telo petlje izvršava ni jednom ili više puta u zavisnosti od vrednosti uslova petlje. U pojedinim primenama poželjno je da se uslov izračunava na kraju petlje, što znači da

se telo petlje izvršava jednom ili više puta. U C jeziku ova mogućnost se ostvaruje pomoću do-while iskaza. Sintaksa do-while iskaza je :

```
do
do_iskaz;
while (uslov_petlje)
sledeci_iskaz;
```

U do - while iskazu prvo se izvršava do-iskaz. Zatim se izvršava uslov\_petlje. Ako je istinit, petlja se nastavlja, inače se petlja završava i kontrola prenosi na sledeći\_iskaz. Može se zaključiti, da je do - while iskaz varijanta while iskaza sa uslovom\_petlje na kraju petlje. Zato se do - while iskaz koristi za slučaj da je petlju neophodno izvršiti najmanje jedanput.

U programu 4.24 invertujemo cifre učitano broja koristeći while iskaz. Program 4.28 je reprogramirani program 4.24 sa korišćenjem do-while iskaza.

```
/*Program 4.28 */
/* Program za permutovanje cifara
nenegativriog celog broja - do-while iskaz */
main()
{
int broj;
printf("Ukucajte ceo broj ? ");
scanf("%d", &broj);
printf("Permutovani broj je ");

do
{
printf("%d", broj%10);
broj =broj/10;
}
while (broj);
printf("\n");
}
```

Program 4.28 Izlaz

```
Ukucajte ceo broj ? 3486
Permutovani broj je 6843
```

#### 4.7.8. If iskaz

If je kontrolna struktura za ostvarivanje programskih grananja. Ova programska struktura ima formu :

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

Semantika ove konstrukcije je sledeća: Ako je vrednost izraz različita od nule (istinit), izvršava se iskaz, a zatim sledeći\_iskaz. U protivnom, ako je vrednost izraza jednaka nuli (neistinit), iskaz se ne izvršava i kontrola direktno prenosi na sledeći iskaz.

Stoga, if iskaz obezbeđuje jednostruko grananje ili odlučivanje. Na primer u fragmentu programa :

```
if (test == FALSE )
printf("Uredjaj neispravan!\n");
```

ako je uslov test = FALSE istinit, izvršava se funkcija printf i štampa se poruka da je uređaj neispravan. Ako je test = FALSE neistinit, funkcija printf se ne izvršava, poruka se ne štampa, a kontrola programa predaje se iskazu neposredno nakon if iskaza. Izraz i iskaz mogu biti svaki izraz i iskaz dozvoljen u C jeziku. Zagrada u if iskazu su obavezne.

Program 4.29 pokazuje jednostavnu upotrebu if iskaza. Program najpre učitava vrednost tipa float, a zatim određuje i štampa apsolutnu vrednost učitane vrednosti.

```
/*Program 4.29 */
/* Program za izracunavanje apsolutne vrednosti celog broja */
main()
{
int broj, a_vred;
printf("Ukucajte ceo broj ? ");
scanf("%d", &broj);
a_vred = broj;

if (a_vred < 0)
a_vred = -a_vred;
printf("Apsolutna vrednost broja %d je %d\n", broj, a_vred);
}
```

**Program 4.29 Izlaz**

Ukucajte ceo broj ? -12

Apsolutna vrednost broja -12 je 12

U programu 4.29 izraz a\_vred = -a\_vred se izvršava samo ako je vrednost promenljive a\_vred manja od nule. U protivnom, izraz a\_vred = -a\_vred se ne izvršava. If iskazi mogu biti ugnježdjeni. Program 5.11 učitava karakter i određuje da li prpada malim ili velikim slovima, ciframa ili specijalnim karakterima. U if iskazima koriste se složeni logički izrazi.

Program 4.30 ukazuje na nedostatke if iskaza. Prvo, ako je promenljiva c sadrži malo slovo, nepotrebno se izračunavaju izrazi u prva dva if iskaza. Drugo, u zadnjem if iskazu ispituje se da li karakter pripada specijalnim karakterima. Kako su specijalni karakteri razvrstani u četiri grupe, mora se napisati logički izraz za svaku grupu ponaosob. Treće, mora se dobro poznavati ASCII kod. Iz tih i sličnih razloga u C jeziku postoje još dve varljante **If** iskaza. To su if-else i else-if iskazi.

```
/*Program 4.30 */
/* Program za odredjivanje da li je karakter malo
ili veliko slovo, cifra ili specijalni karakter -if iskaz */
main()
{
char c;
printf("Ukucajte karakter ? ");
```

```
scanf("%c", &c);
if(c>='a' && c<='z')
printf("Karakter %c je malo slovo\n", c);
if (c >= 'A' && c <= 'Z')
printf("Karakter %c je veliko slovo\n", c);
if ( c >= '0' && c <= '9')
printf("Karakter %c je cifra\n", c);
if (c > ' ' && c <= '/' || c >= ';' && c <= '@' ||
    c >= '!' && c >= '~' || c >= '}')
printf("Karakter %c je specijalni karakter\n", c);}
```

#### Program 4.30 Izlaz

```
Ukucajte karakter ? 1
Karakter 1 je cifra
Ukucajte karakter ? a
Karakter a je malo slovo
Ukucajte karakter ? D
Karakter D je veliko slovo
Ukucajte karakter ? >
Karakter > je specijalni karakter
```

#### 4.7.9. If-else iskaz

If iskaz ostvaruje jednostruko odlučivanje ili grananje. Koristrukcijom if-else ostvarujemo dvostruko odlučivanje ili grananje. Opšta forma if-else iskaza je :

```
if (izraz)
    iskaz1;
else
    iskaz2;
sledeci_iskaz;
```

Semantika if-else iskaza je sledeća: Ako je vrednost izraza različita od nule (istinit) izvršava se iskaz1, iskaz2 preskače i kontrola prenosi na sledeći\_iskaz. Ako je vrednost izraza jednaka nuli (neistinit) preskače se iskaz1 a izvršava iskaz2. Primer if-else iskaza je :

```
if (test == FALSE)
printf("Uredjaj nije u redu\n");
else
printf("Uredjaj je u redu\n");
```

Program 4.31 koristi if-else iskaz određujući da li je učitana godina prestupna ili ne. Godina je prestupna ako je deljiva bez ostatka sa 4 a nije sa 100, pri čemu je svaka 400 godina prestupna.

```

/*Program 4.31*/
/* Program za određivanje da li je godina prestupna ili nije*/
main()
{
int godina,ost_4,ost_100,ost_400;
printf("Ukucajte godinu ? ");
scanf("%d", &godina);
ost_4=godina%4;
ost_100=godina%100;
ost_400=godina%400;
if (ost_4 ==0 && ost_100 != 0 || ost_400== 0)
printf("Godina %d je prestupna\n", godina);
else
printf("Godina %d nije prestupna\n", godina);
}

```

**Program 4.31 Izlaz**

Ukucajte godinu ? 1972

Godina 1972 je prestupna

Ukucajte godinu ? 1970

Godina 1970 nij'e prestupna .

**4.7.10. Else-if iskaz**

C jezik obezbeđuje kontrolnu strukturu za višestruko grananje. To je else-if iskaz. Opšta forma ove kontrolne strukture je sledeća :

```

if (izraz1 )
    iskaz1;
else if (izraz2)
    iskaz2;
.....
else if (izrazN)
    iskazN;
else
    iskaz0;
sledeci_iskaz;

```

u kojoj iskazl ( $i = 1, \dots, N$ ) može biti složeni iskaz. Semantika ove složene kontrolne strukture je sledeća: Ako je izrazl ( $i = 1, \dots, N$ ) istinit (različit od 0) izvršava se iskazl ( $i = 1, \dots, N$ ), preskače se preostali deo konstrukcije i kontrola prenosi na sledeci\_iskaz. Ako je izrazl ( $i = 1, \dots, N$ ) neistinit (različit od 0) iskazl ( $i = 1, \dots, N$ ) se preskače, izračunava se izrazl + 1 i postupak se ponavlja kao za izrazl. Ako nijedan izrazl ( $i = 1, \dots, N$ ) nije istinit izvršava se iskaz0, koji je opcionalan u konstrukciji. Bitno je uočiti da se izvršava samo jedan iskaz u strukturi, čiji je odgovarajući izraz istinit (različit od 0), nakon čega se kontrola prenosi na sledeci\_iskaz.

else-if konstrukciju bolje ćemo razumeti na primeru:

```

/*Program 4.32 */
/* Program za određivanje da li je karakter malo ili veliko slovo, cifra ili specijalni karakter -
else-if iskaz */
main()
{
char c;
printf("Ukucajte karakter ? ");
scanf("%c", &c);
if ( c >= 'a' && c <'z' )
    printf("Karakter %c je malo slovo\n", c);
else if (c >= 'A' && c <= 'Z')
    printf("Karakter %c je veliko slovo\n", c);
else if (c >= '0' && c <= '9')
    printf("Karakter %c je cifra\n", c);
else
    printf("Karakter %c je specijalni karakter\n", c);
}

```

Program 4.32 Izlaz

```

Ukucajte karakter ? 1
Karakter 1 je cifra ,
Ukucajte karakter ? a
Karakter a je malo slovo
Ukucajte karakter ? D
Karakter D je veliko slovo
Ukucajte karakter ? $
Karakter $ je specijalni karakter

```

#### 4.7.11. switch iskaz

Konstrukcija switch omogućava višestruko grananje. Slična je konstrukciji else-if, ali je preglednija kod izbora jedne od više mogućih alternativa. Međutim, dok se u konstrukciji else-if u najgorem slučaju izračunava N izraza i izvršava samo jedan skup iskaza, koji su pridruženi istinitom izrazu (1 do N), dotle se u konstrukciji switch izračunava samo jedan izraz i izvršava samo jedan skup iskaza, koji je pridružen izračunatoj vrednosti izraza.

switch konstrukcija ima opstu formu :

```

switch (izraz)
{
case vrednost1 :
    iskaz1;
.....

break;
case vrednost2:

```

```

        iskaz2;
        .....
        break;
        .....

        case vrednostN:
            iskazN;
            .....
            break;
        default:
            iskaz0;
            .....
            break
        }
        sledeci_iskaz;

```

Semantika switch iskaza je sledeća: Izračunava se izraz. Dobijena vrednost sukcesivno se upoređuje sa vrednostima vrednost1, ..., vrednostN, koji moraju biti konstante ili konstantni izrazi. Ako je pronađen case, čija je pridružena vrednost vrednosti (i = 1,...,N) jednaka izračunatoj vrednosti izraza izraz, izvršava se iskaz, koji sledi case, tj., iskazl. Treba primetiti da nisu potrebne vitičaste zagrade za objedinjavanje skupa iskaza u programsku celinu, ali cela konstrukcija switch mora biti objedinjena vitičastim zagradama.

Ključna reč break ili drugačije break iskaz, informiše o kraju odgovarajućeg case-a, završavajući switch iskaz. break iskaz se ne sme izostaviti na kraju svakog case-a. U protivnom, izvršavaju se iskazi koji slede, bez obzira što pripadaju drugom case-u.

default se izvršava ako vrednost izraza nije jednaka nijednoj od delu iskaza else-if.

U programu 4.33 koristi se switch iskaz. Program 4.33 realizuje aritmetičke operacije množenja, sabiranja, oduzimanja i deljenja. U switch iskazu kao izraz navedena je promenljiva op, koja predstavlja operator. U case-ovima navedene su 4 moguće vrednosti promenljive op, koja predstavlja operator. Uz svaki case navedeni su pridruženi iskazi. Naveden je i default, koji registruje nepredviđene operatore.

```

/*Program 4.33 */
/* Program za izracunavanje prostih aritmetickih izraza
u formi operand1 operator operand2 */

```

```

main()
{
float operand1, operand2;
char op;
printf("Ukucajle Izraz ? \n");
scanf("%f%c%f", &operand1, &op, &operand2);
switch (op)
{
case '+':
printf("%f\n", operand1+operand2);
break;
case '-':
printf("%f\n", operand1-operand2);

```

```

break;
case '*':
printf("%f\n", operand1*operand2 );
break;
case '/':
printf("%f\n", operand1/operand2);
break;
default:
printf("Nepoznat operator\n");
}
}

```

#### Program 4.33 Izlaz

Ukucajte izraz ?

12.9-2.0

10.900000

Ukucajte izraz ?

12.9&2.0

Nepoznat operator

#### 4.7.12. Iskazi bezuslovnog grananja

U daljem tekstu predstaviceemo nekoliko specijalnih iskaza C jezika, koje treba koristiti u tačno određenim slučajevima. U ostalim slučajevima upotreba ovih iskaza treba da bude dobro dokumentovana, jer se narušava koncept strukturiranog programiranja. To su: break, continue i goto iskazi .

##### 4.7.12.1. Break iskaz

Break iskaz se najčešće pojavljuje u switch iskazima. Međutim, break ima opštiju ulogu u C programima. Ponekad se u programima javlja potreba da se for petlja prekine nakon određenog uslova. U delu programa

```

.....
for (i=0; i<N; + +i)
{
.....
if (test == FALSE)
break;.
.....
}
.....

```

testira se deo nekog uređaja. Kada se detektuje otkaz, for petlja se prekida koristeći break iskaz, break iskaz trenutno završava petlju i kontrola programa prenosi se na iskaze koji slede posle for petlje.

break iskaz se može koristiti u for, while, do-while i switch iskazima. Ako se break koristi u ugnježenim petljama, završava se unutrašnja petlja u kojoj se break nalazi.



Upotreba break iskaza u switch iskazu je prirodna i neophodna. Međutim, u drugim slučajevima upotreba break iskaza nije poželjna, jer zamagljuje programsku logiku i narušava sekvencijalni tok programa.

#### 4.7.12.2. Goto iskaz

Goto iskaz uzrokuje bezuslovni programski skok na specificirano mesto u programu. Identifikacija mesta u programu se definiše labelom. Labela se stavlja neposredno pre iskaza, koji je određišno mesto skoka, i mora biti u istom programu kao i goto iskaz. Na primer, iskaz

```
goto poruka;
```

uzrokuje bezuslovni skok na iskaz identifikovan labelom poruka. Ova labela može biti locirana bilo gde u programu, pre ili posle goto iskaza. Na primer,

```
poruka: printf("Prikaz goto iskaza/n");
```

Neki programeri goto iskaz često koriste kao sredstvo izlaska iz ugnježenih petlji. U delu programa

```
for(i=0;i<N; ++i)
    {
    ....
for (test == FALSE)
    {
    ....
if (test == FALSE)
    goto otkaz;
    ....
    }
    ....
    }
otkaz:
.....
```

posle istinitosti izraza test ==FALSE, izvršava se goto iskaz i kontrola prenosi na iskaz identifikovan labelom otkaz.

goto iskaz treba izbegavati bez obzira što postoji u većini modernih programskih jezika. goto nije savremen metod kontrole toka programa, koji je u modernim jezicima suvišan. Kada iskaz goto treba koristiti? U retkim slučajevima koji moraju biti dobro dokumentovani, goto može povećati efikasnost programa. Takođe, u nekim slučajevima iskaz goto pojednostavljuje kontrolu toka programa.

#### 4.7.12.3. Continue iskaz

Continue iskaz završava iteraciju petlje i startuje sledeću. Drugim rečima, iskazi nakon continue iskaza automatski se preskaču i kontrola prenosi na početak petlje.

continue iskaz može se napisati u for, while i do-while petlji. Kao što sledeći primer pokazuje, continue prenosi kontrolu na kraj tekuće petlje (break završava petlju). Sledeća for petlja sa continue iskazom

```

for ( izraz1 ; izraz2; izraz3)
    {
    ...
    continue;
    ...
    }
    ...

```

može se prikazati ekvivalentnim iskazom

```

izraz1;
while (izraz2)
    {
    ...
    goto sledeci;
    ...
sledeci:
izraz3;
    }

```

#### 4.8. Operator uslovnog izraza

Operator uslovnog izraza je najneobičniji operator u C jeziku. Do sada smo imali unarne ili binarne operatore. Operator uslovnog izraza je ternarni operator, jer zahteva tri operanda. Za označavanje se koriste dva karaktera: znak pitanja '?' i dvotačka ':'. Prvi operand je pre znaka pitanja, drugi između znaka pitanja i dvotačke, a treći posle dvotačke. Opšta forma uslovnog izraza je:

$$\text{uslov} ? \text{izraz1} : \text{izraz2}$$

Uslov je proizvoljan izraz, najčešće relacioni izraz, koji se izračunava prvi. Ako je uslov istinit, izračunava se izraz1 i taj rezultat definiše vrednost celog uslovnog izraza. U protivnom, ako je uslov neistinit, izračunava se izraz2 i taj rezultat određuje vrednost celog uslovnog izraza. Uslovni izraz funkcioniše kao if-else iskaz.

Ovaj operator se najčešće koristi za dodeljivanje promenljivoj jedne od dve vrednosti u zavisnosti od određenog uslova. Na primer, izraz :  $s = (x < 0) ? -1 : x * x$ ; dodeljuje promenljivoj s vrednost -1, ako je uslov x istinit, a vrednost  $x * x$ , ako je uslov  $x > = 0$  istinit. Zgrade se pišu radi čitljivosti i nisu obavezne. Prioritet operatora uslovnog izraza je niži od operatora dodeljivanja vrednosti i relacionog operatora < .

Ako se izraz, koji se koristi posle dvotačke sastoji od drugog uslovnog izraza dobijamo efekat else-if klauzule. Na primer, pogledajmo iskaz

$$s = (\text{broj} < 0) ? -1 : ((\text{broj} = 0) ? 0 : 1);$$

Ako je vrednost promenljive broj manja od 0, promenljivoj s se dodeljuje vrednost -1. U protivnom, izvršava se drugi uslovni izraz. Ako u drugom uslovnom izrazu promenljiva broj ima vrednost 0, promenljiva s dobija vrednost 0. U slučaju da je

vrednost promenljive broj veća od 0, promenljiva s doblja vrednost 1. Zagrade nisu obavezne, zato što je asocijativnost uslovnog operatora s desna na levo.

U C jeziku operator uslovnog izraza tretira se kao svaki drugi izraz ili operator. Sloga. operator uslovnog Izraza može se pisati svuda gde se mogu pisati ostali izrazi i operatori.

Program 4.34 učitava broj i stampa -1, 0 ili 1 u zavisnosti da li je učitani broj manji od 0, jednak 0 ili veći od 0, respektivno. Uslovni izraz je smešten kao argument funkcije printf, čime dobijamo kratak, ali koristan program. Evidentno je da uslovni izraz pojednostavljuje program.

```
/*Program 4.34*/
/* Program za odredjivanje znaka broja */
main()
{
float broj;
printf("Ukucajte broj ? \n");
scanf("%f", &broj);
printf("Znak=%d\n", (broj < 0) ? -1 : (broj == 0) ? 0 : 1);
}
```

**Program 4.34 Izlaz**

Ukucajte broj ? 0

Znak=0

Ukucajte broj ? 12.0

Znak:1

## 4.9. Vektori

C jezik, kao i većina drugih programskih jezika visokog nivoa, obezbeđuje sredstva za deklarisanje skupa homogenih podataka. Ovaj tip podatka naziva se vektor (engl. array). Ovo poglavlje opisuje način za deklarisanje vektora i njihovu upotrebu u C programima. U narednim poglavljima opisana je upotreba vektora u funkcijama, strukturama, nizovima karaktera i ukazateljima.

### 4.9.1. Jednodimenzionalni vektori

Programi često koriste homogene podatke. Na primer, pretpostavimo da hoćemo da napišemo program koji manipuliše ispitnim ocenama studenata i da tih ocena po studentu ima 20. Na osnovu pređenog materijala o C jeziku morali bismo da deklariramo svaku ocenu ponaosob, tj ,

```
int ocena1,....ocena20;
```

Očigledno je, da je ovakav način deklarisanja ocena glomazan. C jezik obezbeđuje mogućnost deklarisanja vektora, koji se definišu kao skup homogenih podataka. Vektore treba razlikovati od struktura, koje predstavljaju skup heterogenih podataka. Drugim rečima, elementi vektora su istog tipa, dok su elementi struktura različitih tipova. Primer sa ocenama možemo jednostavno rešiti deklaracijom.

```
int ocena[20];
```

elemenata vektora, kojih ima 20 (broj između uglastih zagrada). Dakle, deklaracija jednodimenzionalnog vektora je tip podatka (u gornjem slučaju `ini`) praćena identifikatorom sa celobrojnim konstantnim izrazom u uglastim zagradama. Vrednost konstantnog izraza, koja mora biti pozitivna, je veličina vektora i određuje broj elemenata u vektoru. Indeks vektora u C jeziku je u opsegu od 0 do veličine vektora umanjenog za 1. Na primer, u deklaraciji

```
intocena[20];
```

donja granica indeksa je 0, gornja granica indeksa je 19, a veličina vektora je 20.

Razmotrimo selekciju elemenata vektora. Ako je ocena vektor, možemo pisati

```
ocena[izraz]
```

gde je izraz celobrojni pozitivni izraz, kojim selektujemo specificirani element vektora. Uglaste zagrade su operatori u C jeziku i imaju najviši prioritet među ostalim operatorima. Asocijativnost ovog operatora je s leva na desno. Izraz je indeks vektora.

Na osnovu deklaracije `int i, ocena[20];` izraz `ocena[i]` može biti korišćen za pristup elementu vektora dodeljivanjem odgovarajuće vrednosti promenljivoj `i`. Ako promenljiva `i` ima vrednost manju od 0 ili veću ili jednaku veličini vektora, tada se ne pristupa elementima vektora. Efekat ovakve greške je pristupanje slučajnom sadržaju u memorijskim lokacijama, koje ne pripadaju vektoru. C prevodilac ne kontrolise da li je vrednost indeksne promenljive van dozvoljenog opsega. Na primer,

```
for (suma = 0, i = 0; i < 20; ++ i)
    suma += ocena[i];
```

sabira vrednosti elemenata vektora `ocena`, dodavajući vrednost nepostojećeg dvadesetog elementa. Slično, fragment programa

```
for (i = 0; i < 20; ++ i)
    printf("%c%10d", (i%5 == 0) ? "\n": ' ', ocena[i]);
```

štampa vrednosti elemenata vektora `ocena` po 5 elemenata u istoj liniji, uključujući i nepostojeći dvadeseti element.

Program 4.35 je jednostavan primer manipulacije elementima vektora. Elementi vektora `ocena` se postavljaju na rastuće vrednosti, korišćenjem `for` petlje. Narednom `for` petljom štampaju se vrednosti elemenata vektora. Program 4.36 je takođe jednostavan primer manipulacije elementima vektora. U programu se izračunava prvih 20 Fibonačijevih brojeva. Fibonačijevi brojevi su brojevi koji zadovoljavaju uslov da je

$$\text{Fibonacci}[i] = \text{Fibonacci}[i-2] + \text{Fibonacci}[i-1]$$

pri čemu je

$$\text{Fibonacci}[0] = 0$$

$$\text{Fibonacci}[1] = 1$$

```

/*Program 4.35*/
/* Program za stampanje elemenata vektora */
#include<stdio.h>
main()
{
int i, ocena[20];

for (i=0; i<20; ++i)
ocena[i] = i;
for (i=0; i<20; ++i)

printf("%cocena[%2d]=%2d", (i%5== 0)?'\n': ' ', i, ocena [i]);

printf("\n");
}

```

#### Program 4.35 Izlaz

```

ocena[ 0]= 0 ocena[ 1]= 1 ocena[ 2]= 2 ocena[ 3]= 3 ocena[ 4]= 4
ocena[ 5]= 5 ocena[ 6]= 6 ocena[ 7]= 7 ocena[ 8]= 8 ocena[ 9]= 9
ocena[10]=10 ocena[11]=11 ocena[12]=12 ocena[13]=13 ocena[14]=14
ocena[15]=15 ocena[16]=16 ocena[17]=17 ocena[18]=18

```

```

/*Program 4.36*/

/* Program za generisanje prvih 20 Fibonacijevih brojeva */
main()
{
int i, fibonaci[20];
fibonaci[0] = 0;
fibonaci[1] = 1;

for (i=2; i<20; ++i)
fibonaci[i] = fibonaci[i-2] + fibonaci[i-1];

for (i=0; i<20; ++i)
printf("%c%5d", (i%5 == 0) ? '\n' : ' ', fibonaci[i]);

printf("\n");
}

```

### 4.9.2. Višedimenzionalni vektori

C jezik dozvoljava višedimenzionalne vektore proizvoljnog tipa. Na primer, vektor vektora. Sa dva para uglastih zagrada dobijamo dvodimenzionalne vektore. Dodavanjem uglastih zagrada dodajemo po jednu dimenziju. Na primer,

Int x[100] je jednodimenzionalni vektor  
float y[5] [3] je dvodimenzionalni vektor  
double z[5] [3] [7] je trodimenzionalni vektor

K-dimenzionalni vektor ima veličinu za svaku od k dimenzija. Broj elemenata k-dimenzionalnog vektora je jednak proizvodu broja elemenata svih dimenzija. Za vektor z iz gornjeg primera broj elemenata je  $5 \cdot 3 \cdot 7 = 105$ . Svi elementi vektora se memorišu jedan do drugog počevši od početne adrese vektora.

U daljem tekstu opisani su dvodimenzionalni vektori, jer se najčešće koriste. Diskusija o dvodimenzionalnim vektorima važi i za višedimenzionalne vektore. Jedna od najprirodnijih primena dvodimenzionalnih vektora je predstavljanje matrica.

Pogledajmo matricu 3\*4

$$\begin{vmatrix} 13 & -18 & 21 \\ 36 & 101 & 0 \\ -5 & 0 & -9 \\ 3 & 1 & 9 \end{vmatrix}$$

Notacija za označavanje elemenata matrice koristi dvostruke indekse, gde prvi indeks ukazuje na vrstu, a drugi na kolonu. Ako gornju matricu označimo sa M, tada za

označavanje elemenata u drugoj vrsti i drugoj koloni koristimo notaciju  $M(i,j)$ , gde je opseg za indeks vrste i od 1 do 4, a opseg za indeks kolone J od 1 do 3.

U C jeziku koristi se analogna notacija za selekciju elemenata dvodimenzionalnih vektora. Međutim, pošto u C jeziku indeksi počinju od 0, prva vrsta matrice je vrsta 0, a prva kolona je kolona 0. Gornja matrica ima sledeću predstavu u C jeziku

	broj kolone (j)		
	0	1	2
broj vrste (i)			
0	13	-18	21
1	36	101	0
2	-5	0	-9
3	3	1	9

Notacija za označavanje elemenata matrice je  $M[i][j]$

gde je prvi indeks broj vrste, a drugi broj kolone. Na primer, u izrazu

$$\text{suma} = M[0][2] + M[2][2]$$

sabiraju se element u nultoj vrsti i drugoj koloni (=21) i element u drugoj vrsti i drugoj koloni (=9).

Dvodimenzionalni vektori se inicijalizuju na sličan način kao jednodimenzionalni. In-icijalizacioni elementi se navode po vrstama. Vitičaste zagrade se mogu koristiti za odvajanje inicijalizacionih elemenata jedne vrste od druge. Inicijalizacija matrice M je

```
static int M[4][3] =
    {
        {13,-18,21},
        {-5,0,-9},
        {3,1,9},
    };
```

Inicijalizacioni elementi jedne vrste su objedinjeni vitičastim zagradama i odvojeni zarezom od ostalih, izuzev zadnje grupe. Upotreba vitičastih zagrada je opcionalna. Gornja inicijalizacija se može napisati kao

```
static int M[4][3] = {13, -18, 21, 36, 101, 0, -5, 0, -9, 3, 1, 9};
```

Slično jednodimenzionalnim vektorima nije neophodno inicijalizovati sve elemente vektora. U deklaraciji

```
static int M[4][3]=
    {
        {13,18},
        {36,101},
        {-5,0},
        {3,1},
    };
```

inicijalizuju se prva dva elementa u svakoj vrsti. Ostali elementi se inicijalizuju na 0. U ovom slučaju vitičaste zagrade za objedinjavanje inicijalizacionih elemenata pojedinih vrsta su potrebne zbog željene inicijalizacije. Ako se vitičaste zagrade ne navedu inicijalizuju se elementi nulte i prve vrste i prva dva elementa u drugoj vrsti.

U programu 6.6 realizovano je skalarno množenje matrice. Matrica je dimenzija 3\*5 sa elementima tipa float. Skalar kojim se množe elementi matrice je tipa float. Vrednosti elemenata matrice su inicijalizovani u programu. Vrednost skalara se učitava.

```

/*Program 4.37*/
/* Program za skalarno množenje matrica */

main()
{
int vrsta, kol;
float scalar;
static float matrica [3][5]= {
    {7.0,16.83,55.131,13.1,12.91},
    {15.0,10.9,42.99,0.0,7.7},
    {-2.2,1.1,2.2,4.4,9.9}
};

printf("Originalna matrica :\n");
for (vrsta=0; vrsta<3; ++vrsta)
    {
    for(kol=0; kol<5; ++kol)
        printf("%10f",matrica[vrsta][kol]);
    printf("\n");
    }

    printf("\nSkalar ?\n");
    scanf("%f", &scalar);

for (vrsta=0; vrsta<3; ++vrsta)
    for (kol=0; kol<5; ++kol)
        matrica[vrsta] [kol]*= scalar;
printf("\nMatrica posle skalarnog množenja:\n");
for (vrsta=0; vrsta<3; ++vrsta)
    {
    for (kol=0; kol<5; ++kol)
        printf("%10f",matrica[vrsta][kol]);
    printf("\n");
    }
}

```

Program 4.37 Izlaz  
Originalna matrica :

7.000000	16.830000	55.131001	13.100000	12.910000
15.000000	10.900000	42.990002	0.000000	7.700000
-2.200000	1.100000	2.200000	4.400000	9.900000

Skalar ?  
3

Matrica posle skalarnog množenja:

21.000000	50.489981	65.393005	39.300003	38.730000
45.000000	32.699997	128.970001	0.000000	23.099998
-6.600000	3.300000	6.000000	13.200001	29.699999



## 4.10. Strukture, unije i enumerisani tipovi

U ovom poglavlju opisani su novi tipovi podataka (strukture, unije i enumerisani tip) i mogućnost dodeljivanja novih naziva tipova podataka. U predhodnom poglavlju definisali smo vektore kao skup homogenih elemenata, tj. svi elementi vektora su istog tipa (int, char, float, itd.). Strukture u C jeziku su skupovi heterogenih elemenata, odnosno, elemenata različitih ili istih tipova (na primer, jedan element je char, drugi element je int, itd.). Unija se uglavnom koristi u složenijim programskim aplikacijama i omogućava memorisanje različitih tipova podataka u istu memorijsku oblast. Enumerisani tip podataka u formi konačnog skupa određuje vrednosti koje se enumerisanoj promenljivoj mogu dodeliti. Ključna reč `typedef` daje mogućnost definisanja novih naziva postojećih podataka, koji se uslovno mogu smatrati novim tipovima podataka.

### 4.10.1. Strukture

Predpostavimo da u C jeziku imamo zahtev za memorisanjem datuma (rođenja nekih osoba, prodaje neke robe, itd.) u formi dan-mesec-godina. Na osnovu dosadašnjeg poznavanja C jezika ovaj zahtev može se rešiti deklarisanjem tri celobrojne promenljive

```
int dan, mesec, godina;
```

Međutim, promenljive dan, mesec i godina mogu biti logički povezane, npr. ukazuju na datume neke robe. Ovo se može rešiti deklarisanjem tri celobrojne promenljive

```
int dan_prodaje, mesec_prodaje, godina_prodaje;
```

Ovaj način je prihvatljiv, ali glomazan. U C jeziku postoji mogućnost grupisanja logički povezanih promenljivih različitog tipa u jednu celinu, strukturu. Koristeći ključnu reč

`struct` možemo definisati strukturu `datum` sa tri celobrojne komponente, koje predstavljaju dan, mesec i godinu. Ovakva struktura se može deklarirati kao

```
struct datum
{
    int dan;
    int mesec;
    int godina;
};
```

Struktura `datum` sadrži tri elementa tipa `int`: dan, mesec i godina, koji se nazivaju članovima strukture. Ključna reč `struct` u predhodnoj definiciji definiše strukturu, tako da promenljive mogu biti deklarirane kao tip `struct datum`. Na primer, u deklaraciji `struct datum prodaja;` promenljiva `prodaja` se deklarira kao tip `struct datum`. Sintaksa ovakvih deklaracija je ista kao i za osnovne tipove podataka, što znači, da deklaraciju `struct datum danas;` možemo objediniti sa predhodnom u jednu deklaraciju, tj. `struct datum prodaja, danas.`

Selekcija članova strukture razlikuje se od selekcije osnovnih tipova podataka i vektora. Za selekciju članova strukture koristi se operator `.` (tačka), tako što se specificira ime promenljive, operator tačka i ime člana. Operator `.` ima najviši prioritet među operatorima C jeziku i istog je prioriteta sa operatorom selekcije elementa vektora (uglaste zagrade). Asocijativnost ovog operatora je s leva na desno. Selekcija člana strukture ima opštu formu

```
ime_promenljive.ime_člana
```

Promenljivu prodaja možemo postaviti na datum 28-06-1987 sledećim iskazima

```
prodaja.dan=28;
prodaja.mesec=6;
prodaja.godina=1987;
```

Program 4.38 ilustruje predhodnu diskusiju o strukturama. Prvi iskaz u programu 4.38 definiše strukturu datum sa tri promenljive tipa int. Ovaj iskaz samo definiše strukturu datum, pa se memorijski prostor ne dodeljuje. Drugi iskaz deklarise promenljivu prodaja kao tip struct datum, pri čemu se rezerviše memorijski prostor za smeštaj promenljive prodaje. Treći, četvrti i peti iskaz dodeljuje članovima promenljive prodaja vrednosti 28, 6, 1987, respektivno. Zadnji iskaz štampa vrednosti članova dan, mesec i godina promenljive prodaja.

```
/*Program 4.38 */
```

```
/*Program za ilustraciju struktura*/
```

```
main()
{
    struct datum {
        int dan;
        int mesec;
        int godina;
    };
    struct datum prodaja;

    prodaja.dan=28;

    prodaja.mesec=6;
    prodaja.godina=1987;

    printf("Datum prodaje je %d-%d-\'%d\n", prodaja.dan,
        prodaja.mesec,
        prodaja.godina%100);
}
```

**Program 4.38 Izlaz**

**Datum prodaje je 28-6-'87**

### 4.10.2. Strukture i vektori

C jezik praktično nema ograničenja u definisanju složenih tipova podataka. Na primer, strukture mogu biti elementi vektora, vektori mogu biti deklarirani kao članovi struktura, strukture mogu biti članovi struktura, itd. U daljem tekstu opisuju se nekoliko složenih tipova podataka.

#### 4.10.2.1. Vektori struktura

U programu 7.2. definisali smo dve promenljive tipa struct vreme. To su promenljive tekuće vreme i naredno\_vreme. Elegantniji način deklariranja je u kombinovanju vektora i struktura, odnosno, deklarirati navedene promenljive kao elemente vektora. U C jeziku sledeća deklaracija je legalna.

```
struct vreme_vreme[2];
```

Ovom deklaracijom deklariraju se vektor\_vreme od dva elementa tipa struct vreme. Slično,

```
struct datum_datum[10];
```

deklariraju vektor\_datum sa 10 elemenata tipa struct datum. Selekcija članova strukture je sasvim prirodna. Na primer, dodeljivanje vrednosti drugom elementu vektora\_datum je

```
_datum[1].dan=28;
_datum[1].mesec=6;
_datum[1].godina=1987;
```

Inicijalizacija vektora struktura slična je inicijalizaciji višedimenzionalnih vektora. Iskaz

```
static struct datum_datum[10]= {
                                {28, 6, 1987},
                                {28, 1, 1962},
                                {2, 11, 1976}
                                };
```

inicijalizuje prva tri elementa (strukture) vektora\_datum. Unutrašnje vitičaste zagrade su opcionalne, pa je moguće pisati

```
static struct datum datum[10]= {28, 6, 1987, 28, 1, 1962, 2, 11, 1976};
```

Međutim u iskazu

```
static struct datum datum[10]= {
                                {28, 6},
                                {28, 1},
                                {2, 11}
                                };
```

vitičaste zagrade su obavezne, jer se inicijalizuju samo po dva člana struktura u prva tri elementa vektora. Ostali članovi inicijalizuju se na 0.

Program 4.39 je reprogramiran program 4.38 i ilustruje primenu vektora struktura. Promenljiva `tekuce_vreme` je prvi, a naredno\_vreme je drugi element vektora.

```

/*Program 4.39*/
/*Program za korekciju tekuceg vremena*/
main()
{
    struct vreme {
        int sat;
        int minut;
        int sekund;
    } _vreme[2];
    printf("Unesite tekuce vreme? [cc:mm:ss:");
    scanf("%d:%d:%d",&_vreme[0].sat,
        &_vreme[0].minut,
        &_vreme[0].sekund);
    vreme[1]=vreme[0];
    if(++_vreme[1].sekund==60)
    {
        _vreme[1].sekund=0;
        if(++_vreme[1].minut==60)
        {
            _vreme[1].minut=0;
            if(++_vreme[1].sat==24)
                _vreme[1].sat=0;
        }
        printf("Naredno vreme je %02d:%02d:%02d\n",
            vreme[1].sat,vreme[1].minut,_vreme[1].sekund);
    }
}

```

#### Program 4.39 Izlaz

Unesite tekuce vreme? [cc:mm:ss] : 23:59:59

Naredno vreme je 00:00:00

#### 4.10.3. Unija

Jedan od najneobičnijih tipova podataka u C jeziku je unija. Ova konstrukcija uglavnom koristi u složenim programskim aplikacijama, kada je potrebno memorisati različite tipove podataka u istu memorijsku oblast, najčešće sa ciljem štednje memorijskog prostora. Na primer, koristeći svojstvo unije u C jeziku možemo definisati promenljivu `x` koja alternativno memoriše karakter, celi ili realan broj. Iskaz

```

union mesovit_tip
{
    char c;
    float f;
    int i;
};

```

definiše uniju mesovit\_tip. dok iskaz

```
union mesovit_tip x
```

deklariše promenljivu x tipa union mesovit\_tip. Ovakvom deklaracijom se ne definišu tri različita člana unije c, f i i, nego se deklariše x da može da sadrži char ili float ili int vrednost (ali ne sve tri vrednosti u istom trenutku). Prevodilac dodeljuje promenljivoj x memorijski prostor, koji može da memorise najveći specificirani član unije. Notacija za selekciju članova unije je identičan notaciji korišćenju za selekciju članova strukture. Na primer, možemo memorisati char u promenljivu x

```
x.c=&;
```

Iskazom

```
printf(Karakter je %c\n, x.c);
```

možemo štampati memorisani karakter. Sličnom predhodnom, možemo memorisati realan broj

```
x.f=123.4567;           ili celi broj           x.i=123
```

Pošto float, char i int koegzistiraju u istim memorijskim lokacijama, samo jedna vrednost može biti memorisana u x u isto vreme. Program mora voditi računa da format za očitavanje upisane vrednosti bude konzistentan sa tipom upisane vrednosti.

Program 4.40 demonstrira način smeštaja int i float vrednosti u memoriji ukazuje na važnost tretmana memorisane vrednosti u programu sa stvarnim tipom memorisane vrednosti. U programu se prvo emoriše celobrojna vrednost. Pokušaj čitanja memorisane vrednosti koristeći format %f daje pogrešnu vrednost. Takođe, memorisanje realne vrednosti i njen tretman kao celobrojne vrednosti daje pogrešnu vrednost.

```
/*Program 4.40*/
```

```
/*Program za demonstraciju preklapanja int i float tipa u unijama*/
```

```
main()
```

```
{
  union{
    int i;
    float f;
  }x;
  x.i=1234;
  printf("int=%d float=%f\n",x.i, x.f);
  x.f=1234.0;
  printf("int=%d float=%f\n", x.i, x.f);
}
```

Program 4.40 Izlaz

```
int=1234      float=0.000000
int=16384     float=1234.000000
```

Postojanje unije dozvoljava definisanje vektora koji mogu memorisati različite tipove podataka. Na primer, iskaz

```

                struct   {int tip;
union   {int i; double d; char c;} podatak;
                } tabela[5];

```

deklariše vektor tabela čiji su elementi strukture. Član podatak memoriše int ili double ili char. Član tip koristi se kao indikator tipa vrednost u članu podataka. Tako, uvodimo konvenciju da vrednost 1 u članu tip pokazuje da podatak sadrži int, vrednost 2 pokazuje da podatak sadrži double, a vrednost 3 pokazuje da podatak sadrži char. Ova informacija nam pokazuje na koji način tretiramo vrednost memorisanu u članu podatak.

Pri obradi elemenata vektora tabela, na osnovu člana tip određujemo tip memorisane vrednosti u članu podatak. Program 4.41 demonstrira obradu elemenata vektora tabela, štampajući memorisane vrednosti. Kontrola konzistentnosti tipa štampane vrednosti je realizovana iskazom switch, kojim se u zavisnosti od vrednosti člana tip elementa vektora obavlja grananje u programu na ispravnu funkciju printf. Program 4.41 takođe detektuje nepredviđeni tip memorisane vrednosti.

*/\*Program 4.41.\*/*

*/\*Program za stampanje vektora sa  
elementima različitih tipova podataka\*/  
main()*

```

    {
    int k;
    static struct {
        int tip;

union {
        char c;
        int i;
        double d;
        } podatak;
        } tabela[ ]={{2},{1},{3},{4}};
    tabela[0].podatak.d=25.55;
    tabela[1].podatak.i=254;
    tabela[2].podatak.c='Z';
    tabela[3].podatak.d=43.0;
    for (k=0;k<4; ++k)
        switch (tabela[k].tip)
            {
            case 1:
                printf(“%d.element=%d\n”, k, tabela[k].podatak.i);
                break;
            case 2:
                printf(“%d.element=%f\n”, k, tabela[k].podatak.d);
                break;
            case 3:
                printf(“%d.element=%c\n”, k, tabela[k].podatak.c);
                break;

```

```

                                default:
                                printf("Nepoznati tip (%d) u %d.elementu\n", tabela[k].tip, k);
                                break;
                                }
}

```

**Program 4.41 Izlaz**

```

0.element=25.550000
1.element=254
2.element=Z
Nepoznati tip (4) u 3.elementu

```

Ilustrovani primer unije može poslužiti za memorisanje tabele simbola pri prevođenju programa, koja sadrži ime simbola, tip simbola, njegovu vrednost i dr.

**4.10.4. Enumerisani tip podataka**

Ključna reč enum deklariše enumerisani tip podataka, enum omogućava definisanje konačnog skupa vrednosti i deklarisanje enumerisanih promenljivih, koje uzimaju isključivo vrednosti iz tog skupa.\* Na primer, u iskazu

```
enum flag {true, false};
```

definiše se enumerisani tip enum flag. Promenljiva deklarirana kao enum flag uzima vrednosti true ili false i nijednu drugu vrednost. U predhodnoj definiciji ključna reč enum definiše enumerisani tip, a flag je identifikator i predstavlja ime definisanog

enumerisanog tipa. Lista vrednosti, uokvirena vitičastim zagradama, je lista dozvoljenih vrednosti, koji se mogu dodeliti promenljivoj deklariranoj kao tip enum flag.

Deklaracija enumerisane promenljive sastoji se od ključne reči enum, imena enumerisanog tipa, skupa (lista) dozvoljenih vrednosti i liste enumerisanih promenljivih koje se deklarišu.

Dodeljivanje vrednosti enumerisanoj promenljivoj je identično kao i za ostale tipove promenljivih. Iskaz:

```

file_end=false;
i
if (input_end= = true)

```

su legalni. Sledeći primer definiše enumerisani tip enum dani

```
enum dani {ponedeljak, utorak, sreda, cetvrtak, petak, subota, nedelja};
```

Enumerisane vrednosti se tretiraju kao celobrojne konstante. C prevodilac počevši od prvog elementa u listi, dodeljuje sekvencijalne celobrojne vrednosti elementima liste, startujući od 0. Na primer, iskazu

```
ovaj_dan = sreda;
```

\* U mnogim C prevodiocima ne generiše se poruka greške, ako se enumerisanoj promenljivoj dodeli vrednost koja nije navedena u skupu mogućih vrednosti.

gde je promenljiva `ovaj_dan` deklarirana kao tip `enum dani`, promenljivoj `ovaj_dan` dodeljuje se vrednost 2 (ne ime `sreda`). Sekvencijalni način dodeljivanja celobrojnih vrednosti elementima liste može se promeniti eksplicitnim dodeljivanjem željene vrednosti. Program 4.42 koristi enumerisani tip podataka, odgovarajući dan u nedelji u odnosu na učitani i pokazuje povezanost enumerisanog tipa i tipa `int`. Obratite pažnju na `kast` operator u zadnjoj funkciji `printf`. Izrazima `k+=1` i `k%=7` određuje se sledeći dan i sprečava da se promenljivoj `sledeci_dan` dodeli vrednost koja nije navedena u listi dozvoljenih vrednosti.

*/\*Program 4.42\*/*

*/\*Program za određivanje sledeceg dana\*/*

```
main()
{
    int k;
    enum dani {ponedeljak, utorak, sreda, cetvrtak, petak, subota, nedelja} sledeci_dan;
    printf("Unesi danasnji dan:\n");
    printf("1=ponedeljak 2=utorak 3=sreda 4=cetvrtak\n");
    printf("5=petak 6=subota 7=nedelja\n? [1..7] : ");
    scanf("%d", &k);
    k+=1; k%=7;
    sledeci_dan=(enum dani) k;
    printf("\nSledeci dan je %d.dan u nedelji\n",
        (int)sledeci_dan);
}
```

#### Program 4.42 Izlaz

Unesite danasnji dan:

1=ponedeljak 2=utorak 3=sreda 4=cetvrtak

5=petak 6=subota 7=nedelja

? [1..7] : 7

Sledeci dan je 1.dan u nedelji

Pišući program sa enumerisanim promenljivim ne smemo se osloniti na činjenicu da se enumerisane vrednosti tretiraju kao celobrojne konstante. Naime, enumerisane promenljive i promenljive tipa `int` treba tretirati kao različite tipove podataka.

#### 4.10.5. Typedef iskaz

Korišćenjem ključne reči `typedef` C jezik omogućava programeru dodeljivanje alternativnih imena postojećim i identifikovanje novih tipova podataka. Tako u iskazu

```
typedef int BROJAC
```



definiše se ime BROJAC, koje je ekvivalentno ključnoj reči int. Promenljive mogu biti deklarirane BROJAC, tj.

BROJAC j,n

C prevodilac tretira promenljive j i n kao celobrojne promenljive. Osnovna prednost typedef u ovoj slučaju je povećanje čitljivosti programa, s obzirom na namenu promenljivih j i n u programu. Slično tome, iskaz

```
typedef double VEKTOR[20];
```

definiše se VEKTOR, koji je ekvivalentan vektoru tipa double sa 20 elemenata. U kasnijim iskazima VEKTOR može da se koristi kao samostalan tip podataka. Na primer, deklaracija VEKTOR vektor1, vektor2; je ekvivalentna deklaraciji double vektor1[20], vektor2[20];

typedef imena se obično piše velikim slovom, da bi se razlikovala od standardnih tipova podataka. Promenljive deklarirane kao novi tipovi ne tretiraju se drugačije od promenljivih standardnih tipova. Program 4.43 koristi typedef, realizujući množenje kvadratnih matrica. Kao primer, uzete su matrice 3\*3. Prvi deo programa, koristeći tri for petlje, množi matrice m1 i m2, dajući rezultatnu matricu m3. Drugi deo programa 4.43 štampa matricu m3 po vrstama i kolonama.

```
/*Program 4.43*/
/*Program za množenje matrice*/
main() {
    int i, j, k;
    typedef double MATRICA [3] [3];
    static MATRICA m1={1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0},
                m2={9.0,8.0,7.0,6.0,5.0,4.0,3.0,2.0,1.0};
    MATRICA m3;

    /*Množenje dve matrice*/
    for (i=0; i<3; ++i)
        for (j=0; j<3; ++j)
            for (m3[i][j]=0, k=0; k<3; ++k)
                m3[i][j]+=m1[i][k]*m2[k][j];

    /*Stampanje proizvoda zadatih matrica*/
    printf("Proizvod matrica m1*m2\n");
    printf("-----\n");
    for (i=0; i<3; ++i)
        for (j=0; j<3; ++j)
            printf("%15.2f%c", m3[i][j], (j/2 ==1)?'\n' : ' ');
}
```

**Program 4.43 Izlaz**  
**Proizvod matrica m1\*m2**

```
-----
    30.00      24.00      18.00
    84.00      69.00      54.00
    138.00     114.00      90.00
```

### 4.11. Funkcije, domeni i memorijske klase

Složeniji problemi rešavaju se dekompozicijom u veći broj manjih problema. Koristeći mogućnosti funkcija u C jeziku može se svaki manji problem isprogramirati jednom ili sa više funkcija. Funkcije, koje pripadaju jednom problemu, izdvajaju se u zasebni modul ili datoteku. Na ovaj način, dolazimo do pojma modularnog programiranja veći programi mogu se lako pisati, razumevati, testirati, modifikovati i održavati. U dosadašnjim programskim primerima pojavljivale su se sistemske funkcije `printf`, `scanf` i `main`. Funkcija `printf` rešavala je probleme izlazne aktivnosti, `scanf` problem ulazne aktivnosti, dok je `main` informisala sistem o početku izvršnog programa. Funkcije operišu sa programskim promenljivima, čija raspoloživost zavisi od njihovih domena i memorijskih klasa. Ovo poglavlje obrađuje vrlo važne aspekte C jezika: funkcije, domene identifikatora i memorijske klase funkcija i promenljivih.

#### 4.11.1. Definicija funkcije

Funkcija je skup iskaza organizovanih na poseban način. C prevodilac tretira funkciju kao zasebnu programsku celinu. Definicija funkcije sastoji se od zaglavlja i tela. Zaglavlje funkcije sadrži identifikator (ime funkcije), par zagrada (i) sa opcionalnom listom argumenata funkcije i definicije argumenata funkcije. Telo funkcije je skup izvršnih iskaza i deklaracija promenljivih korišćenih u funkciji. Deklaracije promenljivih moraju se pisati pre izvršnih iskaza. Najprostija moguća funkcija u C jeziku je bezefektna funkcija.

```
bezefektna_fja() {}
```

Pogledajmo nešto složeniji primer funkcije, koja štampa poruku Zdravo. Kako ste?.

```
poruka()                               /*Zaglavlje funkcije*/
{                                       /*Telo f-je su svi iskazi izme|u {}*/
    printf("zdravo. Kako ste?\n");
}
```

Identifikator poruke u zaglavlju funkcije je ime funkcije, kojim se funkcija identifikuje i poziva iz funkcije `main` ili iz drugih funkcija. Prazna lista argumenata između zagrada (i) znači da funkcija `poruka` nema argumenata (iz pozivajućeg programa se ne prenosi nijedna vrednost), između vitičastih zagrada je telo funkcije. U funkciji `poruka` telo funkcije sadrži samo jedan iskaz, kojim se poziva funkcija `printf`. Kako se u funkciji ne koriste promenljive, nisu navedene deklaracije promenljivih. Identifikator, odnosno, ime funkcije, zagrade (i) i vitičaste zagrade su obavezni sintaksni elementi u definiciji funkcije. Program 4.44 objedinjuje definisanu funkciju sa glavnim programom u kompletan C program. Program 4.44 sadrži dve funkcije: `poruka` i `main`. U funkciji `main` nalaze se iskaz `poruka()` koji pokazuje da se na tom mestu mora izvršiti funkcija `poruka`. Na tom mestu kontrola programa direktno se prenosi u funkciju `poruka` i

izvršava telo funkcije, kojim se štampa poruka Zdarvo. Kako ste?. Na kraju funkcije, zatvorenom vitičastom zagradom, kontrola programa se vraća u main, na iskaz neposredno nakon iskaza kojim se poziva funkcija poruka. Zagrade (i) u gornjem iskazu kazuju da se u funkciju poruka ne prenose argumenti.

*/\*Program 4.44\*/*

*/\*Program za stampanje poruke\*/*

```
poruka()                                /*fja za stampanje poruke*/
{
    printf("Zdravo. Kako ste?\n");
}

main()
{
    poruka();
}
```

#### Program 4.44 Izlaz

**Zdarvo. Kako ste?**

Kao što se vidi, pozivanje funkcije nije novost. U dosadašnjim primerima koristili smo funkcije printf i scanf. Jedina razlika je što su printf i scanf sistemske funkcije, koje su deo sistemske biblioteke. One postoje u svakom C sistemu, pa se na sreću ne moraju pisati.

Funkcija se može pozivati neograničen broj puta u programu.

Navedeni primeri funkcija su vrlo jednostavni. Program treba pisati kao skup većeg broja manjih funkcija. Jednostavnije je napisati malu funkciju, koja obavlja neku zaokruženu obradu. Tada je testiranje programa jednostavnije. Takođe je jednostavnije održavanje i modifikacija takvih programa. Male funkcije su samodokumentovanije i čitljivije. Korisna poruka za pisanje dobrih programa je pisati funkcije tako da izvorni kod funkcije ne prelazi jednu stranicu programske hartije.

#### 4.11.2. Argumenti funkcija

Funkcija printf zahteva jedan ili više argumenata: konverzioni niz i opcionalne vrednosti koje se štampaju. Ova mogućnost povećava korisnost i fleksibilnost funkcije printf, jer za razliku od funkcije poruka, može štampati proizvoljnu poruku. U sledećem primeru definisana je funkcija suma\_kvadrata, koja kao argument prihvata celobrojnu vrednost n, izračunava sumu kvadrata celih brojeva od 1 do n i štampa izračunatu vrednost. U programu 8.3. main poziva funkciju suma\_kvadrata tri puta za različite vrednosti argumenta n. Funkcija suma\_kvadrata zahteva kratak opis. Prva linija funkcije

suma\_kvadrata (n)

je deklaracija funkcije, koja definiše ime funkcije (identifikator `suma_kvadrata`), broj argumenata funkcije i njihova imena. U ovom slučaju, definiše se samo jedan argument `n`. Argument `n` naziva se formalni argument, jer se koristi samo u definiciji funkcije. Formalni argumenti moraju biti definisati na samom početku funkcije posle deklaracije funkcije i pre prve vitičaste zagrade. U funkciji `suma_kvadrata` to je iskaz `int n`; kojim se formalni argument `n` definiše kao tip `int`.

```

/*Program 4.45.*/
/*Program za izracunavanje sume
kvadrata celih brojeva od 1 do n*/
suma_kvadrata(n)                /*f ja za izracunavanje*/
    int n;                       /*suma kvadrata*/
    {
    int i;
    long suma=0;
    for (i=1; i<=n; suma+=(long)i*i, ++i);
    printf("Suma kvadrata od 1 do %d je %ld\n", n, suma);
    }

main()
{
    suma_kvadrata(10);
    suma_kvadrata(15);
    suma_kvadrata(20);
}

```

#### Program 4.45 Izlaz

Suma kvadrata od 1 do 10 je 385  
Suma kvadrata od 1 do 15 je 1240  
Suma kvadrata od 1 do 20 je 2870

Nakon zaglavlja funkcije definisano je telo funkcije. Kako se želi suma kvadrata brojeva od 1 do `n` potrebne su dve promenljive, i za kontrolu for petlje i suma za agregaciju sume kvadrata u toku izračunavanja. Promenljiva `i` je deklarirana kao `int`, a suma kao `long int`. Deklaracija i inicijalizacija promenljivih je identična i inicijalizaciji promenljivih u `main` funkciji.

Promenljive deklaracije u telu funkcije (promenljive i suma) su automatske lokalne promenljive, jer se automatski "stvaraju" svaki put kada se funkcija poziva i lokalne su u granicama funkcije. Lokalnost u granicama funkcije znači da se lokalnoj promenljivoj ne može pristupiti iz druge funkcije.

`main` iz programa 8.3. poziva funkciji `suma_kvadrata` i kao vrednost formalnog argumenta specificira 10. Argument 10 je stvarni argument funkcije, jer se sa tom vrednošću argumenta funkcija stvarno izvršava. Stvarni argument mora biti istog tipa kao i formalni argument. `main` funkcija poziva funkciju `suma_kvadrata` više puta sa različitim vrednostima stvarnih argumenata.

U programu 4.46 definisana je funkcija `suma_k_stepena`, koja izračunava sumu `k`-tih stepena celih brojeva od 1 do `n`. U funkciji `suma_k_stepena` definiše se dva formalna argumenta: broj `n` do kog se izračunava suma stepenovanih brojeva i `k` koji određuje stepen na koji treba stepenovati brojeve od 1 do `n`. Deklarišu se četiri lokalne promenljive: `i` i `j` tipa `int`, suma i `k_stepen` tipa `long int`. Upotrebom dve for petlje

izračunava se suma k-tih stepena celih brojeva od 1 do n. Zadnja vitičasta zagrada označava kraj funkcije. Rezultat se štampa koristeći funkciju printf. main funkcija poziva funkciju suma\_k\_stepena za različite vrednosti stvarnih argumenata.

*/\*Program 4.46.\*/*

*/\*Program za izracunavanje sume  
k-tih stepena celih brojeva od 1 do n\*/*

```

suma_k_stepena(n, k)                /*fja za izracunavanje*/
    int n, k;                        /*suma k-tih stepena*/
    {
        int i,j,l;
        long k_stepen, suma=0;
        for(i=1; i<=n; suma +=k_stepen, ++i)
            for(k_stepen=1, j=1; j<=k; k_stepen*=(long)i, ++j);
        printf("Suma %d.stepena od 1 do %d je %ld\n", k, n, suma);
    }

main()
{
    suma_k_stepena(10, 2);
    suma_k_stepena(10, 3);
    suma_k_stepena(10, 4);
}

```

**Program 4.46 Izlaz**

```

Suma 2. stepena od 1 do 10 je 385
Suma 3. stepena od 1 do 10 je 3025
Suma 4. stepena od 1 do 10 je 25333

```

#### 4.11.3. Rezultati funkcija

U funkcijama suma\_kvadrata i suma\_k\_stepena obavljaju se specificirana izračunavanja i štampa rezultat izračunavanja. Slično mogućnosti prenosa ulaznih argumenata u funkcije, postoji mogućnost vraćanja rezultata funkcije u pozivajući program (main ili neka druga funkcija). Konstrukcija, koja ovo omogućava je

```
return(izraz);
```

Iskaz return vraća kontrolu programa nazad u pozivajući program sa izračunatom vrednošću izraza izraz. Iskaz return je opcionalan u funkciji. Ukoliko nije naveden funkcija se izvršava do zadnje zatvorene vitičaste zagrade i tek tada se kontrola vraća u pozivajući program. C prevodilac po definiciji vraća vrednost tipa int u pozivajući program. Znači, C prevodilac uvek kada ispred imena funkcije nije specificiran tip povratne vrednosti uzima da je tip povratne vrednosti int. Nije loša programska praksa da se uvek u deklaraciji funkcije zadaje tip povratne vrednosti, jer to

smanjuje mogućnost greške pri kasnijim modifikacijama funkcije. Ako funkcija vraća vrednost, koja nije tipa int, mora se obavezno zadati tip povratne vrednosti u deklaraciji funkcije.

C funkcija vraća samo jednu vrednost koristeći mogućnosti iskaza return. Za razliku od FORTRAN\_a ili PASCAL\_a, C jezik ne pravi razliku između subrutina (procedura) i funkcija. U C jeziku postoji samo funkcija, koja može, ali ne mora, vratiti vrednost. U poglavlju o ukazateljima biće opisan način kako C funkcija može vratiti više od jedne vrednosti (princip prenosa argumenata referisanjem).

Program 4.47. koristi Njutn-Rafsonovu (Newton-Raphson) metodu za aproksimativno izračunavanje kvadratnog korena. U programu 4.47. koriste se dve funkcije: a\_vrednost za izračunavanje apsolutne vrednosti i kvadratni\_koren za izračunavanje kvadratnog korena. Funkcija a\_vrednost vraća vrednost tipa double i zato u deklaraciji funkcije moramo zadati tip double. Funkcija a\_vrednost se koristi u određivanju uslova završetaka while petlje funkcije kvadratni\_koren. Za izračunavanje kvadratnog korena u funkciji kvadratni\_koren koristi se sledeći algoritam:

1. Promenljiva `kk_x` se postavlja na vrednost 1.

2. Ako je apsolutna vrednost izraza  $kk\_x * kk\_x - x$  manja od epsilon, funkcija se završava. U protivnom, promenljiva `kk_x` se postavlja na vrednost  $(x/kk\_x + kk\_x)/2$  i ponavlja se korak 2.

Promenljiva epsilon određuje željenu tačnost izračunatog kvadratnog korena broja `x`. U našem slučaju epsilon je 0.00000001.

*/\*Program 4.47.\*/*

*/\*Program za izracunavanje kvadratnog korena - Newton-Raphson-ov metod\*/*

```
double a_vrednost(x)                /*fja za izracunavanje*/
    double x;                        /*apsolutne vrednosti*/
    {
    return((x<0)?-x : x);
    }
```

```
double kvadratni_koren(x)           /*fja za izracunavanje*/
    double x;                        /*kvadratnog korena*/
    {
    double kk_x=1.0, epsilon=0.00000001;
    while (a_vrednost(kk_x*kk_x - x)>=epsilon)
    kk_x=(x/kk_x+kk_x)/2;
    }
```

```
main()
{
    printf("Kvadratni koren broja 144 je %8f\n",
           kvadratni_koren(144.0));
    printf("Kvadratni koren broja 2 je %8f\n",
           kvadratni_koren(2.0));
}
```

**Program 4.47 Izlaz**

**Kvadratni koren broja 144 je 12.00000000**

**Kvadratni koren broja 2 je 1.41421356**

*Voditi računa o zauzeću memorije !!!!!!!!!!!*

Funkcija se može koristiti u svim slučajevima u kojima je dozvoljen izraz C jezika. Funkcija može biti pozvana iskazom

```
f(lista_argumenata);
```

C prevodilac ne proverava uslove i mesto korišćenja funkcije i činjenicu da li funkcija vraća ili ne vraća vrednost. Iz ovih razloga može doći do situacije da funkcija, koja ne vraća vrednost, bude navedena u izrazu. Ovaj tip greške C prevodilac neće signalizirati. Koristeći ključnu reč void funkcija se eksplicitno deklarira da ne vraća nijednu vrednost (funkcija nema iskaz return ili izraz u iskazu return). Na ovaj način omogućeno je C prevodiocu signaliziranje grešaka, kada se od funkcije, koja realno nema povratnu vrednost, zahteva vrednost. U sledećem programu C prevodilac zahvaljujući ključnoj reči void signalizira sintaksnu grešku.

*main()*

```
{
  int k;
  void print_char();

  k=print_char('x');
}
```

*gde je funkcija print\_char definisano kao*

```
void print_char(c)
  char c;
  {
    printf("%c ima vrednost %d\n", c, c);
  }
```

Ako se kontrola vraća iz funkcije u pozivajući program, koristeći iskaz return bez izraza ili dolaskom na kraj petlje, vrednost koja se tom prilikom eventualno koristi u pozivajućem programu nije definisana. Ako se ovakva vrednost dalje koristi u programu, programer nije o tome upozoren. Na žalost i pored prednosti, mnogi programeri ne koriste tip void u deklaracijama funkcija.

**4.11.4. Vektori kao argumenti funkcija**

Prenos vektora kao argumenata funkcija obuhvata prenos pojedinačnih elemenata i prenos celih vektora (jedno- i višedimenzionalnih).

Prenos pojedinačnih elemenata ne razlikuje se od prenosa promenljivih ili konstanti. Na primer, iskaz za izračunavanje kvadratnog korena  $i+1$ -og elementa vektora  $x$  i dodeljivanje izračunate vrednosti promenljivoj koren je :

```
koren=kvadratni_koren(x[i]);
```

pod pretpostavkom da je tip  $i+1$ -og elemenata jednak tipu formalnog elementa funkcije kvadratni\_koren (=double). U protivnom, ako tip elementa vektora  $x$  ne odgovara tipu formalnog argumenta, neophodna je konverzija u korektan tip.

U funkciji kvadratni\_koren ništa ne treba biti korigovano ili dodato za obradu elemenata vektora  $x$ . Funkcija kvadratni\_koren, prihvata element vektora  $x$  kao i svaki drugi argument tipa double. Vrednost elementa vektora  $x$  se dodeljuje formalnom argumentu u trenutku pozivanja funkcije.

Program 4.48. koristi funkciju kvadratni\_koren. U main funkciji deklarisan je vektor sa elementima tipa int, koji su inicijalizovani u deklaraciji. Funkcija kvadratni\_koren se poziva sukcesivno za svaki element vektora. Operator kast vrednost int se konvertuje u vrednost double, koju kvadratni\_koren prihvata kao argument. U programu 4.48. je evidentno da funkcija kvadratni\_koren prihvata elemente vektora  $x$  kao i svaku drugu vrednost.

*/\*Program 4.48.\*/*

*/\*Program za izracunavanje kvadratnog korena  
- Newton-Raphson-ov metod\*/*

```
double a_vrednost(x)                                /*f ja za izracunavanje*/
    double x;                                       /*apsolutne vrednosti*/
    {
        return((x<0)?-x : x);
    }
double kvadratni_koren(x)                          /*f ja za izracunavanje*/
    double x;                                       /*kvadratnog korena*/
    {
        double kk_x=1.0, epsilon=0.00000001;
        while (a_vrednost(kk_x*kk_x-x)>=epsilon)
            kk_x=(x/kk_x+kk_x)/2;
        return(kk_x);
    }

main()
{
    int i;
    static int x[5]={4, 144, 2, 3, 5};
    for (i=0; i<5; ++i)
        printf("Kvadratni koren broja %d je %8f\n", x[i],
            kvadratni_koren((double) x[i]));
}
```

**Program 4.48 Izlaz**

```
Kvadratni koren broja 4 je 2.00000000
Kvadratni koren broja 144 je 12.00000000
Kvadratni koren broja 2 je 1.41421356
Kvadratni koren broja 3 je 1.73205081
Kvadratni koren broja 5 je 2.23606798
```



## 4.11.5. Strukture kao argumenti funkcija

C prevodioci dozvoljavaju da se strukturalne vrednosti prenose kao argumenti funkcija i vraćaju kao rezultati funkcija. Novije verzije C prevodilaca dozvoljavaju prenošenje celih struktura, s tim što se, za razliku od vektora, prenose vrednošću.

Program 4.49. ispituje način prenosa strukturalnog argumenta. Funkcija `assign1` kao argumente ima strukturalnu vrednost `x` i dve vrednosti, `real` i `imag` tipa `float`. U funkciji `main` strukturalnoj promenljivoj `x` dodeljuje se vrednost 12 i -10.5 i štampaju prvom `printf` funkcijom. U funkciji `assign1` argumentu `x` dodeljuje se vrednost -55 i 66 i štampaju `printf` funkcijom u funkciji `assign1`. Nakon vraćanja iz funkcije `assign1` ponovo se štampaju članovi promenljive `x`. Vrednost članova se nije promenila u funkciji `assign1` i pored eksplicitnog dodeljivanja vrednosti u funkciji `assign1`. Očigledno da se strukturalni argument prenosi vrednošću, jer bi u protivnom po izlasku iz funkcije `assign1` imao vrednosti -55 i 66.

```

/*Program 4.49.*/
/*Program za proveru nacina prenosa
strukturalnih vrednosti*/

typedef struct {
    float re;
    float im;} COMPLEX;

assign1(x, real, imag)                                /*f ja za proveru metoda*/
    COMPLEX x;                                        /*prenosa struktura u f je*/
    float real, imag;
    {
        x.re=real;
        x.im=imag;
        printf("Clanovi u f ji:\n");
        printf("x.re=%f  x.im=%f\n", x.re, x.im);
    }

main()
    {
        COMPLEX x;
        x.re=12.0; x.im=-10.5;
        printf("Clanovi pre pozivanja f je:\n");
        printf("x.re=%f  x.im=%f\n", x.re, x.im);
        assign1(x,-55.0, 66.0);
        printf("Clanovi posle vracanja iz f je:\n");
        printf("x.re=%f  x.im=%f\n", x.re, x.im);
    }

```

## Program 4.49. Izlaz

```

Clanovi pre pozivanja f je:
x.re=12.000000  x.im=-10.500000
Clanovi u f ji:
x.re=-55.000000  x.im=66.000000
Clanovi posle vracanja iz f je:
x.re=12.000000  x.im=-10.500000

```

Program 4.50. realizuje sabiranje i množenje kompleksnih brojeva. Kompleksni brojevi su predstavljeni strukturom COMPLEX u kojoj član *re* predstavlja realan deo, a član *imag* imaginarni deo kompleksnog broja. Struktura COMPLEX je definisana pomoću typedef i napisana je na početku programa van svih funkcija. U ovom slučaju oblast definisanosti strukture COMPLEX je ceo izvorni program. Zato tip COMPLEX možemo koristiti u svim funkcijama. Drugim rečima, struktura COMPLEX je globalna u programu. U drugom slučaju, da smo strukturu definisali u nekoj od funkcija, onda bi ona mogla da se koristi samo u toj funkciji. Ovakva definicija strukture je lokalna u toj funkciji. Za ostale funkcije ova struktura ne bi bila vidljiva (ili definisana), pa bi strukturu COMPLEX morali da definišemo u svakoj funkciji u kojoj se koristi.

```
/*Program 4.50.*/
```

```
/*Program za sabiranje i mnozenje
kompleksnih brojeva*/
```

```
typedef struct {
    float re;
    float im;} COMPLEX;
```

```
COMPLEX assign(real, imag)                /*f ja za dodeljivanje*/
float real, imag;                          /*vrednosti kompleksnim*/
{                                           /*brojevima*/
    COMPLEX z;
```

```
    z.re=real;
    z.im=imag;
    return(z);
}
```

```
COMPLEX add(x, y)                          /*f ja za sabiranje*/
COMPLEX x, y;                              /*kompleksnih brojeva*/
{
    COMPLEX z;
```

```
    z.re=x.re+y.re;
    z.im=x.im+y.im;
    return(z);
}
```

```
COMPLEX multiply(x, y)                     /*f ja za mnozenje*/
COMPLEX x, y;                             /*kompleksnih brojeva*/
{
    COMPLEX z;
```

```
    z.re=x.re*y.re-x.im*y.im;
    z.im=x.re*y.im+x.im*y.re;
    return(z);
}
```

```

main()
{
    COMPLEX x, y, z;

    x=assign(12.0, 14.0);
    y=assign(8.5, -10.5);
    z=add(x, y);
    printf("(%.2f+i*%.2f)+(%.2f+i*%.2f)=%.2f+i*%.2f\n",
           x.re, x.im, y.re, y.im, z.re, z.im);
    z=multiply(x, y);
    printf("(%.2f+i*%.2f)*(%.2f+i*%.2f)=%.2f+i*%.2f\n",
           x.re, x.im, y.re, y.im, z.re, z.im);
}

```

#### Program 4.50. Izlaz

```

12.00+i*(14.00)+(8.50+i*(-10.5))=20.50+i*(3.50)
(12.00+i*(14.00))*(8.50+i*(-10.50))=249.00+i*(-7.00)

```

Funkcija `assign` kao argumente prihvata float vrednosti, dodeljuje članovima promenljive `z` vrednosti `real` i `imag` i tako dobijenu vrednost vraća u `main`. Kako `assign` vraća rezultat tipa `COMPLEX`, u deklaraciji se mora zadati tip `COMPLEX`. Funkcija `add` sabira dva kompleksna broja predstavljena strukturom `COMPLEX`. Argumenti funkcije `add` su vrednosti `x` i `y` tipa `COMPLEX`. Definiše se lokalna promenljiva `z`, koja memoriše zbir vrednosti `x` i `y` po članovima. Funkcija vraća vrednosti tipa `COMPLEX`. Funkcija `multiply` prihvata kao argumente dve strukturalne vrednosti tipa `COMPLEX`, množi realne i imaginarne delove i vraća proizvod tipa `COMPLEX`. Funkcija `assign`, `add` i `multiply` mogu biti realizovane i bez lokalne promenljive `z`.

#### 4.11.6. Rekurzivne funkcije

Algoritmi mogu biti iterativni ili rekurzivni. Iterativne funkcije realizuju iterativne algoritme i korišćene su u dosadašnjim programima više puta, na primer, funkcija `kvadratni_koren` u programu 4.51. C jezik takođe podržava rekurzivne algoritme. Rekurzivne funkcije realizuje rekurzivne algoritme i to su funkcije koje pozivaju same sebe, direktno ili indirektno. Program 4.51. je jednostavan primer rekurzije funkcije `main`, koja se nikada ne završava (da li je baš tako?). Iskaz `main()` u programu 4.51. je rekurzivni poziv iste funkcije `main`.

```

/*Program 4.51.*/
/*Rekurzija f je main*/
main()

{
    printf("Rekurzija f je main\n");
    main()
}

```

#### Program 4.51. Izlaz

```

Rekurzija f je main
Rekurzija f je main

```

Program 4.52. je sledeći jednostavan primer rekurzije za izračunavanje sume pozitivnih brojeva i počiva na rekurzivnom algoritmu da je suma  $n$  pozitivnih celih brojeva jednaka sumi broja  $n$  i sume  $n-1$  predhodnih brojeva:  $S(n)=n+S(n-1)$

Uslov završetka rekurzije  $n \leq 1$ . Funkcija `main` poziva funkciju `suma` za  $n=10$ . Kontrola programa se prenosi u funkciju `suma`. Kako je uslov  $n \leq 1$  neistinit, izvršava se `else` iskaz u čijem izrazu je poziv funkcije `suma`, ali za  $n-1=9$  i tako redom, sve dok se ne ispuní uslov  $n \leq 1$ . U tom momentu ispunjen je uslov za izvršavanje iskaza `return(n)`. Sada se dešava obrnut proces vraćanja u pozivajuće funkcije `suma`, sve do funkcije koja odgovara argumentu  $n=10$ , kada se kontrola vraća u `main`, koja štampa izračunatu sumu. Proce vraćanja u pozivajuće funkcije `suma` je prikazan u tabeli 8.1. Leva kolona sadrži poziv funkcije `suma`, a desna vraćenu vrednost.

poziv funkcije	vraćena vrednost
<code>suma(1)</code>	1
<code>suma(2)</code>	2+suma(1) ili 2+1
...	...
<code>suma(8)</code>	8+suma(7) ili 8+7+6+5+4+3+2+1
<code>suma(9)</code>	9+suma(8) ili 9+8+7+6+5+4+3+2+1
<code>suma(10)</code>	10+suma(9) ili 10+9+8+7+6+5+4+3+2+1

Tabela 8.1. Rekurzivni proces funkcije `suma(n)`

*/\*Program 4.52.\*/*

*/\*Program za rekurzivno izracunavanje  
sume n pozitivnih celih brojeva\*/*

```
int suma(n)                                /*fja za izracunavanje sume*/
{
    int n;
    {                                       /*pozitivnih celih brojeva*/
        if (n<=1)
            return(n);
        else
            return(n+suma(n-1));
    }
}

main()
{
    int n=10;

    printf("Suma brojeva od 1 do %d je %d\n", n, suma(n));
}
```

Program 4.52 Izlaz

Suma brojeva od 1 do 10 je 55

#### 4.11.7. Domeni identifikatora

Domeni ili oblast definisanosti (engl. scope) identifikatora, odnosno, objekata (promenljive, funkcije, konstante, labele i tipove) je područje programa u kojem je identifikator pristupačan. Drugim rečima, u okvirima domena određenog identifikatora može se pristupiti objektu, kojeg identifikator imenuje.

Blokom u C jeziku nazvaćemo složeni iskaz, koji na svom početku sadrži deklaracije promenljivih. Blokovi mogu biti ugnježdjeni, kada spoljašnji blok u potpunosti obihvata unutrašnji, ili paralelni, kada su u potpunosti odvojeni. Blokovi se mogu delimično preklapati. Osnovno pravilo za određivanje domena identifikatora je da je identifikator definisan u bloku u kojem je deklarisan, kao i u svim ugnježdenim lokovima izuzev u blokovima u kojima je deklarisan isti indikator. Tada je spoljašnji identifikator maskiran unutrašnjim. Identifikator je nedefinisan i nepoznat van granica bloka (određene vitičastim zagradama).

Domene identifikatora u raznim slučajevima pokazaćemo na konkretnim primerima.

U programu 4.53. u funkciji main nalaze se dva bloka. U spoljašnjem bloku deklarirana je promenljiva x i inicijalizovana na 11. U unutrašnjem bloku deklarirana je promenljiva pod istim imenom i inicijalizovana na 22. Vrednost promenljive sa imenom x se štampa pre ulaska u unutrašnji blok, pre i posle izlaska iz unutrašnjeg bloka. Na osnovu izlaza programa 4.53. lako se zaključuje da se u prvom i trećem štampanju štampa vrednost spoljašnje promenljive x. U drugom štampanju štampa se vrednost unutrašnje promenljive x. Znači, u unutrašnjem bloku spoljašnja promenljiva x je maskirana unutrašnjom promenljivom sa istim imenom. Posle izlaska iz unutrašnjeg bloka unutrašnja promenljiva x nestaje. Možemo zaključiti da je domen unutrašnje promenljive x unutrašnji blok.

*/\*Program 4.53.\*/*

*/\*Program za proveru domena promenljivih - 2 bloka\*/*

```
main()
{
    /*Spoljasni blok*/
    int x=11;
    printf("Vrednost pre ulaska u un. blok je %d\n", x);
    {
        /*Unutrašnji blok*/
        int x=22;

        printf("Vrednost pre izlaska iz un. bloka je %d\n", x);
    }
    /*Spoljasni blok*/
    printf("Vrednost posle izlaska iz un. bloka je %d\n", x);
}
```

**Program 4.53. Izlaz**

Vrednost pre ulaska u un. blok je 11  
 Vrednost pre izlaska iz un. bloka je 22  
 Vrednost posle izlaska iz un. bloka je 11

#### 4.11.8. Memorijske klase identifikatora

Promenljive i funkcija u C jeziku imaju dva atributa: memorijsku klasu i tip. Memorijska klasa određuje lokaciju i vreme postojanja memorijskog bloka dodeljenog promenljivoj ili funkciji. Tip određuje značenje memorijske vrednosti u rezervisanom bloku. U C jeziku postoje četiri memorijske klase: eksterni, automatski, statički i registarski. Odgovarajuće ključne reči, koje se dodaju deklaraciji ili definiciji su: `extern`, `auto`, `static` i `register`, respektivno.

##### 4.11.8.1. Automatske promenljive

U predhodnom tekstu više puta su pomenute automatske lokalne promenljive. Promenljive deklarisanе na početku bloka su automatske. Nastaju pri ulasku u blok ili funkciju, nestaju kada se blok ili funkcija završe. Deklaracije automatskih promenljivih mogu opcionalno imati ključnu reč `auto`.

Kada se ulazi u blok ili funkciju rezervišе se memorija za automatske promenljive, koja se oslobađa nakon izlaska iz bloka ili funkcije, a vrednost automatskih promenljivih gubi. Znači, domen ili oblast definisanosti automatske promenljive su granice bloka ili funkcije. Pri ponovnom izvršavanju bloka ili funkcije iznova se rezervišе memorija, ali je predhodna vrednost nepoznata.

##### 4.11.8.2. Eksterne promenljive

U dosadašnjim primerima pretpostavili smo da se program nalazi u jednoj datoteci zajedno sa svim funkcijama, koje program poziva, izuzev sistemskih funkcija `printf` i `scanf`. Ovaj prilaz je dobar za male programe. Međutim, u slučaju većih programa ovaj prilaz nije dobar, posebno u uslovima timskog rada. C jezik omogućava, kao i svi moderni jezici, modularno programiranje. Veći program se može sastojati iz više modula ili datoteka. C prevodilac omogućava njihovo odvojeno prevođenje i kasnije objedinjavanje u izvršni program. Primer su funkcije iz sistemske biblioteke. Bez obzira da li se program nalazi u jednoj ili više datoteka neophodna su programska sredstva za komunikaciju između istovremeno ili odvojeno prevedenih modula. Jedan od načina komunikacije između modula, bloka i funkcija je korišćenje globalnih i eksternih promenljivih. Promenljiva je globalna na nivou modula, ako je deklarisanа van svih funkcija u modulu. Memorijski prostor se trajno dodeljuje i njena memorijska klasa je `extern`. Deklaracija globalne promenljive se ne razlikuje od deklaracije bilo koje druge promenljive, globalnost je određena mestom njene deklaracije. Na primer, deklaraciju `int prom=0` stavimo na početak programa, van svih funkcija, onda se promenljivoj `prom` može pristupiti iz svih funkcija u programu. U ovom slučaju, promenljiva `prom` je globalna promenljiva\*. U programu 4.54. demonstrirano je korišćenje globalnih promenljivih, deklarisanih van svih funkcija u modulu, i njihovo maskiranje lokalnim promenljivim. Naime, globalna promenljiva može da bude maskirana lokalnom promenljivom istog imena.

---

\* Funkcije su po definiciji globalni objekti i nikakva dodatna akcija nije potrebna da bi se funkcija deklarovala kao globalna u okviru modula u kojoj je definisana.

```

/*Program 4.54.*/
/*Program za demonstraciju globalnih
i lokalnih promenljivih*/
int x=11, y;
int f()
    {int y=22;
    return(y);}
main()
    {
    printf("x(global)=%d y(global)=%d y(local)=%d\n", x, y, f());
    }

```

**Program 4.54. Izlaz**

x(global)=11 y(global)=0 y(local)=22

**4.11.8.3. Statičke promenljive**

Omogućava lokalnoj promenljivoj da zadrži vrednost kada se završi blok u kome je deklarirana i stvara u sprezi sa globalnom deklaracijom mehanizam za privatnost globalnih promenljivih. Ovo je vrlo važno zbog održanja modularnosti programa.

```

/*Program 4.55.*/
/*program za ilustraciju statickih promenljivih*/
int fun1()
{
static int x=0;
int y=0;
printf("static = %d auto = %d\n",x,y);
++x,++y;
}
main()
{
int i;
for(i=0;i<5;++i)
fun1();
}

```

**Program 8.19 Izlaz:**

```

static = 0 auto = 0
static = 1 auto = 0
static = 2 auto = 0
static = 3 auto = 0
static =4 auto = 0

```

Funkcija main više puta poziva funkciju fun1, koja inkrementira statičku i automatsku promenljivu. Vrednost statičke promenljive se zadržava, dok vrednost automatske promenljive nestaje nakon izlaska iz funkcije fun1.

#### 4.11.8.4. Registarske promenljive

C prevodilac pruža sredstva, koja dozvoljavaju programeru da utiče na efikasnost izvršnog programa. Ako funkcija često koristi određenu promenljivu, može se zahtevati da se vrednost promenljive memoriše u brzim registrima centralne procesorske jedinice (CPU), uvek kada se funkcija izvršava\*. Ovo je moguće ako se ispred deklaracija promenljive stavi ključna reč register. Na primer,

```
register char c;  
register int i;
```

Pošto CPU ima ograničen broj registara, od kojih se mnogi koriste u druge svrhe, promenljiva deklarirana kao registarska postaje automatska, ako nema slobodnih registara ili ih nema uopšte. C prevodila pri tome ne signalizira grešku. Registarska promenljiva treba da bude deklarirana što bliže mestu gde se koristi, što povećava iskorišćenost raspoloživih registara, jer se rezervišu samo onda kada se stvarno koriste. Na primer, u sledećem delu programa promenljiva ili se deklarira kao registarska, neposredno pre korišćenja u for petlji

```
    {  
    register int l;  
    for (i=0; l<5; ++i)  
    {  
    ...  
    }  
  
    ...  
    }
```

Završetak bloka u kome je deklarirana registarska promenljiva oslobađa registar. Registarske promenljive se ponašaju kao automatske promenljive, izuzev što se, ako ima raspoloživih registara, memorišu u registrima.

#### 4.12. Ukazatelji

U ovom poglavlju opisani su ukazatelji (engl. pointers) u C jeziku. Fleksibilnost i mogućnosti u radu sa ukazateljima i pravila ukazateljske aritmetičke čine C jezik znatno moćnijim i pogodnijim za sistematsko programiranje u odnosu na ostale programske jezike, kao što je Pascal. Ukazatelji u C jeziku omogućavaju formiranje složenih dinamičkih struktura podataka (liste, magacini, stabla, redovi i dr.) i s tim u ezi dinamičko dodeljivanje memorije, prenos argumenata funkcije referisanjem, efikasniji rad sa veldorima i prenos funkcija kao argumenata u druge funkcije.

---

\* Kandidati za registarske promenljive su najčešće promenljive za kontrolu petlje i lokalne promenljive u funkcijama.

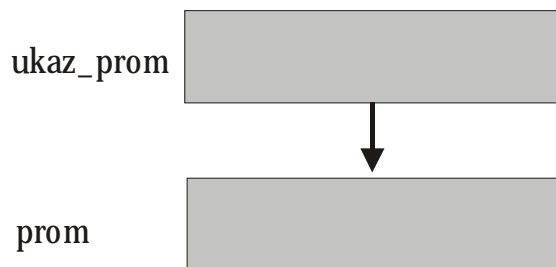


#### 4.12.1. Pojam ukazatelja

Razumevanje rada sa ukazateljima zahteva opis pojma indirekcije. Pojam indirekcije vrlo često koristimo u svakodnevnom životu. Na primer, predpostavimo da hoćete da nabvite traku za svoj štampač. U firmi u kojoj radite sve nabavke se vrše preko nabavne službe. Znači, morate zvati Peru Perića iz nabavne službe i dati zahtev za štampač. Na kraju ovog procesa dobijate tako željenu traku. Način na koji ste nabavili traku je indirektan, jer niste direktno kontaktirali sa lakolanim snabdevačem. Slično se definiše indirekcija u C jeziku. Proizvoljna programska promenljiva memorisana je u određenom memorijskom bloku, na određenoj programskoj lokaciji ili adresi. Ukazatelji omogućavaju indirektni način za pristupanje programskim promenljivim, korišćenjem i manipulacijom adresama promenljivih. Predpostavimo da postoji sledeća deklaracija celobrojne promenljive prom

```
int prom=0;
```

U C jeziku možemo deklarirati promenljivu, ukaz\_prom kao int\*ukaz\_prom; koja omogućava da se indirektno pristupi vrednosti promenljive prom. Karakter zvezdica '\*' deklarira promenljivu ukaz\_prom kao ukazatelj, a tip int da je promenljiva ukaz\_prom ukazatelj na celobrojnu promenljivu. Znači, kombinacija int\* deklarira da je promenljiva ukaz\_prom tipa "ukazatelj na tip int", označavajući da će u programu biti korišćena za indirektan pristup celobrojnim promenljivim.



Slika 9.1. Ukazatelj na int.

Adresni operator & je korišćen u funkciji scanf za prenos učitane vrednosti u pozivajuću funkciju main. Ako je x promenljiva proizvoljnog tipa, tada je vrednost izraza &x adresa promenljive x. Vrednost izraza &x može biti dodeljena promenljivoj, koja je deklarirana kao ukazatelj na isti tip kao što je x. Za promenljive prom i ukaz\_prom možemo pisati iskaz

```
ukaz-prom=&prom;
```

koji promenljivoj ukaz\_prom dodeljuje adresu promenljive prom. Na slici 9.1. prikazan je odnos ukazatelja i promenljive na koju se ukazuje. Usmerena linija pokazuje da ukaz\_prom ne sadrži direktno vrednost prom, nego ukazatelj na prom.

Adresni operator & je unaran istog prioriteta kao ostali unarni operatori. Asocijativnost operatora & je s desna na levo. Opseg vrednosti, koje se mogu dodeliti ukazateljskoj promenljivoj je 0 i skup pozitivnih celih brojeva, koje računarski sistem može interpretirati kao memorijske adrese. Na primer, celobrojnom ukazatelju ukaz\_prom mogu se dodeliti vrednosti

```
ukaz_prom=0;
ukaz_prom=(int*) 15320;
```

U prvom primeru ukaz\_prom dobija vrednost 0. U drugom primeru, operator kast je potreban, jer C prevodilac vrednost 15320 tretira kao celobrojnu konstantu. U protivnom, C prevodilac signalizira nepodudarnost tipova podataka.

Operator indirekcije \* omogućava indirektno pristupanje promenljivoj koristeći ukazatelj na tu promenljivu. Ako je x deklarirana kao int, tada se iskazom

```
x=*ukaz_prom;
```

promenljivoj x dodeljuje vrednost promenljive na koju ukazatelj ukaz\_prom ukazuje. Pošto je promenljiva ukaz\_prom u ranijem iskazu postavljena na vrednost &prom, efekat predhodnog iskaza je dodeljivanje promenljivoj x vrednost promenljive prom

(=5). Na ovaj način promenljivoj prom se pristupa indirektno preko ukazatelja ukaz\_prom. Direktna vrednost ukaz\_prom je adresa promenljive prom, dok je \*ukaz\_prom indirektna vrednost prom, tj. vrednost na adresi sadržanoj u ukaz\_prom.

Operatori & i \* su ilustrovani u programu 4.56. Program 4.56. pokazuje inverznost operatora & i \*. Promenljive prom i x su deklarirane kao int, dok je promenljiva ukaz\_prom deklarirana kao ukazatelj na tip int. Adresnim operatorom dodeljuje se adresa promenljive prom. Operator indirekcije promenljivoj x indirektno dodeljuje vrednost promenljive na koju ukaz\_prom ukazuje, a to je promenljiva prom (=5). Na osnovu izlaza programa 4.56. očigledno je da su operatori \* i & inverzni.

```
/*Program 4.56.*/
```

```
/*Program za ilustraciju adresnog i indirektnog operatora*/
```

```
main()
{
    int x, prom=5, *ukaz_prom;
    ukaz_prom=&prom;
    x=*ukaz_prom;
    printf("prom=%d\nx=%d\n", prom, x);
}
```

**Program 4.56. Izlaz**

```
prom=5
x=5
```

Adresni i indirektni operator mogu se naći u izrazima ravnopravno sa ostalim operatorima C jezika. Program 4.57. ilustruje primenu ovih operatora u jednostavnim aritmetičkim izrazima. Prioritet operatora \* i & je viši u odnosu na binarne aritmetičke operatore.

---

```
/*Program 4.57.*/
```

```
/*Program za ilustraciju primene
operatora * i & u aritmetickim izrazima*/
```

```
main()
{
    int x=5, y=10, z, *ux=&x, *uy=&y, *uz;
    printf("x=%d y=%d\n\n", x, y);
    printf("***& ux=%d\n",
        **& ux);
    printf("5*-* ux/* uy+7=%d\n",
        5*-* ux / * uy+7);
    printf("**(uz=&z)=* ux** uy=%d\n",
        *(uz=&z)=* ux** uy);
}
```

**Program 4.57. Izlaz**

```
x=5 y=10

**&ux=5
5*-*ux/*uy+7=5
*(uz=&z)=*ux**uy=50
```

Adresni operator se ne može primeniti na sve objekte C jezika. Operator & se ne može primeniti na:

1. konstante. Izraz &100 je ilegalan. Dodeljivanje apsolutne adrese (na primer, 2356) može se izvršiti operatorom kast;
2. ime vektora. Ime vektora je konstanta i predstavlja adresu vektora. Stoga, ako je a vektor, tada je &a ilegalno;
3. izraze C jezika. Na primer, &(x+100) je ilegalno;
4. registarske promenljive. Ako je v deklarisana kao registarska promenljiva, tada je izraz &v ilegalan.

Adresni operator se može primeniti na promenljive i elemente vektora. Ako je a vektor, tada su izrazi &a[10] i &a[i+j] legalni.

#### 4.12.2. Ukazatelji i strukture

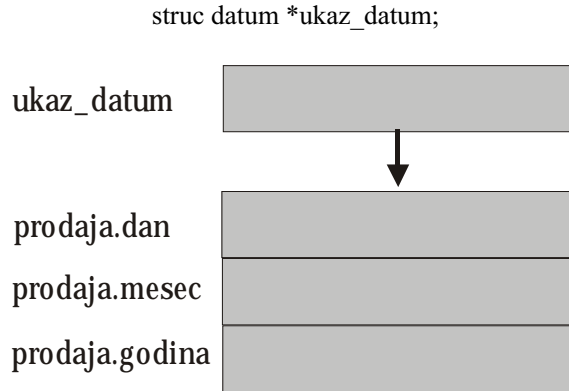
U poglavlju o strukturama definisana je struktura datum sa tri celobrojna člana: dan, mesec i godina.

```
struct datum{int dan; int mesec; int godina;}
```

Struktuirana promenljiva deklarise se iskazom:

```
struc datum prodaje;
```

dok se ukazatelj na strukturu datum deklarise iskazom:



Slika 9.2. Ukazatelj na strukturu

Promenljiva `ukaz_datum` je ukazatelj na tip `struc datum`. Na primer, postavljanje ukazatelja `ukaz_datum` na adresu astrukturirane promenljive prodaje je sledeće

```
ukaz_datum=&prodaja;
```

Posle ovakvog postavljanja ukazateljske promenljive `ukaz_datum`, strukturiranoj promenljivoj `prodaja` se može pristupiti indirektno, posredstvom ukazatelja `ukaz_datum`. Na slici 9.2. prikazana je povezanost ukazatelja `ukaz_datum` na strukturu `struc datum` i promenljive `prodaja`, takođe tipa `struc datum`. Selekcija člana `godina` u strukturiranoj promenljivoj na koju ukazuje `ukaz_datum` je moguće sledećim iskazu

```
(*ukaz_datum) godina=1987;
```

Zagrade `'( )'` su neophodne, jer je operator višeg prioriteta u odnosu na operator `*`. Predhodni iskaz je ekvivalentan iskazu

```
prodaja godina=1987;
```

kojim se član `godina` promenljive `prodaja` postavlja na vrednost 1987. Umesto iskaza

```
(*ukaz_datum) godina=1987;
```

koji je glomazan zbog upotrebe zagrada, u C jeziku postoji specijalni operator za indirektnu selekciju članova strukture. To je operator `->`. Predhodni iskaz je ekvivalentan iskazu

```
ukaz_datum->godina=1987;
```

Operator `->` je operator selekcije članova strukture ukazateljima na strukturu. Sastoji se od dva karaktera `'-'` i `'>'`, koji se ne smeju razdvajati. Ima najviši prioritet među operatorima C jezika, u rangu je sa operatorima pozivanja funkcije, selekcije elemenata vektora i selekcije člana strukture imenom strukture. Asocijativnost ovog operatora je sa leva na desno.

Program 4.58. ilustruje primenu ukazatelja na strukturu. Ukazateljska promenljiva `ukaz_datum` postavlja se na adresu promenljive `prodaja`. Članovi `dan`, `mesec` i `godina`, korišćenjem ukazatelja i operatora `->` postavljaju se na vrednosti 28, 6 i 1987, respektivno, a funkcija `printf` štampa njihove vrednosti.

```

/*Program 4.58.*/
/*Program za ilustraciju ukazatelja na strukturu*/
main()
{
    struct {
        int dan;
        int mesec;
        int godina;
    } prodaja, *ukaz_prodaja=&prodaja;

    ukaz_prodaja->dan=28;
    ukaz_prodaja->mesec=6;
    ukaz_prodaja->godina=1987;
    printf("Datum prodaje je %d-%02d-\'%d\n",
        (*ukaz_prodaja).dan,
        (*ukaz_prodaja).mesec,
        (*ukaz_prodaja).godina%100);
}

```

Program 4.58. Izlaz

Datum prodaje je 28-06-'87

### 4.12.3. Dinamičke strukture podataka

Samoreferisane strukture koriste se u definiciji složenih dinamičkih struktura podataka. Za razliku od vektora ili skalarnih promenljivih, kojima se implicitno dodeljuje memorija prilikom njihovog stvaranja i oslobađanja prilikom brisanja, što je nevidljivo za programera, dinamičke strukture zahtevaju eksplicitno dodeljivanje i oslobađanje memorije.

#### 4.12.3.1. Funkcije malloc, calloc, free i realloc

Dinamičke strukture zahtevaju eksplicitno rezervisanje i oslobađanje memorije u sistemskoj biblioteci svakog C prevodioca postoje funkcije malloc, calloc, realloc i free, kojima se rezervišu i oslobađaju memorijski blok. Definicije ovih funkcija je sledeća:

1. char\*malloc(size). Rezervišu memorijski blok veličine size bajtova, koji se inicijalizuje na 0. Argumenti size je unsigned. U slučaju uspešne rezervacije malloc vraća ukazatelj na tip char, koji ukazuje na rezervisani memorijski blok. U protivnom, vraća 0.

2. char\*calloc(n, size). Rezervišu memorijski blok dovoljan za memorisanje n elemenata svaki veličine size bajtova, znači n\*size. Argumenti n i size su size. Rezervisan memorijski blok je inicijalizovan na 0. U slučaju uspešne rezervacije calloc vraća ukazatelj na char, koji ukazuje na rezervisani memorijski blok. U protivnom, vraća 0.

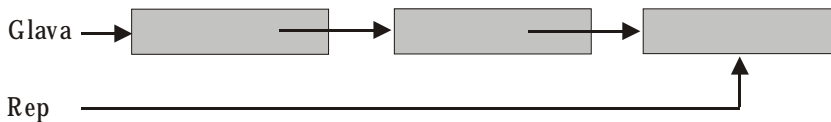
3. void free(ukaz). Oslobađa memorijski blok, koji je rezervisan funkcijama calloc ili malloc. Argument ukaz je ukazatelj na char, koji ukazuje na memorijski blok za oslobađanje. Funkcija free ne vraća nikakvu vrednost.

4. char\*realloc(ukaz, size). Oslobađa rezervisani memorijski blok i rezervišu novi veličine size bajtova. Argument ukaz je ukazatelj na char i definiše memorijski blok, koji se realocira. Argument size je unsigned i određuje veličinu realociranog memorijskog bloka. Argument size je unsigned i određuje veličinu realociranog memorijskog bloka. Ako je realociranje uspešno realloc vraća ukazatelj na char, koji ukazuje na memorijski blok. U protivnom, realloc vraća 0.

Funkcija `malloc` se poziva još dva puta, s tim što vraćeni ukazatelj memorišemo u predhodno rezervisani blok. Višestruko primenom operatora indirekcije pristupamo članovima dinamičkih struktura. Funkcija `printf` štampa član `fakt` u svim rezervisanim blokovima. Na karju, funkcija `free` oslobađa rezervisane memorijske blokove, pri čemu je redosled oslobađanja vrlo bitan, da ne bi izgubili informaciju o rezervisanom bloku.

#### 4.12.3.2. Redovi

Red (engl. queue) je dinamička struktura koja se identifikuje sa dva ukazatelja: glavom, koja ukazuje na prvi element reda, i repom (engl. tail, rear), koji ukazuje na zadnji element reda. Ukazatelji u elementima ukazuju na naredni element u redu, a ukazatelj zadnjeg elementa postavljen je na 0. U slučaju praznog reda glava i rep se postavljaju na 0. Slika 9.6. pokazuje red sa tri elementa.



Slika 9.6. Red

Operacije na redovima su sledeće:

1. unošenje u red na strani repa (engl. `add_queue`),
2. brisanje iz reda sa strane glave reda (engl. `delete_queue`),
3. čitanje prvog elementa na strani glave (engl. `read_queue`).

Ovako definisane operacije omogućavaju da prvi unet element bude prvi uzet ili ispisan iz reda (engl. first-in first-out ili FIFO). Za definisanje funkcija, koje reallizuju operacije na redovima, koristićemo strukturu `QUEUE`, koja se definiše kao

```

typedef struct queue{
    DATA fakt;
    struct queue *sled;
} QUEUE;
  
```

Radi preglednosti definiše se i tip ukazatelja na tip `QUEUE` kao

```

typedef QUEUE *HEAD;
  
```

Glava i rep reda definišu se kao vektor sa dva elementa. Elementi vektora su tipa `HEAD`, odnosno, ukazatelji na strukturu `QUEUE`, pri čemu je prvi element glava, a drugi element rep reda.

Elementi vektora inicijalizuju se na 0. Znači, glava i rep deklarišu se iskazom

```

static HEAD glava[2]={0, 0};
  
```

Tip podataka, koji je memorisan u elementu reda (član `fakt`) je opšteg tipa `DATA`, koji se u ovom slučaju definiše kao `int`, tj.

```
typedef int DATA
```

Funkcije `add_queue`, `delete_queue` i `read_queue` realizuje operacije unošenja, brisanja i čitanja reda.

Funkcija `add_queue` unosi element u red sa strane repa. Argument `queue` nosi informaciju o glavi i repu reda, a `x` je podataka koji se upisuje u element reda. Funkcija `malloc` rezerviše memorijski blok za smeštanje elemenata. U if iskazu ispisuje se da li je red prazan ili ne, i ukoliko je prazan ažurira se glava i rep, u protivnom, ažurira se samo rep reda.

```
void add_queue(queue, x)
    HEAD queue[];
    {
        HEAD pu;
        char *malloc();
        pu=(HEAD) malloc(sizeof(Queue));
        pu->fakt=x;
        pu->sled=0;
        if(queue[1]!=0)
            {
                queue[0]=pu;
                queue[1]=pu;
            }
        else
            {
                queue[1]->sled=pu;
                queue[1]=pu;
            }
    }
```

Funkcija `delete_queue` briše prvi element iz reda. Ako je red prazan funkcija vraća 0, u protivnom, vraća vrednost 1. Zadnji if iskaz je neophodan zbog postavljanja repa reda, kada red postaje prazan posle udaljavanja zadnjeg elementa.

```
int delete_queue(queue)
    HEAD queue[];
    {
        HEAD pu;
        if (queue[0]!=0)
            return(0);
        pu=queue[0];
        queue[0]=queue[0]->sled;
        free(pu);
        if (queue[0]!=0)
            queue[1]=0;
        return(1);
    }
```

Funkcija `read_queue` vraća ukazatelj na prvi element u redu, u protivnom, vraća 0.

```

HEAD read_queue(queue)
    HEAD queue[];
    {
        return(queue[0]);
    }

```

Program 4.59. ilustruje primenu funkcija `add_queue`, `delete_queue` i `read_queue` koristeći jednostavan test model. Funkcijom `add_queue` formira se red tri elementa. Koristeći `do-while` iskazom i funkcijom `read_queue` čita se prvi element reda, a zatim funkcijom `delete_queue` briše se iz reda uz štampanje odgovarajuće poruke. U programu 4.59. koriste se već definisani i objašnjeni tipovi podataka, strukture i ukazatelji.

```

/*Program 4.59.*/
/*Program za ilustraciju operacija na redovima*/
typedef int DATA;
typedef struct queue {
    DATA fakt;
    struct queue *sled;
}QUEUE;
typedef QUEUE *HEAD;

/*Prepisati funkciju add_queue*/
/*Prepisati funkciju delete_queue*/
/*Prepisati funkciju read_queue*/
main()
{
    static HEAD glava[2]={0, 0};
    static int x[]={1, 2, 3, 4};
    int i;
    for (i=0; i<4; ++i)
    {
        add_queue(glava, x[i]);
        printf("Vrednost %d u red\n", x[i]);
    }
    for (i=0; i<4; ++i)
    {
        printf("Vrednost %d iz reda\n", read_queue(glava)->takt);
        delete_queue(glava);
    }
}

```

#### Program 4.59 Izlaz

```

Vrednost 1 u red
Vrednost 2 u red
Vrednost 3 u red
Vrednost 4 u red
Vrednost 1 iz reda
Vrednost 2 iz reda
Vrednost 3 iz reda
Vrednost 4 iz reda

```



### 4.12.3.3. Magacini

Magacin (engl. stack) je dinamička struktura, koja se u C jeziku realizuje pomoću povezane linearne liste. Elementi magacina su istog tipa, a njihov broj u magacinu je promenljiv. Operacije na magacinu zasniva se na operisanju sa elementom na vrhu magacina. Element se može upisati u magacin ili uzeti iz magacina, isključivo sa vrha, na koji ukazuje specijalni ukazatelj, glava magacina. Magacin radi na principu poslednji upisan, prvi se uzima (engl. last-in first-out ili LIFO). Na slici 9.7. magacin je prikazan grafički. Operacije, koje se definišu na magacinu su:

1. čitanje magacina (engl. read),
2. upis u magacin (engl. push),
3. ispis iz magacina (engl. pop).

Elementi magacina, pored ukazatelja na sledeći element u magacinu, sadrže korisne podatke. Tip tih podataka može biti proizvoljan. U ovom slučaju tip podataka je DATA, koji se definiše kao

```
typedef int DATA
```

Tip DATA se koristi pri definisanju strukture elemenata magacina. Struktura elemenata magacina definiše se kao

```
typedef struct stack      {
    DATA fakt;
    struct stack *sled;
} STACK;
```

Radi preglednosti funkcija i programa definiše se tip ukazatelja na strukturu STACK

```
typedef STACK *TOP;
```

Program 9.11. ilustruje primenu funkcija, koje realizuju operacije na magacinu.



Slika 9.7. Magacin

Funkcija `read_stack` realizuje operaciju čitanja vrha magacina, pri čemu se vrsni element ne briše iz magacina. `read_stack` zahteva ukazatelj na glavu magacina, a vraća vrednosti tipa DATA, koja je memorisana u elementu na vrhu magacina.

Funkcija `push` realizuje operaciju upisa u magacin. `push` kao argumente zahteva ukazatelj na glavu magacina (argument `top`) i vrednost tipa DATA, koja treba da se memoriše u magacinu (argument `x`). `push` ne vraća nikakvu vrednost (tip `void`).

Funkcija `pop` realizuje operaciju čitanja vrha magacina, pri čemu se briše vrsni element magacina. `pop` kao argument zahteva ukazatelj na glavu magacina, a vraća vrednost tipa DATA, koja je bila memorisana u prvom elementu magacina.

```

/*Program 4.60.*/
/*Program za ilustraciju operacija na magacinu*/
typedef int DATA;
typedef struct stack {
    DATA fakt;
    struct stack *sled;
} STACK;
typedef STACK *TOP;

void push(top, x)                /*F ja za upis u magacin*/
    TOP *top;
    DATA x;
    {
        TOP prom; char *malloc();

        prom=(TOP) malloc(sizeof(STACK));
        prom->fakt=x; prom->sled=*top; *top=prom;
    }

DATA pop(top)                    /*F ja za destruktivno*/
    TOP *top;                    /*citanje magacina*/
    {
        TOP pu; DATA x;
        if (*top==0)
            return(0);
        x=(*top)->fakt; pu=*top;
        *top=(*top)->sled; free(pu);
        return(x);
    }

DATA read_stack(top)            /*F ja za nedestruktivno*/
    TOP *top;                    /*citanje magacina*/
    {
        return((*top)->fakt);
    }

main()
    {
        TOP magacin=0;
        static DATA x[]={111, 222, 333};
        int i;
        for(i=0; i<3; ++i)
            {
                push(&magacin, x[i]);
                printf("push=%d\n", read_stack(&magacin));
            }
        for(i=0; i<3; ++i)
            printf("pop=%d\n", pop(&magacin));
        if (magacin==0)
            printf("Magacin prazan\n");
        else
            printf("Magacin nije prazan\n");
    }

```

**Program 4.60 Izlaz**

```
push=111
push=222
push=333
pop=333
pop=222
pop=111
```

Magacin prazan

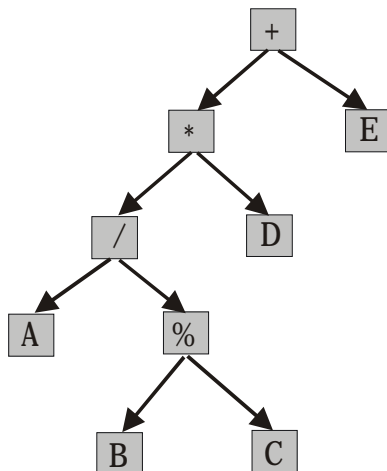
**4.12.3.4. Binarna stabla**

Stablo je konačan skup elemenata, nazvanih čvorovima, takav da: (1) postoji specijalno označen čvor nazvan koren; (2) ostali čvorovi su podeljeni u  $n \geq 0$  disjunktivnih skupova  $T_1, \dots, T_n$  od kojih je svaki stablo.  $T_1, \dots, T_n$  nazivaju se podstabla korena. Čvor, koji nema podstabla, naziva se list.

Binarno stablo je stablo u kome svaki čvor ima najviše dva podstabla. Stoga, binarno stablo možemo definisati kao konačan skup čvorova, koji može biti prazan ili se sastoji od korena i dva disjunktta binarna stabla, nazvanih levo i desno podstablo. Na slici 9.8. prikazano je binarno stablo sa aritmetičkim izrazom.

Postoji mnogo operacija, koje se mogu definisati na stablima. Jedno od najčešćih i najvažnijih je obilazak stabla, pri kojoj se svaki čvor obilazi tačno jedanput. Mogu se definisati tri načina za obilazak stabla u zavisnosti od redosleda obilaska korena, levog i desnog podstabla:

1. inorder redosled, pri kojem se obilazi redom levo podtablo, koren, desno podstablo,
2. preorder redosled, pri kojem se obilazi redom koren, levo podstablo, desno podstablo,
3. postorder redosled, pri kojem se obilazi redom levo podstablo, desno podstablo, koren.



Slika 9.8. Binarno stablo sa aritmetičkim izrazom

Pri definiciji funkcija u C jeziku, koje realizuju pojedine operacije na binarnom stablu, korišćemo sledeće strukture i tipove

```
typedef char DATA;
typedef struct node {
    DATA fakt;
    struct node *levo;
    struct node *desno;
} NODE;
typedef NODE *TREE
```

Tip DATA definiše tip podataka memorisan u čvoru stabla. Svaki čvor je tipa NODE. NODE je struktura u kojoj član fakt memoriše koristan podatak u čvoru, a čvorovi levo i desno su ukazatelji na levo i desno podstablo čvora. Tip TREE je ukazatelj NODE.

Funkcija inorder rekurzivno obilazi binarno stablo. Argument je ukazatelj na koren stabla

```
void inorder(koren)
    TREE koren;
    {
        if (koren !=0)
            {
                inorder(koren->levo);
                printf("%c", koren->fakt);
                inorder(koren->desno);
            }
    }
```

Primenom funkcije inorder na stablo sa slike 9.8. dobija se sledeći niz

A/B%C\*D+E

što predstavlja infiksnu formu aritmetičkog izraza.

Funkcija preorder se dobija izmenom redosleda iskaza u if konstrukciji funkcije inorder. Definicija funkcije preorder je

```
void preorder(koren)
    TREE koren;
    {
        if (koren !=0)
            {
                printf("%c", koren->fakt);
                preorder(koren->levo);
                inorder(koren->desno);
            }
    }
```

Primenom funkcije preorder na stablo sa slike 9.8. dobija se niz

+\*/A%B C D E

što predstavlja prefiksnu formu aritmetičkog izraza.

Slično, funkcija postorder se definiše kao

```
void postorder(koren)
    TREE koren;
    {
    if (koren !=0)
        {
        preorder(koren->levo);
        inorder(koren->desno);
        printf("%c", koren->fakt);
        }
    }
```

Funkcija postorder, primenjena na stablo sa slike 9.8. štampa podstfiksnu formu aritmetičkog izraza, tj.

A B C % / D \* E +

#### 4.12.4. Ukazatelji i vektori

Ukazatelji na vektore imaju specifičan tretman u C jeziku. Operacija, koja se izvodi indeksiranjem vektora, može takođe biti izvedena korišćenjem ukazatelja. Međutim, operacija izvedene ukazateljima su u opštem slučaju brže, ali teže za razumevanje. Ukazatelj je promenljiva, čije su vrednosti adrese. Ime vektora je takođe adresa, ali kontaktna ili fiksna. Kada se vektor deklariraše, C prevodilac rezerviše

memorijski blok, dovoljan za smeštanje svih elemenata vektora. Početna adresa rezervisanog bloka je ime vektora, simbolički predstavljamo imenom u deklaraciji vektora. Početna adresa bloka je istovremeno adresa prvog elementa vektora. Iskaz

```
int x[10]
```

deklariše vektor od 10 celobrojnih elemenata. C prevodilac rezerviše memorijski blok, dovoljan za smeštaj 10 celobrojnih vrednosti. Ako je promenljiva pa deklarisana kao `int *pa;` tada su izrazi `pa=&a[0];` i `pa=a;` ekvivalentni. Na osnovu pravila ukazateljske aritmetike, iskazi `pa=&a[1];` i `pa=a+1;` su takođe ekvivalentni.

Povezanost ukazatelja i višedimenzionalnih vektora pokazaćemo na primeru dvodimenzionalnih vektora. Na primer, deklarisaćemo vektor x kao

```
int x[3][4];
```

Dvodimenzionalni vektor se može predstaviti u obliku matrice

	kolone			
vrste	1	2	3	4
1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

#### 4.12.4.1. Ukazateljska aritmetika

Ukazateljska ili adresna aritmetika je vrlo jednostavna, ali vrlo korisna i predstavlja veliku prednost C jezika u odnosu na ostale jezike visokog nivoa. Efikasno programiranje podrazumeva odlično poznavanje pravila adresne aritmetike. Pretpostavićemo da su:

a                   vektor sa elementima proizvoljnog tipa T  
n                    celobrojni izraz  
v                    izraz tipa T  
pa1, pa2 ukazatelj na tip T, koji ukazuju na elemente vektora a.

Pravila adresne aritmetike su sledeća:

a                    je ukazatelj na prvi element vektora a. Tip ukazatelja na T.  
&a[0]               je ukazatelj na prvi element vektora a. Tip ukazatelja na T.  
&a[n]               je ukazatelj na n+1-ti element vektora a. Tip ukazatelja na T.  
\*pa1                je element vektora a, na koji pa1 ukazuje. Tip T.  
\*pa1=v             dodeljuje vrednost v elementu na koji pa1 ukazuje. Tipa T.  
++pa1              postavlja pa1 na sledeci element vektora a, bez obzira na tip elementa vektora a. Tip ukazatelja na T.  
--pa1              postavlja pa1 na predhodni element vektora a, bez obzira na tip elementa vektora a. Tip ukazatelj na T.  
\*++pa1            inkrementira pa1, zatim pristupa elementu vektora a, na koji pa1 ukazuje. Tipa T.  
\*pa1++            pristupa elementu vektora a na koji pa1 ukazuje, zatim inkrementira pa1. Tip T.  
pa1+n             je ukazateljski izraz, koji ukazuje na n-ti naredni element vektora a, u odnosu na element na koji pa1 ukazuje. Tip ukazatelj na T.  
pa1-n             je ukazateljski izraz, koji ukazuje na n-ti predhodni element vektora a, u odnosu na element na koji pa1 ukazuje. Tip ukazatelj na T.  
\*(pa1+n)=v       dodeljuje vrednost v narednom n-tom elementu vektora a, u odnosu na element na koji pa1 ukazuje. Tip T.  
pa1>pa2           ispituje vrednost adresa u pa1 i pa2\*  
pa2-pa1           određuje broj elementa vektora a smestjenih izmedju ukazatelja pa2 i pa1 (pod pretpostavkom da je pa1>pa2).  
\*(a+n)            je n-ti element vektora a. Tip T.

Program 4.61. ilustruje neke navedene principe adresne, odnosno, ukazateljske aritmetike.

*/\*Program 4.61.\*/*

*/\*Program za ilustraciju adresne aritmetike\*/*

```
main()
{
    static char c[]={'a','b','c','d'};
```

```

static int i[]={1,2,3,4};
static double d[]={1.1,2.2,3.3,4.4};
char *pc=&c[0], *pc0=&c[3];
int *pi=&i[0], *pi0=&i[3];
double *pd=&d[0], *pd0=&d[3];

printf(" *c=%c *i=%d *d=%f\n",
      *c, *i, *d);
printf(" *pc=%c *pi=%d *pd=%f\n",
      *pc, *pi, *pd);
printf(" *(pc+1)=%c *(pi+2)=%d *(pd+3)=%f\n",
      *(pc+1), *(pi+2), *(pd+3));
printf(" *++pc=%c *++pi=%d *++pd=%f\n",
      *++pc, *++pi, *++pd);
printf(" *pc++=%c *pi++=%d *pd++=%f\n",
      *pc++, *pi++, *pd++);
++pc, ++pi, ++pd;
printf(" *pc=%c *pi=%d *pd=%f\n",
      *pc, *pi, *pd);
printf(" *pc=%c *pi=%d *pd=%f\n",
      *pc='e', *pi=5, *pd=5.5);
pc-=3, pi-=3, pd-=3;
printf(" *pc=%c *pi=%d *pd=%f\n",
      *pc, *pi, *pd);
printf(" pc0-pc=%d pi0-pi=%d pd0-pd=%d\n",
      pc0-pc, pi0-pi, pd0-pd);
}

```

## Program 4.61. Izlaz

```

*c=a *i=a *d=1.100000
*pc=a *pi=1 *pd=1.100000
*(pc+1)=b *(pi+2)=3 *(pd+3)=4.400000
*++pc=b *++pi=2 *++pd=2.200000
*pc++=b *pi++=2 *pd++=2.200000
*pc=d *pi=4 *pd=4.400000
*pc=e *pi=5 *pd=5.500000
*pc=a *pi=1 *pd=1.100000
pc0-pc=3 pi0-pi=3 pd0-pd=3

```

Program 4.62. ilustruje operaciju inkrementiranja i dekrementiranja ukazatelja. Ako je `pt` ukazatelj tipa `t` operacijom inkrementiranja i dekrementiranja vrednosti `pt` se uvećava ili umanjuje za `sizeof(t)`. Izlaz programa 4.62. pokazuje da se ukazatelj na `char`, operacijom inkrementiranja, uvećava za 1, ukazatelj na `int` za 2, a ukazatelj na `double` za 8. Uočite kast operator u funkcijama `printf`. Da nismo upotrebili kast dobili bismo broj elemenata između ukazatelja, odnosno, vrednosti 1 u sva tri slučaja.

---



---

```
/*Program 4.62.*/
```

```
/*Program za ilustraciju inkrementiranja i
dekrementiranja ukazatelja*/
```

```
main()
{
    char c, *p1c=&c, *p2c=&c+1;
    int i, *p1i=&i, *p2i=&i+1;
    double d, *p1d=&d, *p2d=&d+1;
    printf("char inkrement=%d\n", (int)p2c-(int)p1c);
    printf("int inkrement=%d\n", (int)p2i-(int)p1i);
    printf("double inkrement=%d\n", (int)p2d-(int)p1d);
}
```

Program 4.62. Izlaz

```
char inkrement=1
int inkrement=2
double inkrement=8
```

### 4.13. Nizovi karaktera

Ovo poglavlje detaljno opisuje nizove karaktera. C jezik omogućava efikasnu i jednostavnu manipulaciju nizovima karaktera. Svaki C prevodilac u svojoj sistemskoj biblioteci sadrži veliki broj funkcija za rad sa nizovima, kao što su: testiranje karaktera, konkatencija, komparacija, kopiranje, konverzija nizova karaktera, id. Nizovi

karaktera se najčešće javljaju pri obradi teksta, pa C jezik, sa svojom sistemskom bibliotekom, omogućava stvaranje snažnih procesora tekstova.

#### 4.13.1. Pojam nizova karaktera

Nizovi karaktera su jednodimenzionalni vektori tipa char. U C jeziku specijalni karakter označava kraj niza. Taj karakter se naziva završni ili nulti karakter (engl. null), jer je njegova numerička predstava jednaka 0. Stoga, nizovi karaktera u C jeziku su promenljive dužine sa završnim nultim karakterom '\0', sa maksimalnom dužinom, koja je određena veličinom rezervisanog memorijskog prostora u deklaraciji niza. Veličina niza, definisana u deklaraciji, mora da uključi i završni karakter. Na primer, deklaracija:

```
char string[10];
```

deklariše vektor tipa char veličine 10 karaktera za prihvatanje niza od 9 karaktera i završnog nultog karaktera.



Konstantni niz karaktera se piše između duplih navodnica. Na primer, "xyz10" je konstantni niz karaktera veličine 5 karaktera (šesti karakter je završni karakter). Zbog prisustva završnog karaktera, "x" nije jednak konstantnom karakteru 'x', jer je "x" niz od dva karaktera, 'x' i '\0'. Slično tome, niz "" je nulti ili prazan niz, jer se sastoji samo od završnog karaktera.

Prva značajna karakteristika nizova u C jeziku je mogućnost inicijalizacije vektora tipa char konstantnim nizom. Na primer, iskaz

```
static char string[10]="Demo niz";
```

Inicijalizuje niz string koji se inicijalizuje karakterima: 'D', 'e', 'm', 'o', ' ', 'n', 'i', 'z',

```
static char string[10]={'D','e','m','o',' ','n','i','z','\0'};
```

Očigledno je, da je prvi iskaz jednostavniji za pisanje i čitanje.

Druga značajna osobina nizova u C jeziku tiče se štampanja. Specijalni konverzioni karakter %s u konverzionom nizu funkcije printf može da se koristi za štampanje vektora tipa char, koji sadrži nizove ili za štampanje konstantnih nizova. Koristeći predhodnu deklaraciju vektora string, poziv funkcije printf.

```
printf("%s\n", string);
```

može biti korišćen za štampanje sadržaja vektora string. Funkcija printf, kada u konverzionom nizu naiđe na niz %s, "shvata" da se na odgovarajućem mestu u lista argumenata nalazi ukazatelj na niz.

Program 4.63. ilustruje manipulaciju nizovima karaktera koristeći funkciju strcpy. Funkcija strcpy kopira niz karaktera, specificiran prvim argumentom s1, u drugi niz, specificiran drugi argumentom s2. Argumenti s1 i s2 su ukazatelji na niz karaktera, a rezultat funkcije je ukazatelj na drugi niz. strcpy nije preporučljivo pozivati sa jednakim prvim i drugim karakterom, s1=s2, odnosno, strcpy(s, s), jer funkcija neće naći završni karakter, pa se kopira cela memorija u naznačeni vektor do abortovanja programa od strane operativnog sistema.

Ovaj problem se može izbeći reprogramiranjem while iskaza u

```
while (*s1==*s2)
    {
        ++s1; ++s2
    }
```

*/\*Program 4.63.\*/*

*/\*Program za ilustraciju kopiranja nizova karaktera\*/*

```
char*strcpy(s1, s2)
    char*s1, *s2;
    {
        char *u;

        u=s2;
```

*/\*F ja za kopiranje nizova\*/*

```

    while (*s2++=*s1++);
        return(u);
}

```

```

main()
{
    static char demo1[]="Ovaj se niz kopira";
    char demo2[20];

    strcpy(demo1, demo2);
    printf("Niz demo1=%s\n", demo1);
    printf("Niz demo2=%s\n", demo2);
}

```

#### Program 4.63. Izlaz

```

Niz demo1=Ovaj se niz kopira
Niz demo2=Ovaj se niz kopira

```

U programu 4.64. koristi se funkcija `strlen`, koja određuje dužinu niza bez završnog karaktera. `strlen` vraća unsigned vrednost, a kao argument prihvata ukazatelj na niz, cilja se dužina određuje. Radi efikasnosti programa, argument `s` i funkcijska lokalna promenljiva `length` su deklarirane kao registarske.

*/\*Program 4.64.\*/*

```

/*Program za odredjivanje duzine nizova karaktera*/
unsigned strlen(s)                                /*F ja za odredjivanje duzine niza*/
    register char*s;
    {
        register unsigned i;
        for (i=0; *s !='\0'; ++s, ++i);
        return(i)
    }
main()
    {
        static char demo[]="Uvod u C jezik";
        printf("Duzina niza demo je %d\n", strlen(demo));
    }

```

#### Program 4.64 Izlaz

```

Duzina niza demo je 14

```

Treća važna karakteristika je učitavanje nizova karaktera. Funkcija `scanf`, slično funkciji `printf`, prihvata konverzioni karakter `%s` za učitavanje niza karaktera do prvog praznog mesta (blank, tab) ili do kraja linije. Na primer, koristeći deklaraciju niza `s1`.

```

char s[81];

```

funkcija `scanf` u iskazu `scanf("%s", s)`; učitava niz karaktera i smešta u vektor `s`. Ako se izvršava funkcija `scanf` i na terminalu ukuca `test` tada `scanf` učitava niz "test" i smešta

u vektor s. Ako se pri izvršavanju scanf ukucaju dve reči test1 test2 tada se prvo učitava reč test1 i smešta u s kao niz, pošto prazno mesto posle prve reči označava kraj reči, odnosno, niza. Ako se scanf izvršava ponovo, učitava se reč test2 i smešta u s, jer scanf nastavlja učitavanje od zadnjeg predhodno učitavanog karaktera. Kraj druge reči je označen krajem linije. scanf automatski dodaje završni karakter posle učitanih reči.

Program 4.65. ilustruje proces učitavanja niza. U programu 4.65. koriste se funkcije isdigit i isalpha. isalpha prihvata karakter kao argument i vraća vrednost različitu od 0, ako je karakter slovo (malo ili veliko). U protivnom, vraća 0. isdigit vraća vrednost različitu od 0, ako je argument cifra, u protivnom vraća 0. main učitava proizvoljni niz i za svaki karakter u nizu određuje da li je slovo, cifra ili nešto treće. Funkcije isalpha i isdigit su primer funkcija za testiranje karaktera.

```

/*Program 4.65.*/
/*Program za ucitavanje nizova karaktera*/
int isdigit(c)                                /*F ja za odredjivanje cifre*/
    char c;
    {
    return((c>='0' && c<='9')? 1 : 0);
    }
int isalpha(c)                                 /*F ja za odredjivanje cifre/slova*/
    char c;
    {
    return(((c>='A' && c<='Z') || (c>='a' && c<='z')) ? 1 : 0);
    }
main()
    {

    char s[81], *u;
    int i;

    printf("Ukucajte niz:\n");
    scanf("%s", s);
    u=s;
    for (i=0; *u !='\0'; ++i, ++u)
        {
        printf("%2d. karakter je ", i+1);
        if (isdigit(*u))
            printf("%s\n", "cifra");
        else if (isalpha(*u))
            printf("%s\n", "slovo");
        else
            printf("%s\n", "drugo");
        }
    }

```

#### Program 4.65. Izlaz

Ukucajte niz:  
12aA.j

1. karakter je cifra

2. karakter je cifra
3. karakter je slovo
4. karakter je slovo
5. karakter je drugo
6. karakter je slovo

#### 4.14. Operacije na bitovima

C jezik je prvenstveno namenjen za sistemsko programiranje. Ukazatelji su sredstva za eksplicitnu kontrolu nad rezervacijom i oslobađanjem operativne memorije. Pored ove činjenice, u sistemskom programiranju često se javlja potreba za operacijama na bitovima. Takođe, pri projektovanju sistema za upravljanje u realnom vremenu nužna je mogućnost pristupa pojedinačnim bitovima u okviru bajta ili reči (reč sadrži 2-4 bajta). C jezik u svom asortimanu operatora ima operatore, koji su specijalno namenjeni za rad sa bitovima. Naredni tekst detaljnije opisuje pojam bita, operacije na bitovima i novi tip podataka, bitsko polje (engl. bit field), koji u određenim slučajevima olakšava manipulaciju bitovima.

##### 4.14.1. Pojam bita

U predhodnim poglavljima često smo koristili pojam bajta. Bajt se sastoji od osam bita i u mnogim računarskim sistemima predstavlja najmanju adresibilnu ćeliju operativne memorije. Bitovi predstavljaju binarne cifre i shodno tome, mogu imati

vrednosti 0 ili 1. Na prier, bajt memorisan na određenoj memorijskoj lokaciji, možemo predstaviti kao niz od osam binarnih cifara

01100110

Bit na krajnjem desnom mestu je bit najmanje važnosti (engl. lest significant bit ili LBS), a bit na krajnjem levom mestu je bit najveće važnosti (engl. most significant bit ili MSB) u bajtu. Kao i u svakom brojnom sistemu, i u binarnom sistemu cifre nose određenu težinu. U gornjem bajtu prva cifra (posmatramo sa desna na levo) nosi težinu 1, druga 2, treća 4, četvrta 8, ..., osma 128. Shodno tome, gornji bajt sadrži vrednost izraženu u dekadnom sistemu

$$0*128+1*64+1*32+0*16+0*8+1*4+1*2+0*1=102$$

Predstavljanje negativnih brojeva je nešto drugačije. U mnogim računarskim sistemima negativni brojevi su predstavljeni dvojnim komplementom. U ovoj notaciji bit najveće važnosti sadrži informaciju o znaku broja, pa tako ako je ovaj bit jednak binarnoj 1, znači da je memorisan negativan broj, a ako je jednak binarnoj 0 memorisani broj je pozitivan. Ostali bitovi predstavljaju vrednost broja. Dvojni komplement negativnog broja dobija se kada se na jednačini komplement njegove apsolutne vrednosti doda vrednost 1. Pod jediničnim komplemetiranjem podrazumeva se operacija u kojoj svi

bitovi menjaju vrednost, tj., binarne 0 postaju binarne 1, a binarne 1 postaju binarne 0. Na primer, dvojni komplement broja -27 dobija se na sledeći način

$$\begin{array}{rcl}
 \text{apsolutna vrednost 27} & = & 00011011 \\
 \text{jedinični komplement} & = & 11100100 \\
 \text{dodavanje 1} & = & \phantom{11100}1 \\
 \hline
 \text{dvojni komplement -27} & = & 11100101
 \end{array}$$

Konverzija dvojnog komplementa nekog broja u ekvivalentnu binarnu vrednost je inverzna: od dvojnog komplementa broja oduzima se vrednost 1 i jediničnim komplementiranjem tako dobijene vrednosti dobija se apsolutna vrednost memorisane negativne vrednosti u binarnom brojnem sistemu. Na primer,

$$\begin{array}{rcl}
 \text{dvojni komplement -27} & = & 11100101 \\
 \text{oduzimanje 1} & = & \phantom{11100}1 \\
 & = & 11100100 \\
 \text{jedinični komplement} & = & 00011011 \\
 \hline
 \text{apsolutna vrednost 27} & = & 00011011
 \end{array}$$

Imajući u vidu dvojni komplement i rezervisanje bita najveće važnosti za predstavljanje znaka broja, najveća vrednost koja se može smestiti u bajtu je +127, a najmanja -128. Slično, ako pretpostavimo da se vrednosti tipa int memorišu u dva bajta, najveća vrednost tipa int je +32767, a najmanja -32768. Kvalifikator unsigned eliminiše upotrebu bita najveće važnosti za predstavljanje znaka, pa se u navedene memorijske lokacije memorišu samo apsolutne vrednosti, koristeći i bit najveće važnosti. Time se povećava opseg pozitivnih brojeva, pa se u bajt mogu smestiti vrednosti od 0 do 255, a u

2 bajta vrednosti od 0 do 65535. U daljem tekstu postavićemo da se za vrednost int koriste 2 bajta (u nekim sistemima upotrebljavaju se 4 bajta).

#### 4.14.2. Operacije na bitovima

Operatori C jezika, koji realizuju operacije na bitovima, predstavljeni su u tabeli. Svi navedeni operatori su binarni, izuzev operatora jediničnog komplementiranja, koji je unaran. Operatori se mogu primenjivati na tipove char i int, sa ili bez kvalifikatora short, long i unsigned. Upotreba ovih operatora na tipove float i double nije moguće, a reakcije C prevodilaca u ovakvim slučajevima je vrlo različita.

Operacije na bitovima	Operatori
Jedinični komplement	~
AND	&
Inkluzivno OR	
Ekskluzivni OR ili XOR	^
Levo pomeranje	<<
Desno pomeranje	>>

Kada se operacije na bitovima obavljaju između dve vrednosti različite veličine (recimo između long int i short int), C prevodilac automatski sa leve strane popunjava nedostajuća cifarska mesta kraćeg operanda. Ako je kraći operand vrednost sa znakom (bez unsined), tada se znak proširuje na levo do dužine dužeg operanda. Znači, ako je kraći operand negativan, tada se binarna 1 proširuje do dužine dužeg operanda. Ako je pozitivan, proširuje se binarno 0. Na primer, ako se vrednost tipa short int memoriše u 16 bita, a vrednost tipa long int u 32 bita, tada se pri primeni operatora na bitovima, vrednost tipa short int proširuje na 32 bita i gornja dva bajta popunjavaju binarnim 1, ako je vrednost tipa short int negativna, ili binarnim 0, ako je vrednost tipa short int pozitivna. Ako je kraći operand tipa unsigned, onda se bitovi do dužine dužeg operanda popunjavaju binarnim 0.

Operatori na bitovima, kao i svi operatori C jezika, imaju prioritet i asocijativnost, koji određuju redosled primene operatora u izrazima.

#### 4.14.2.1. Jedinični komplement

Operator  $\sim$  realizuje operaciju jediničnog komplementiranja i naziva se operator jediničnog komplementa. Operator  $\sim$  je unaran operator i invertuje bitove operanda. To znači, da se sve binarna 1 operanda komplementiraju u binarne 0, a binarne 0 u binarne 1. Tabela istinitosti je:

b	$\sim b$
$\overline{1}$	$\overline{0}$
0	1

gde b označava proizvoljni bit operanda. Jedinični komplement ne treba mešati sa unarnim minusom i logičkom negacijom. Ako je p promenljiva tipa int postavljena na vrednost 0, tada se primenom operatora unarnog minusa dobija vrednost 0. Primenom operatora logičke negacije dobija se vrednost 1(=TRUE), jer promenljiva p ima vrednost 0 (=FLASE). Međutim, primenom operatora jediničnog komplementa dobija se broj, čiji su bitovi postavljeni na binarne 1, a to je vrednost -1.

#### 4.14.2.2. Inkluzivni OR

Operator  $|$  realizuje operaciju OR i naziva se operator inkluzivni OR (ili samo operator OR). Operator  $|$  je binarni operator, koji upoređuje pojedinačne bitove binarnih predstava operanada. Ako su odgovarajući bitovi prvog ili drugog operanda ili oba jednaki binarnoj 1, rezultat primene operatora inkluzivni OR je binarna 1. U protivnom, ako su oba bita jednaka binarnoj 0, dobija se binarna 0. Tablica istinitosti je sledeća:

b1	b2	$b1   b2$
$\overline{0}$	$\overline{0}$	$\overline{0}$
0	1	1
1	0	1
1	1	1

gde su b1 i b2 odgovarajući bitovi prvog i drugog operanda. Ako je p1 promenljiva tipa unsigned int, postavljena na oktalnu vrednost 0110572, i p2 promenljiva tipa unsigned int, postavljena na oktalnu vrednost 0014327, tada se operatorom inkluzivni OR dobija rezultat

p1	1 001 000 101 111 010	0110572
p2	0 001 100 011 010 111	0014327
p1 p2	<u>1 001 100 111 111 111</u>	<u>0114777</u>

#### 4.14.2.3. AND

Operator & realizuje operaciju AND, upoređujući pojedinačne bitove binarnih predstava operanada. Ako su odgovarajući bitovi jednaki binarnim 1, rezultat primene operatora & je binaran 1. U svakom drugom slučaju, rezultat je binarna 0. Operator & naziva se operator AND. Shodno definiciji operacije AND, tablica istinitosti operatora AND je:

b1	b2	b1 & b2
<u>0</u>	<u>0</u>	<u>0</u>
0	1	0
1	0	0
1	1	1

gde su b1 i b2 odgovarajući bitovi prvog i drugog operanda, respektivno. Ako je p1 promenljiva tipa unsignedint, postavljena na oktalnu vrednost 0110572, i p2

promenljiva tipa unsigned int, postavljena na oktalnu vrednost 0014327, tada se operatorom AND dobija sledeća vrednost

p1	1 001 000 101 111 010	0110572
p2	0 001 100 011 010 111	0014327
p1&p2	<u>0 001 000 001 010 010</u>	<u>0010122</u>

#### 4.14.2.4. Ekskluzivni OR

Operator ^ realizuje operaciju ekskluzivni OR (ili samo operacija XOR). Operator ekskluzivni OR daje kao rezultat vrednost binarne 1, ako odgovarajući bitovi u prvom ili drugom operandu imaju različite binarne vrednosti. U protivnom, rezultat primene ovog operatora je binarna 0. Tablica operatora ekskluzivne OR je sledeća

b1	b2	b1^b2
<u>0</u>	<u>0</u>	<u>0</u>
0	1	1
1	0	1
1	1	0

gde su b1 i b2 odgovarajući bitovi prvog i drugog operanda, respektivno.

Ako je p1 promenljiva tipa int, postavljena na oktalnu vrednost 0110572 i p2 promenljiva tipa int, postavljena na oktalnu vrednost 0014327, tada se operatorom ekskluzivno OR dobija rezultat

p1	1 001 000 101 111 010	0110572
p2	0 001 100 011 010 111	0014327
p1^p2	<u>1 000 100 110 101 101</u>	<u>0104655</u>

Ako je p1=p2 operator daje vrednost 0. Ovo svojstvo operatora ekskluzivni OR može se koristiti za međusobnu izmenu vrednosti dve promenljive, bez korišćenja pomoćne promenljive.

Program 4.66. ilustruje korišćenje operatora jediničnog komplementa, inkluzivnog i ekskluzivnog OR i operatora AND. Drugi i treći poziv funkcije printf ilustruje efekt operatora jediničnog komplementa, AND, inkluzivnog i ekskluzivnog OR. Interesantni su izrazi u četvrtom i petom pozivu funkcije printf, koji pokazuju De Morganova pravila

$$a | b = \sim(\sim a \& \sim b)$$

$$a \& b = \sim(\sim a | \sim b)$$

Šesti poziv funkcije printf verifikuje svojstvo ekskluzivnog OR operatora za jednake operande. U zadnjem pozivu funkcije printf verifikuje se postupak izmene vrednosti promenljivih p1 i p2 bez korišćenja pomoćne promenljive.

```

/*Program 4.66.*/
/*Program za ilustraciju operacije na bitovima*/
main()
{
    unsigned int p1=015252, p2=0052525;
    printf("p1=%0 p2=%0\n", p1, p2);
    printf("~p1=%0 ~p2=%0\n", ~p1, ~p2);
    printf("p1&p2=%0 p1/p2=%0 pi^p2=0\n", p1&p2, p1/p2, p1^p2);
    printf("p1&p2=%0 ~(~p1/~p2)=%0\n", p1&p2, ~(~p1/~p2));
    printf("p1/p2=%0 ~(~p1&~p2)=%0\n", p1/p2, ~(~p1&~p2));
    printf("p1^p1=%0 p2^p2=%0\n", p1^p1, p2^p2);
    p1^=p2;
    p2^=p1;
    p1^=p2;
    printf("p1=%0 p2=%0\n", p1, p2);
}

```

Program 4.66 Izlaz

```

p1=125252 p2=52525
~p1=52525 ~p2=125252
p1&p2=0 p1/p2=177777 p1^p2=177777
p1&p2=0 ~(~p1/~p2)=0
p1/p2=177777 ~(~p1&~p2)=177777
p1^p1=0 p2^p2=0
p1=52525 p2=125252

```



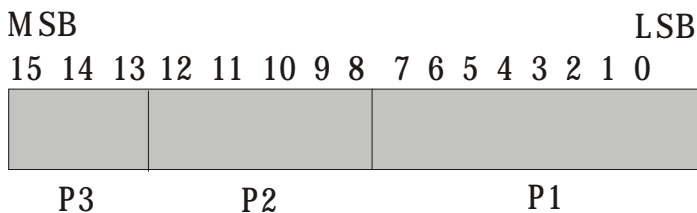
### 4.14.3. Bitsko polje

U C jeziku postoje dve metode za pakovanje i raspakovanje podataka sa ciljem štednje memorijskog prostora. Prvi način je jednostavno predstavljanje podataka u memorijskom prostoru celobrojnih vrednosti, tj. tipa int, koristeći princip maskiranja bitova i definisane operacije na bitovima. Drugi način je mogućnost definisanja specijalne strukture sa upakovanim podacima, koja se u C jeziku naziva bitsko polje (ili polje bitova). Radi ilustracije ova dva načina, pretpostavićemo da trebamo upakovati tri podatka p1, p2 i p3. Podatak p1 uzima vrednosti od 0 do 255, pa se može predstaviti sa 8 bitova; podatak p2 uzima vrednosti u opsegu od 0 do 31, što zahteva 5 bitova za predstavljanje podataka p2; podatak p3 uzima vrednosti u opsegu od 0 do 7, što zahteva 3 bita.

Prvi način za pakovanje podataka p1, p2 i p3 zahteva promenljivu tipa int u koju se pakuju podaci p1, p2 i p3. Stoga, deklarisaćemo promenljivu pack\_data iskazom

```
unsigned int pack_data;
```

u kojoj proizvoljnim redosledom dodeljujemo bitove ili bitska polja za memorisanje podataka p1, p2 i p3. Na slici 11.1. predstavljena je jedna moguća raspodela bitovima za podatke p1, p2 i p3, pod pretpostavkom da tip int zauzima 16 bitova.



Slika 11.1. Raspodela bitova u promenljivoj pack\_data

Pravilnom sekvencom operacija na bitovima u blisko polje sa slike 11.1. mogu se upakovati podaci p1, p2 i p3. Na primer, pakovanje podataka p2 zahteva dva postavljenja: (1) rezervisanog bitskog polja na 0 i (2) bitskog polja na neku željenu vrednost, tj.

```
pack_data &=~(0x1F << 8);
pack_data |= d2 << 8;
```

gde d2 predstavlja vrednost, koja se memoriše u rezervisano bitsko polje. Maska 0x1F, pomerena za 8 bitskih mesta ulevo i invertovana, briše predhodni sadržaj bitskog polja i ostavlja nepromenljene ostale bitove u promenljivoj pack\_data. Vrednost d2 mora biti manja ili jednaka maksimalnoj vrednosti, koja se može smestiti u bitsko polje od 5 bitova (31), jer se, u protivnom, bitsko polje postavlja na pogrešnu vrednost. C prevodilac ne signalizira ovakav tip greške.

Raspakivanje podataka p2 se ostvaruje iskazom:

```
p2=(pack_data >> 8) & 0x1F;
```

Operacijom AND između maske 0x1F i vrednosti promenljive pack\_data pomerene za 8 bitskih mesta dobijamo vrednost upakovanog podatka. Promenljivoj p2 dodeljuje se raspakovana vrednost iz promenljive pack\_data.

Program 4.67. ilustruje korišćenje operatora na bitovima za pakovanje i raspakovanje već definisanih podataka p1, p2 i p3. Prva tri izvršna iskaza postavljaju rezervisana bitska polja na 0. Druga grupa izvršnih iskaza pakuje vrednosti d1, d2 i d3 u promenljivu pack\_data. Na kraju programa 11.4. obavlja se raspakivanje i štampanje raspakovanih vrednosti. Možemo zaključiti, da metod za pakovanje i raspakovanje predstavljen u programu 11.4. ima sledeće nedostatke:

1. Pakovanje i raspakovanje podataka primenom operacija na bitovima zahteva izraze, koji nisu dovoljno čitljivi,
2. Otežanu modifikaciju maski u slučaju promene širine bitskog polja u kasnijoj fazi razvoja programa.

*/\*Program 4.67.\*/*

*/\*Program za ilustraciju pakovanja i raspakovanja koriscenjem operacija na bitovima\*/*

```
main()
{
    unsigned int p1, p2, p3;
    unsigned int pack_data, d1=127, d2=15, d3=6;
    printf("d1=%d d2=%d d3=%d\n", d1, d2, d3);

    pack_data &=~0xFF;
    pack_data &=~(0x1F<<8);
    pack_data &=~(0xF <<13);

    pack_data |=d1;
    pack_data |=d2<< 8;
    pack_data |=d3<<13;

    p1=pack_data & 0xFF;
    p2=(pack_data>>8) & 0x1F;
    p3=(pack_data>>13) & 0x7;

    printf("p1=%d p2=%d p3=%d\n", p1, p2, p3);
}
```

**Program 4.67 Izlaz**

```
d1=127 d2=15 d3=6
d1=127 d2=15 d3=6
```

#### 4.15. Ulaz i izlaz

C jezik ne obezbeđuje specijalne iskaze za obavljanje ulazno/izlazne aktivnosti. Za razliku od FORTRAN-a, na primer, koji ima komande READ i WRITE definisane kao deo jezika, sve ulazno/izlazne operacije u C jeziku se objavljuju korišćenjem specijalnih funkcija iz systemske biblioteke. Pošto ove funkcije nisu deo C jezika, mogu se razlikovati od sistema do sistema, kako po imenu, tako i po načinu funkcionisanja. Međutim, neke funkcije, kao printf i scanf, su manje ili više standardizovane i postoje u svim C prevodiocima. Ove dve funkcije (i mnoge druge) deo su standardne systemske biblioteke C prevodilaca. Datoteka stdio.h sadrži definisane makroe, strukture, promenljive i konstante, koje su neophodne za funkcionisanje ulazno/izlaznih funkcija. Zato se datoteka stdio.h mora uključiti u program iskazom

```
#include <stdio.h >
```

U ovom poglavlju opisane su standardne ulazno/izlazne funkcije, prisutne u najvećem broju C sistema. Na nesreću, prostor nam ne dozvoljava detaljan opis ovih funkcija niti opisivanje ostalih funkcija iz standardne biblioteke. Zbog mogućih razlika između sistema, preporučujemo da, pre upotrebe ovih funkcija, konsultujete dokumentaciju sistema na kojem radite.

##### 4.15.1. Ulaz/izlaz karaktera- getchar() i putchar()

getchar i putchar su najčešće realizovane kao makrodefinicije. getchar učitava karakter sa terminala i ne zahteva argumente. Ako je c promenljiva tipa char, tada se getchar poziva iskazom

```
c=getchar();
```

putchar je inverzna makrou (ili funkciji) getchar i ispisuje karakter na terminal. Argument je karakter, koji se ispisuje na terminal. Ako je c tipa char, tada iskaz

```
putchar(c);
```

ispisuje karakter na terminal. Na primer,

```
putchar('A');
```

ispisuje karakter 'A' na ekran terminala.

##### 4.15.2. Formatizovani ulaz/izlaz

Funkcija printf i scanf su korišćene u mnogim dosadašnjim programskim primerima. Prvi argument ovih funkcija je niz, koji specificira format po kojem se ispisuju ili učitavaju podaci.

#### 4.15.2.1. Funkcija printf()

Funkcija printf omogućava proizvoljnu dužinu liste argumenata i specificiranje formata, koji "skriva" komplikovanost konverzije podataka. Lista argumenata ima dva dela: konverzioni niz za specificiranje formata i listu argumenata, čije se vrednosti ispisuju. Format, po kome funkcija printf ispisuje vrednosti iz lista argumenata, navodi se u prvom argumentu i počinje karakterom procent '%', a završava sa jednim od konverzionih karaktera. Opšti oblik prvog argumenta je

```
"[tekst]%[-] [sirina_polja] [.] [tacnost] [[] konverzioni_karakter"
```

Karakter i između '%' i konverzionog karaktera nazivaju se modifikatori konverzije i definišu položaj vrednosti u okviru polja (levo ili desno poravnjanje), širinu polja u kojem se vrednost ispisuje i tačnost prikazane vrednosti.

Karakter	Opis ispisivane vrednosti
d	Dekadna celobrojna vrednost tipa int.
u	Dekadna celobrojna vrednost tipa unsigned.
o	Oktalna celobrojna vrednost.
x	Heksadecimalna celobrojna vrednost; za cifre veće od 9 koriste se mala slova a, b, c, d, e i f.
X	Heksadecimalna celobrojna vrednost; za cifre veće od 9 koriste se velika slova A, B, C, D, E i F.
f	Realan broj; ako tačnost nije navedena, ispisuje se 6 decimalnih mesta
e	Realan broj u naučnoj notaciji; jedna cifra se uvek prikazuje pre decimalne tačke, broj decimalnih mesta je po definiciji 6, ako drugačije nije određeno; kspONENT je predhodan malim slovom e.
E	Isto kao format e, izuzev što se koristi veliko slovo E.
g	Format e ili f, u zavisnosti od toga, koji je kraći.
G	Format E ili f, u zavisnosti od toga, koji je kraći.
c	Karakter.
s	Niz karaktera do završnog karaktera; specificiranjem tačnosti ispisuje se manji broj karaktera.

Tabela 13.2. Konverzioni karakteri funkcije printf

Program 4.68. ilustruje neke moguće kombinacije. Vertikalna crta vizuelno prikazuje širinu polja. Prva grupa poziva funkcije printf prikazuje modifikatore i konverzije karaktera za ispisivanje celih brojeva u tri moguće notacije: dekadna, oktalna i heksadecimalna. Druga grupa iskaza ispisuje po raznim formatima vrednosti tipa float i double, dok treća grupa iskaza ilustruje primenu raznih formata za štampanje karaktera i nizova karaktera. Zapazite poziv :

```
printf("%5.1s/n", &c);
```

u kojoj se po formatu s štampaju karakteri. Ovaj primer je naveden kao ilustracija zavisnosti tačnosti i broja štampanih karaktera, odnosno, ako je tačnost manja od dužine niza, štampa se broj karaktera određen tačnošću.

```

/*Program 4.68.*/
/*Program za ilustraciju modifikatora i konverzionih karaktera*/
main()
{
    char          c='x';
    static char   s[]="Niz karaktera";
    int           i=22;
    unsigned int  u=0x1AEF;
    long int      l=150000L;
    float         f=9.935;
    double        d=-2.123456789;

    printf("\nCeli brojevi:\n");
    printf("%d|x|ld\n", i, u, l);
    printf("%d%5d%-5d|05d\n", i, i, i, i);
    printf("%x|7.2u|%-7.2u|07d\n", u, u, u, u);
    printf("%ld|010lx|%-lo|lu\n", l, l, l, l);
    printf("\nRealni brojevi:\n");
    printf("%f|f|\n", f, d);
    printf("%20.10e|%-20.10f|\n", f, d);
    printf("%020.10E|%.2e|\n", f, d);
    printf("%3.f|%.2f|\n", f, d);
    printf("%5f|.8e|\n", f, d);
    printf("%20f|%-20e|\n", f, d);
    printf("%-020.1f|%-20.3E|\n", f, d);
    printf("\nNizovi i karakteri:\n");
    printf("%c|s|\n", c, s);
    printf("%c|3c|%-3c|\n", c, c, c);
    printf("%5.l|\n", &c);
    printf("%s|\n", s);
    printf("%13s|20s|%-20s|\n", s, s, s);
    printf("%10.2s|10.4s|10.7s|\n", s, s, s);
    printf("%-10.2s|%-10.4s|%-10.7s|\n", s, s, s);
    printf("%.*s|%.2s|%.4s|10.7s|\n", 10, 4, s, 10, 7, s, s);
}

```

### Program 4.68 Izlaz

#### Celi brojevi:

```

|22|1aef|150000| |
|22| 22|22 |00022|
|1aef| 6895|6895 |0006895|
|150000|00000249f0|444760|150000|

```

#### Realni brojevi:

```

|9.935000|-2.123457|
| 9.9350004196e+00|-2.1234567890 |
| 9.9350004196E+00|-2.12e+00|
| 10|-2|
|9.935000|-2.12345679e+00|
| 9.935000|-2.123457e+00 |
|9.9 |-.2123E+00 |

```

**Nizovi i karakteri:**

```

|x|Niz karaktera|
|x| x|x |
| x|
|Niz karaktera|
|Niz karaktera| Niz karaktera|Niz karaktera |
| Ni| Niz | Niz kar|
|Ni |Niz |Niz kar |
| Niz | Niz kar| Niz kar|

```

**4.15.2.2. Funkcija scanf()**

Funkcija `scanf` ima dve važne karakteristike: dužina liste argumenata je proizvoljan i format za učitavanje "skriva" komplikovanost konverzije podataka. Lista argumenata ima dva dela: konverzioni niz za specificiranje formata po kome se učitavaju podaci i lista argumenata sa adresama promenljivih u koje se smeštaju učitane i konvertovane vrednosti. Opšti oblik formata, koji se specificira kao prvi argument, je sledeći

```
"[tekst%][*] [sirina_polja] [h] [] konverzioni_karakter%"
```

Slično funkciji `printf`, karakter između '%' i konverzionog karakteri nazivaju se modifikatori konverzije.

Funkcija `scanf` prilikom učitavanja, preskače nešampajući ili nevidljive karaktere (blank, tab, linija), izuzev u slučaju konverzionog karaktera `c`. U tom slučaju učitava se sledeći karakter, bez obzira da li je vidljiv ili nevidljiv. Sličan efekat postiže se korišćenjem uglastih zagrada, u kojima se navode svi dozvoljeni karakteri u nizu (mogu biti navedeni i nevidljivi).

Tabela 13.4. Konverzioni karakteri funkcije `scanf`

Karakter	Opis
d	Učitava se celobrojna vrednost u dekadnoj notaciji; odgovarajući argument ukazuje na promenljivu tipa <code>int</code> ; ako je ispred karakter <code>d</code> naveden modifikator <code>i</code> ili <code>h</code> , tada argument ukazuje na promenljivu tipa <code>long</code> ili <code>short</code> , respektivno.
o	Učitava se celobrojna vrednost u oktalnoj notaciji; odgovarajući argument ukazuje na promenljivu tipa <code>int</code> ; ako je ispred karaktera naveden modifikator <code>h</code> ili <code>i</code> , tada argument ukazuje na promenljivu tipa <code>short</code> ili <code>long</code> , respektivno.
x	Učitava se celobrojna vrednost u heksadecimalnoj notaciji; odgovarajući argument ukazuje na promenljivu tipa <code>int</code> ; ako je ispred karaktera <code>x</code> naveden modifikator <code>h</code> ili <code>i</code> , tada argument ukazuje na promenljivu tipa <code>short</code> ili <code>long</code> , respektivno.
D, O, X	Ekvivalentni formatima <code>ld</code> , <code>lo</code> i <code>lx</code> , respektivno. Argument iz liste ukazuje na promenljivu tipa <code>long int</code> .
f	Učitava se realan broj; broj može biti predhodan znakom minus ili plus i/ili izražen u naučnoj notaciji; odgovarajući argument

	ukazuje na promenljivu tipa float; ako karakter i predhodi karakteru f, argument ukazuje na promenljivu tipa double.
e	Identičan konverzionom karakteru f.
F, E	Ekvivalentni formati if i ie; odgovarajući argument ukazuje na promenljivu tipa double.
c	Učitava se karakter; odgovarajući argument ukazuje na promenljivu tipa char; opcionalni broj ispred karaktera c, a posle % specificira broj karaktera, koji se učitava; u tom slučaju odgovarajući argument ukazuje na vektor tipa char.
s	Učitava se niz karaktera; niz počinje prvim vidljivim karakterom, a završava prvim nevidljivim karakterom; odgovarajući argument ukazuje na vektor tipa char, u kojem mora biti dovoljno mesta za upisivanje završnog karaktera (scanf automatski dodaje završni karakter); ako je specificirana širina polja, učitava se broj karaktera jednak navedenoj širini ili do prvog nevidljivog karaktera.
[...]	Karakter uokvireni uglastim zagradama pokazuju da treba biti učitani niz karaktera; karakteri između zagrada su legalni karakteri u nizu, koji se učitava; ako se bilo koji drugi karakter

pronade u ulaznom nizu, učitavanje se završava, jer se tretira kao nelegalan; ako je prvi karakter u uglastim zagradama strelica '^', tada isključivo navedeni karakteri mogu završiti učitavanje, odnosno, ako se bilo koji od navedenih karaktera pronade u ulaznom nizu, učitavanje niza karaktera se završava.

Funkcija scanf završava učitavanje, kada se učitava broj karaktera jednak specificiranoj širini polja ili ako karakter nije legalan za tip učitane vrednosti. U slučaju celih brojeva, legalni karakteri su znak minus i plus i cifre, i to: 0-9 za dekadnu, 0-7 za oktalnu, 0-9 i a-f ili A-F za heksadecimalnu notaciju. Za brojeve tipa float legani karakteri su znak minus ili plus i sekvenca dekadnih cifara, decimalna tačka i druga sekvenca dekadnih cifara. Brojevi tipa float mogu biti specificirani u naučnoj notaciji. Za razdvajanje eksponenta od stalnog dela broja, koriste se karakteri e ili E. Za nizove karaktera legalni su svi vidljivi karakteri. Međutim, za format c dozvoljeni su svi karakteri (vidljivi i nevidljivi). U slučaju korišćenja uglastih zagrada, legalni su isključivo karakteri u uglastim zagradama. Ako je strelica 'Č' prvi karakter u uglastim zagradama, legalni su svi karakteri, koji nisu navedeni u uglastim zagradama.

Karakter navedeni u konverzionom nizu i koji se ne nalaze između karaktera '%' i legalnih konverzionih karaktera, očekuje se u ulaznom nizu i ne učitavaju se. Na primer, iskaz

```
scanf("%d:%d:%d", &sat, &minut, &sekund);
```

učitava tri dekadne celobrojne vrednosti, koje se memorišu u promenljive sat, minut i sekund. U konverzionom nizu nalazi se karakter ':', koji igra ulogu separatora između tri celobrojne vrednosti. Ako se karakter '%' očekuje u ulaznom nizu, tada se navodi dvostruki karakter '%'. Na primer,

```
scanf("%d%%", &procenat);
```

učitava celobrojnu vrednost, iza koje sledi znak procentat. Navođenjem blanka u konverzionom nizu preskače se proizvoljan broj nevidljivih karaktera u ulaznom nizu.

Funkcija `scanf` vraća kao rezultat broj uspešno obavljenih konverzija. Vrednosti EOF (simboličko ime) se vraća, ako je pronađen marker kraja datoteke (\*engl. `end_of_fille`). Najčešće EOF ima vrednost -1. Vrednost 0 se vraća, ako nijedna konverzija nije uspešna i ova vrednost je uvek različita od EOF. Ako u ulaznom nizu postoji nelegalan karakter, konverzija se prekida, a rezultat funkcije `scanf` je vrednost 0. Učitavanje se ne završava, dok se ne "istroše" svi konverzioni karakteri u konverzionom nizu. Ako u ulaznom nizu nije specificiran dovoljan broj vrednosti, funkcija `scanf` "čeka" ulaz preostalih vrednosti. Na primer, pozivom

```
scanf("%d, %*s%c%%5s, &i, &c, niz);
```

i sa specificiranim nizom

```
123 , cd w % abcdefg
```

vrednost 123 dodeljuje se promenljivoj `i`, preskače zarez, karakter `'w'` dodeljuje promenljivoj `c`, preskače znak procenta i niz "abcde" smešta u vektor niz tipa `char`. Pošto su uspešno obavljene tri konverzije `scanf` vraća vrednost.

### 4.15.3. Upravljanje datotekama

C biblioteka sadrži veliki broj funkcija za rad sa datotekama. Pri izvršavanju ovih funkcija datoteka se tretira kao sekvencijalni niz karaktera. Datotekama se pristupa korišćenjem ukazatelja na strukturu `FILE`, koja je definisana u include datoteci `stdio.h`. Struktura `FILE` sadrži članove, koji opisuju tekuće stanje datoteke. Datoteka `stdio.h` se uključuje u program iskazom `#include`. U `stdio.h` definiše se i konstante EOF i NULL. Konstanta EOF označava kraj datoteke i najčešće ima vrednost -1. Konstanta NULL ima vrednost 0 i vraća se kao rezultat nekih funkcija za upravljanje datotekama u slučaju neuspešnog izvršavanja funkcija.

#### 4.15.3.1. Funkcija `fopen()`

Datoteka, pre bilo kakvog procesiranja mora biti otvorena, pozivanjem odgovarajuće funkcije i specificiranjem imena datoteke. Sistem ispisuje da li postoji datoteka pod tim imenom i pod određenim uslovima stvara (ili kreira) novu datoteku pod specificiranim imenom. Takođe se, prilikom otvaranja datoteke, mora specificirati tip željene ulazno/izlazne operacije. Na primer, ako se želi pročitati sadržaj datoteke, specificira se režim čitanja (engl. `read mode`); ako se neki sadržaj upisuje u datoteku, specificira se režim upisa (engl. `write mode`); ako se želi dodati neki sadržaj, specificira se režim dodavanja (engl. `append mode`). U režimu upisa i dodavanja sistem stvara novu datoteku, ako datoteka pod specificiranim imenom ne postoji. Kako se u programu može otvoriti veći broj datoteka, tada se u funkcijama željena datoteka identifikuje



ukazateljem na struktuiranu promenljivu tipa FILE. Ukazatelj na strukturu FILE postavlja se prilikom otvaranja datoteke.

Funkcija fopen iz C biblioteke koriste se za otvaranje datoteka. Zaglavalje funkcija fopen je

```
FILE*fopen(file_name, file_mode)
char*file_name, *file_mode;
```

Funkcija fopen ima dva argumenta: file\_name i file\_mode tipa ukazatelj na char. Argument file\_name ukazuje na niz sa imenom datoteke, dok file\_mode ukazuje na niz sa režimom otvaranja. Tabela 13.5. opisuje tri moguća režima otvaranja

Tabela 13.5. Tri režima za otvaranje datoteke

Karakter	Režim otvaranja
r	čitanje
w	upisivanje
a	dodavanje

fopen vraća ukazatelj na strukturu FILE, kojim se datoteka identifikuje u drugim funkcijama za rad sa datotekama. Ako je otvaranje neuspešno iz bilo kojih razloga, fopen vraća vrednost NULL. Na primer, sledeći sled iskaza

```
#include <stdio.h>
...
FILE*in_file, *fopen();
...
in_file=fopen("file", "w");
```

otvara datoteku pod imenom file 1 u režimu upisa. fopen vraća ukazatelj na strukturu FILE, koji se dodeljuje promenljivoj in\_file istog tipa. U slučaju neuspešnog otvaranja fopen, vraća vrednost NULL. Iz tog razloga neophodno je ispitati vraćenu vrednost, odnosno, dodati iskaz

```
if (in_file == NULL)
printf("Datoteka file 1 ne moze biti otvorena\n");
```

Pozivanje funkcije fopen i ispitivanje vraćenog rezultata može se objediniti u jedan if iskaz

```
if (in_file=fopen("file 1", "w")) == NULL)
printf("Datoteka file1 ne mo'e biti otvorena\n");
```

U gornjem iskazu if poziva se fopen, vraćeni rezultat funkcije se dodeljuje promenljivoj in\_file i ispituje da li vrednost promenljive jednak NULL. Podsećamo da operator == ima više prioritet u odnosu na operator =.

### 4.15.3.2. Funkcije fprintf() i fscanf()

Funkcije fprintf i fscanf obavljaju analogne operacije, kao i funkcije printf i scanf, s tom razlikom što fprintf i fscanf zahtevaju dodatni argument za identifikaciju datoteke u koju se upisuje ili iz koje se čitaju podaci. Opša forma ovih funkcija je

```
fprintf(file_pointer, konverzioni_niz, lista_argumenata)
fscanf(file_pointer, konverzioni_niz, lista_argumenata)
```

u kojoj su: file\_pointer ukazatelj na tip FILE za identifikaciju datoteke; konverzioni\_niz format po kome se upisuju ili čitaju podaci; lista\_argumenata lista vrednosti koje se upisuju u datoteku ili ukazatelja na promenljive u koju se smeštaju podaci iz datoteke identifikovana ukazateljem file\_pointer. fprintf i fscanf imaju promenljiv broj argumenata. Na primer, iskazom

```
fprintf(out_file, "Niz se upisuje u datoteku\n");
```

u datoteku identifikovanu ukazateljem out\_file upisuje se niz "Niz se upisuje u datoteku\n". Slično, iskazom

```
fprintf(out_file, "Vrednost promenljive je %f\n", x);
```

u datoteku identifikovanu ukazateljem out\_file upisuje se specificirani niz zajedno sa vrednošću promenljive x tipa float.

Efekat funkcije fscanf je inverzan. Funkcija fscanf učitava podatke iz specificirane datoteke po određenom formatu. Na primer, pozivom

```
fscanf(in_file, "%f", &x);
```

iz datoteke identifikovane ukazateljem in\_file učitava se vrednost po formatu %f i smešta u promenljivu x.

Funkcije fprintf i fscanf imaju identične konverzije karaktere i modifikatore kao i printf i scanf.

### 4.15.3.3. Funkcija feof()

Funkcija feof ispisuje da li je dosegnut kraj datoteke. Zaglavlje funkcije feof je

```
int feof(file_pointer)
FILE *file_pointer;
```

gde je file\_poinetr ukazatelj za identifikaciju datoteke. Funkcija feof vraća nenultu vrednost, ako je dosegnut kraj datoteke. U protivnom, vraća vrednost 0. Na primer, u iskazu

```
if (feof(in_file))
    printf("Kraj datoteke\n");
```

štampa se poruka "Kraj datoteke", ako su pročitani svi podaci iz datoteke in\_file.

**4.15.3.4. Funkcija fclose()**

Funkcijom fopen otvaramo datoteku i time omogućavamo njeno procesiranje u programu. Ukazatelj na strukturu FILE povezuje naš program i željenu datoteku. Kada se procesiranje završi, vezu između programa i datoteke treba raskinuti. Ovo raskidanje naziva se zatvaranje datoteke, koje se postiže pozivom funkcije fclose. Zatvorenoj datoteci ne možemo pristupiti bez ponovnog otvaranja. Zaglavlje funkcije fclose je

```
int fclose(file_close)
FILE *file_pointer;
```

gde je: file\_pointer ukazatelj za identifikaciju datoteke. fclose vraća vrednost EOF ako ukazatelj nije povezan sa nekom datotekom. U programu 13.2. funkcija fclose nije pozvana, zato što sistem automatski zatvara sve datoteke prilikom završavanja programa. Međutim, program rada optimalnije, ako ima manji broj otvorenih datoteka (VAX-11 i VMS ograničava na 20 istovremeno otvorenih datoteka). fclose se poziva iskazom:

```
fclose(in_file);
```

gde je in\_file ukazatelj na strukturu FILE.

Program 4.69. ilustruje primenu nekih definisanih funkcija za rad sa datotekama. U programu 4.69. otvara se tekstualna datoteka i njen sadržaj ispisuje na ekran terminala. Ako izvršni program, u procesu prevodenja, nazovemo tupe, tada se program 4.69. startuje komandom `type infile` gde je infile ime datoteke, čiji sadržaj prikazujemo na ekranu terminala. Ime datoteke se prenosi kao argument u funkciju main. Sadržaj datoteke se čita funkcijom fgets, pri čemu je maksimalna dužina niza (linije) 80 karaktera, a štampa funkcijom printf. Kada fgets vrati vrednost NULL, dosegnut je kraj datoteke i prikazivanje sadržaja je završeno.

**Program 4.69.**

```
/*Program za stampanje i sadrzaja datoteka*/
#include <stdio.h>
#define LINE_LENGTH 81
main(argc, argv)
    int argc;
    char *argv[];
    {
    char buffer[LINE_LENGTH];
    FILE *in_file, *fopen();
    char *fgets();
    if (argc !=2)
        printf("\nFormat k-de: %s name_of_file\n", argv[0]);
    else
        if ((in_file=fopen(argv[1], "r"))= =NULL)
            printf("Datoteka %s ne moze biti otvorena\n", argv[1]);
        else
            while(fgets(buffer, LINE_LENGTH, in_file) !=NULL)
                printf("%s", buffer);
    }
```

**Program 4.69. Izlaz**

type prog.c

**4.15.3.5. Funkcija exit()**

U nekim slučajevima javlja se potreba za završavanje programa pre fizičkog kraja programa, na primer, pri detekciji uslova, koji zahtevaju završetak programa, ranije je rečeno, da se program implicitno završava posle zadnjeg iskaza u funkciji main. Funkcija exit eksplicitno završava program na mestu gde se poziva. Na primer, iskaz

exit(n);

završava izvršavanje programa u kome se nalazi. Otvorene datoteke zatvaraju se automatski. Celobrojne promenljive n nazivaju se uslovni kod (engl. condition code). U UNIX-u je usvojena konvencija da n=0 označava uspešan završetak programa, a n#0 neuspešan završetak. Vrednost uslovnog koda može biti ispitivano u drugim programima u cilju određivanja da li je izvršavanje programa uspešno ili neuspešno.

**4.15.3.6. Grafička biblioteka**

PC računar ima neku od video adaptera (ili kartica). To mogu biti Monochrome Display Adapter (MDA) za osnovno prikazivanje teksta ili adapter za prikazivanje grafike, kao što su Color Graphlcs Adapter, Hercules Monochrome Graphlcs Adapter (Hercules) ili Enhanced Graphlcs Adapter (EGA). Svaki od adaptera može raditi u tekstualnom (40 ill 80 kolona) ili grafičkom načinu rada i biti u nekom tipu prikazivanja (boja, monohromatski ili crno&belo).

U tekstualnom načinu rada ekran je podeljen u ćelije (80 ill 40 kolona širine sa 25 linija široko). Svaka ćelija se sastoji od atributa i karaktera. Karakter je prikazani ASCII karakter na ekranu, dok atribut specificira kako se karakter prikazuje (boju, intenzitet. itd). Gornji levi ugao na ekranu ima koordinate (1, 1 ) sa x-koordinatom, koja se povećava sa leva na desno, i y-koordinatama, koje se povećavaju odozgo na dole.

U grafičkom načinu rada, ekran je podeljen u tačke (piksele). Svaki piksel prikazuje jednu tačku na ekranu. Broj piksela (rezolucija) zavisi od tipa video adaptera, koji je u PC računaru i od izabranog načina rada. U grafičkom modu gornji levi ugao ima koordinate (0,0) sa x- i y-koordinatama, koje se povećavaju na isti način kao i kod tekstualog načina rada.

Turbo C uvodi pojmove prozora (window), vidika(viewport) i koordinata. Prozor je pravougaona oblast na ekranu u tekstualnom načinu rada. Kada, koristeći funkcije Turbo C, ispisuje nešto na ekranu, izlaz je ograničen na aktivan prozor. Ostali deo ekrana ostaje nepromenjen. Vidik je pravougaona oblast na ekranu u grafičkom načinu rada. Kada grafički program crta na ekranu, izlaz je ograničen samo na aktivan vidik. Ostali deo ekrana (van vidika) ostaje nepromenjen.

Svaka ćelija ili piksel, u skladu sa predhodnim, ima svoje x- i y-koordinate.

***Tekstualni način rada***

Funkcije Turbo C omogućavaju rad u jednom od pet mogućih tekstualnih načina rada (BW40, C40, BW80, C80 i MONO). Ove funkcije omogućavaju

izlaz i manipulaciju tekstem (tabela G.5), kontrolu prozora i načina rada (tabela G.6), kontrolu atributa (tabela G.7) i upite o stanju (tabela G.8).

Tabela G.5 Turbo C -Funkcije za izlaz I manipulaciju tekstem

Funkcija	Opis funkcije
	Prikazivanje ili učitavanje teksta
cprintf	Prikazivanje formatizovanog teksta
cputs	Prikazivanje niza
putch	Prikazivanje karaktera
getche	Učitavanje karaktera i eho na ekran
	Manipulacija teksta na ekranu
clrscr	Brisanje prozora
creol	Brisanje od kursora do kraja linije
delline	Unistava liniju
gotoxy	Pozicioniranje kursora
insline	Insertovanje linije
movetext	Kopiranje teksta
	Pomeranje teksta u ili iz memorije
gettext	Kopiranje teksta sa ekrana u memoriju
puttext	Kopiranje teksta iz memorije na ekran

Tabela G.6 Turbo C -Funkcije za kontrolu prozora i načina rada

Funkcija	Opis funkcije
textmode	Setovanje ekrana u tekstualni mod
window	Definicija prozora

#### *Grafički način rada:*

Grafičke funkcije Turbo C se dele u sedam kategorija:

1. Kontrola grafičkog sistema (tabela G.9),
2. Crtanje i popunjavanje (tabela G.10),
3. Manipulacija ekranima i vidicima (tabela G.11),
4. Izlaz teksta (tabela G.12),
5. Kontrola boje (tabela G.13),
6. Upravljanje greškama (tabela G.14),
7. Upit o stanju (tabela G.15).

Tabela G.7 Turbo C -Funkcije za kontrolu atributa

Funkcija	Opis funkcije
Setovanje prvog plana i pozadine	
textcolor	Setovanje boje prvog plana
textbackground	Setovanje boje pozadine
textattr	Setovanje boje prvog plana i pozadine
Setovanje intenziteta	
highvideo	Setovanje na povećani intenzitet
lowvideo	Setovanje na smanjeni intenzitet
normvideo	Setovanje na normalan intenzitet

TABELA G.8 Turbo C - Funkcije za upit o stanju

Funkcija	Opis funkcije
gettextinfo	Upit o tekućem prozoru
wherex	Upit o x-koordinati kursora
wherey	Up it o y-koordinati kursora

Tabela G.9 Turbo C -Funkcije za kontrolu grafičkog sistema

Funkcija	Opis funkcije
closegraph	Obaranje graličkog sistema
detectgraph	Određivanje graličkog drajvera
graphdefaults	Resetovanje graličkih promenljivih
_graphfreemem	Oslobađanje grafičke memorije
_graphgetmem	Dodeljivanje grafičke memorije
getgraphmode	Vraćanje tekućem grafičkom načinu
getmoderange	Informacije o najnižem i najvišem
initgraph	Inicijalizacija graličkog sistema
registerbgdriver	Specificiranje grafičkog drajvera
restorecrtmode	Obnavlja originalni način rada ekrana
setgraphbufersize	Specifikacija veličine grafičkog bafera
setgraphmode	Selekcija grafičkog načina rada

Tabela G.10 Turbo C -Funkcije za crtanje i popunjavanje

Funkcija	Opis funkcije
	Crtanje
arc	Crtanje kružnog luka
circle	Crtanje kruga
drawpoly	Crtanje poligona
ellipse	Crtanje eliptičnog luka
getarccoords	Koordinate zadnjeg poziva arc i ellipse
getaspectratio	Aspect ratio tekućeg grafičkog moda
getlinescttings	Informacija o liniji
line	Crtanje linije
linereel	Crtanje linije relativno
lineto	Crtanje linije od tekuće pozicije
moveto	Pomeranje kursora
movercl	Pomeranje kursora relativno
rectange	Crtanje pravougaonika
setllnestyle	Setovanje karakteristika linije
	Popunjavanje
bar	Crtanje i popunjavanje pravougaonika
bar3d	Crtanje i popunjavanje 3-D kvadra
fillpoly	Crtanje i popunjavanje poligona
floodfill	Crtanje i popunjavanje regiona
getfillpattern	Korisnički definisan uzorak popune
getfillsettings	Informacija o tekućem uzorku
pieslice	Crtanje i popunjavanje pie slice
setfillpatern	Selekcija oblika za popunjavanje
setfillstyle	Setovanje uzorka i boje za popunjavanje

Tabela G. 11 Turbo C , Manipulacija ekranima i vidicima

Funkcija	Opis funkcije
	Manipulacija ekranima
cleardevice	Brisanje ekrana
setactivepage	Postavljanje aktivne starnice
setvisualpage	Postavljanje broja stranice

## Manipulacija vidicima

clearviewport	Brisanje tekućeg vidika
getviewsettings	Informacija o tekućem vidiku
setviewport	Postavljanje tekućeg vidika

## Manipulacija slikom

getimage	Memorisanje bitne mape ekrana
imagesize	Broj bajtova potreban za memorisanje pravougaonog regiona na ekranu
putimage	Prikazivanje ranije memorisanog regiona

## Manipulacija pikselima

getpixel	Boja piksela na koordinati (x,y)
putpixel	Prikazivanje piksela na koordinati x

Tabela G.12 Turbo C -Prikazivanje teksta

Funkcija	Opis funkcije
gettextsettings	Vraca tekući font, smer, veličinu i poravnanja
outtext	Prikaz linija na tekuću poziciju
outtextxy	Prikaz teksta na poziciju (x,y)
registerbfont	Korisnički font za uključivanje u toku povezivanja
settextjustify	Specificiranje izgleda teksta
settextstyle	Specificiranje fonta, stila, i dr. teksta
setusercharsize	Specificiranje širine i visine fonta
textheight	Visina fonta u pikselima
textwidth	Širina fonta u pikselima



Tabela G.13 Turbo C -Kontrola boja

Funkcija	Opis funkcije
Informacije o boji	
getbkcolor	Boja pozadine
getcolor	Boja za crtanje
getmaxcolor	Maksimalna vrednost boje
getpalette	Tekuća paleta i veličina
Specificiranje boje	
setallpalette	Promena boje palete
setbkcolor	Specificiranje boje pozadine
setcolor	Specificiranje boje crtanja
setpalette	Promena jedne boje palete

Tabela G. 14 Turbo C -Upravljanje greskama

Funkcija	Opis funkcije
grapherrormsg	Poruka greške za specificirani kod
graphresult	Kod greške za zadnju grafičku operaciju

Tabela G.15 Turbo C -Upit o stanju

Funkcija	Opis funkcije
getarcoords	Koordinate zadnjeg poziva arc i ellipse
getaspectratio	Aspect odnos grafikog ekrana
getbkcolor	Boja pozadine
getcolor	Boja crtanja
getfillpatern	Oblik za popunjavanje
getfillsettings	Oblik i boja za popunjavanje
getgraphmode	Tekući grafički način rada
getlinessettings	Tekući stil, oblik i debljina linije
getmaxcolor	Najveća ispravna vrednost piksela
getmaxx	x-rezolucija
getmaxy	y-rezolucija
getmoderange	Opseg rada drajvera
getpalette	Tekuća paleta i veličina
getpixel	Boja piksela na (x, y)
gettextsettings	Tekući font, smer, veličina i poravnanje
getviewsettings	Informacije o tekućem vidiku
getx	x-koordinata tekuće pozicije
gety	y-koordinata tekuće pozicije

Heder file	What It Does
alloc.h	Declares memory management functions (allocation, deallocation, etc.).
assert.h	Defines the assert debugging macro.
bcd.h	Declares the C class bcd and the overloaded operators for bcd and bcd math functions.
bios.h	Declares various functions used in calling IBM-PC ROM BIOS routines.
complex.h	Declares the C complex math functions.
conio.h	Declares various functions used in calling the DOS console I/O routines.
ctype.h	Contains information used by the character classification and character conversion macros.
dir.h	Contains structures, macros, and functions for working with directories and path names.
direct.h	Defines structures, macros, and functions for working with directories and path names.
dirent.h	Declares functions and structures for POSIX directory operations.
dos.h	Defines various constants and gives declarations needed for DOS and 8086-specific calls.
errno.h	Defines constant mnemonics for the error codes.
fcntl.h	Defines symbolic constants used in connection with the library routine open.
float.h	Contains parameters for floating-point routines.
fstream.h	Declares the C stream classes that support file input and output.
generic.h	Contains macros for generic class declarations.
io.h	Contains structures and declarations for low-level input/output routines.
iomanip.h	Declares the C streams I/O manipulators and contains macros for creating parameterized manipulators.
iostream.h	Declares the basic C (version 2.0) streams (I/O) routines.
limits.h	Contains environmental parameters, information about compile-time limitations, and ranges of integral quantities.
locale.h	Declares functions that provide country- and language- specific information.
math.h	Declares prototypes for the math functions, defines the macro HUGE_VAL, and declares the exception structure used by matherr.
mem.h	Declares the memory-manipulation functions. (Many of these are also defined in string.h.)
memory.h	Memory manipulation functions.
new.h	Access to operator new and newhandler.
process.h	Contains structures and declarations for the spawn and exec functions.
search.h	Declares functions for searching and sorting.
setjmp.h	Defines a type used by longjmp and setjmp.
share.h	Defines parameters used in functions that use file-sharing.
signal.h	Defines constants and declarations for signal and raise.
stdarg.h	Defines macros used for reading the argument list in functions declared to accept a variable number of arguments.
stddef.h	Defines several common data types and macros.

---

stdio.h	Defines types and macros needed for the Standard I/O Package defined in Kernighan and Ritchie and extended under UNIX System V. Defines the standard I/O predefined streams stdin, stdout, stderr, and declares stream-level I/O routines.
stdiostr.h	Declares the C stream classes for use with stdio FILE structures.
stdlib.h	Declares several commonly used routines: conversion routines, search/sort routines, and other miscellany.
string.h	Declares several string and memory-manipulation routines.
strstrea.h	Declares the C stream classes for use with byte arrays in memory.
sys\locking.h	Definitions for mode parameter of locking function.
sys\stat.h	Defines symbolic constants used for opening and creating files.
sys\timeb.h	Declares the function ftime and the structure timeb that ftime returns.
sys\types.h	Declares the type time_t used with time functions.
time.h	Defines a structure filled in by the time-conversion routines, and a type used by other time routines also provides prototypes for these routines.
utime.h	Declares the functions utime and the structure utimbuf
values.h	Defines important constants, including machine dependencies provided for UNIX System V compatibility.
varargs.h	Defines old style macros for processing variable argument lists.

## # INCLUDE <MATH.H> omogućuje upotrebu funkcija:

abs	log, logl
acos, acosl	log10, log10l
asin, asinl	matherr, _matherrl
atan, atanl	modf, modfl
cabs, cabsl	poly, polyl
cos, cosl	pow, powl
cosh, coshl	pow10, pow10l
exp, expl	sin, sinl
fabs, fabsl	sinh, sinhl
floor, floorl	sqrt, sqrtl
fmod, fmodl	tan, tanl
frexp, frexpl	tanh, tanhl
hypot, hypotl	
labs	
ldexp, ldexpl	



**PRAKTIKUM ZA LABORATORIJSKE VEŽBE  
IZ PROGRAMSKIH JEZIKA I**



Viša tehnička škola  
Niš

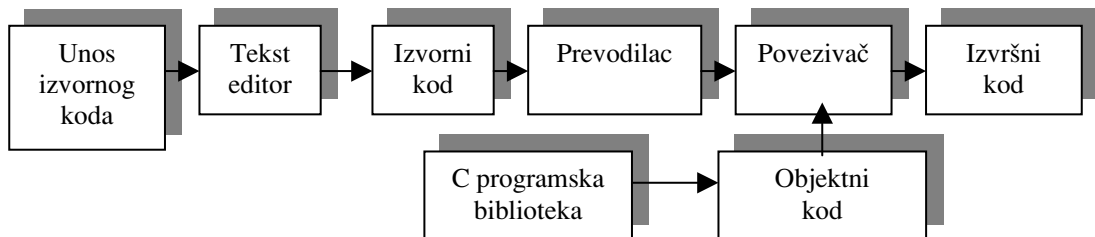
## PRVA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

### 1. Postupci kreiranja programa u DOS - u

Za unos izvornog koda u C jeziku na operativnom sistemu DOS može biti upotrebljen jedan od mnogih dostupnih tekst editora ( Edlin, Norton editor, Edit, Q editor, ... ) ili neki od dostupnih tekst procesora. Kod procesora teksta je važno, da se izvorni kod programa snimi u ASCII formatu, jer su dodatne informacije koje se odnose na oblikovanje i prelom teksta u tekst procesorima nerazumljivi za prevodilac.

Na softverskom tržištu su prisutni i editori specijalno dizajnirani za programiranje, kao naprimer Microsoft Editor ( ME ) ili Word Perfektov- Program Editor. Pored navedenih, postoje i editori koji su ugrađeni u integrisane razvojne okoline: Borlandov Turbo C i Microsoft Quick C.

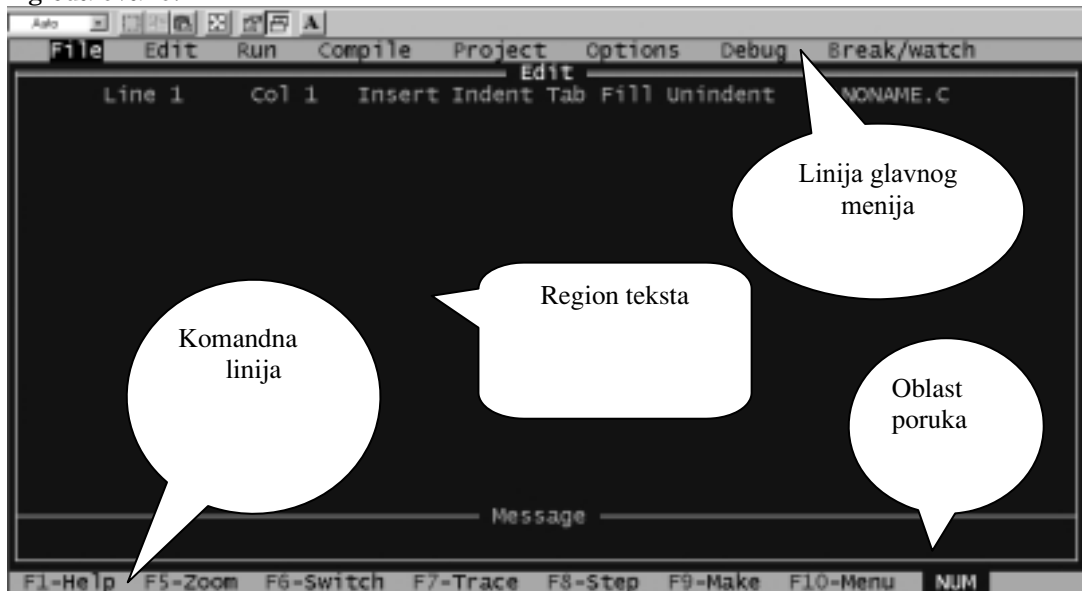
Imena svih datoteka se pod operativnim sistemom DOS interno tretiraju kao da su napisana velikim slovima i sastoje se od kombinacije osam karaktera kojom bliže definišemo namenu programa, dok je ekstenzija izvornog programa .c . Proces prevođenja C programa pod operativnim sistemom DOS može se predstaviti kao:



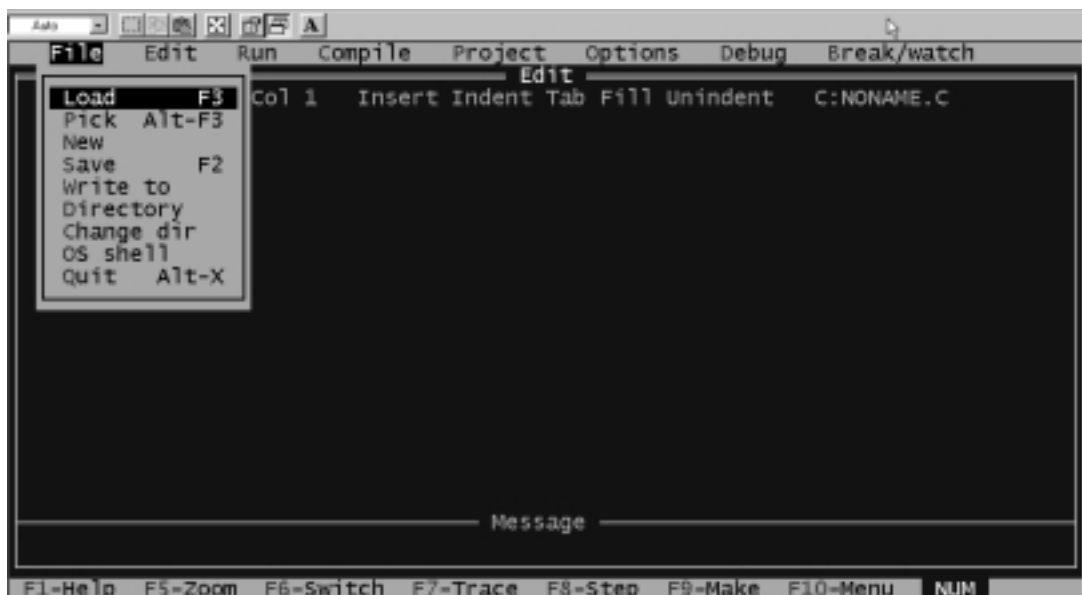
Proces prevođenja se realizuje pozivom prevodioca koji najpre kreira objektnu datoteku čije ime je identično sa osnovnim imenom izvorne datoteke, ali se sada ekstenzija menja u .obj . Zatim se vrši povezivanje programa čiji se poziv realizuje naredbom link kada se objektnom kodu dodaje bibliotečki kod i kod za početno povezivanje sa operativnim sistemom. Na taj način se kreira izvršna datoteka, koja sada ima ekstenziju .exe , a osnovno ime je identično - kao kod objektne i izvorne datoteke . DOS prevodilac ne briše objektnu datoteku, tako da će one ostati na tekućem direktorijumu zajedno sa izvornim kodom i izvršnom datotekom. Ako je proces prevođenja i linkovanja bio uspešan, tj. u programu nije pronađena ni jedna greška, možemo krenuti sa izvršavanjem programa kucajući samo njegovo ime. Ako je pak prilikom prevođenja dojavljena greška, opisana oznakom programske linije i njenim tipom, moramo se vratiti u editor i ispraviti načinjene greške pa zatim pokrenuti ponovo postupak kompajliranja i linkovanja.

Borlandovo Turbo C razvojno okruženje poseduje ugrađeni editor koji se može upotrebiti za pisanje C programa. Preko sistena padajućih menija omogućeno je imenovanje i snimanje izvorni kodova i realizacija postupaka stvaranja izvršnog koda bez napuštanja integrisanog okruženja. Pozivanje ovog okruženja može se izvršiti sa

Desk topa startovanjem ikonice Turbo C DOS kada će se otvoriti osnovni prozor koji izgleda ovako:



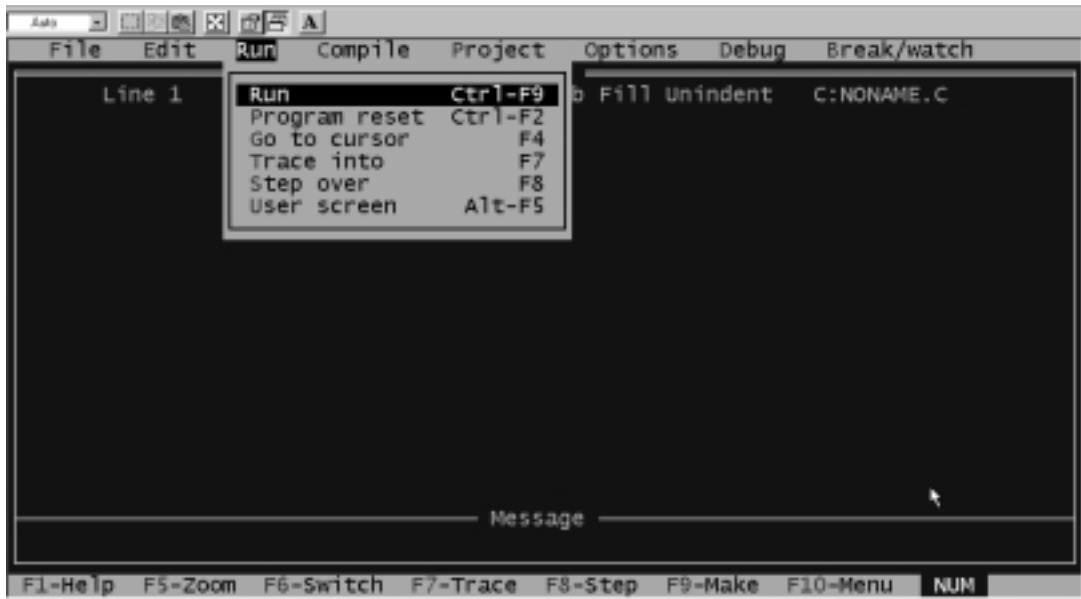
Korišćenjem opcije File omogućeno je punjenje Load datoteke izvornog koda, otvaranje nove datoteke New, zapisivanje editovane datoteke na disk Save, zapisivanje editovane datoteke na disk pod drugim imenom Write to, sagledavanje tekućeg direktorijuma Directory, promena tekućeg direktorijuma Change dir i izlaz iz Turbo C-a komandom Quit.



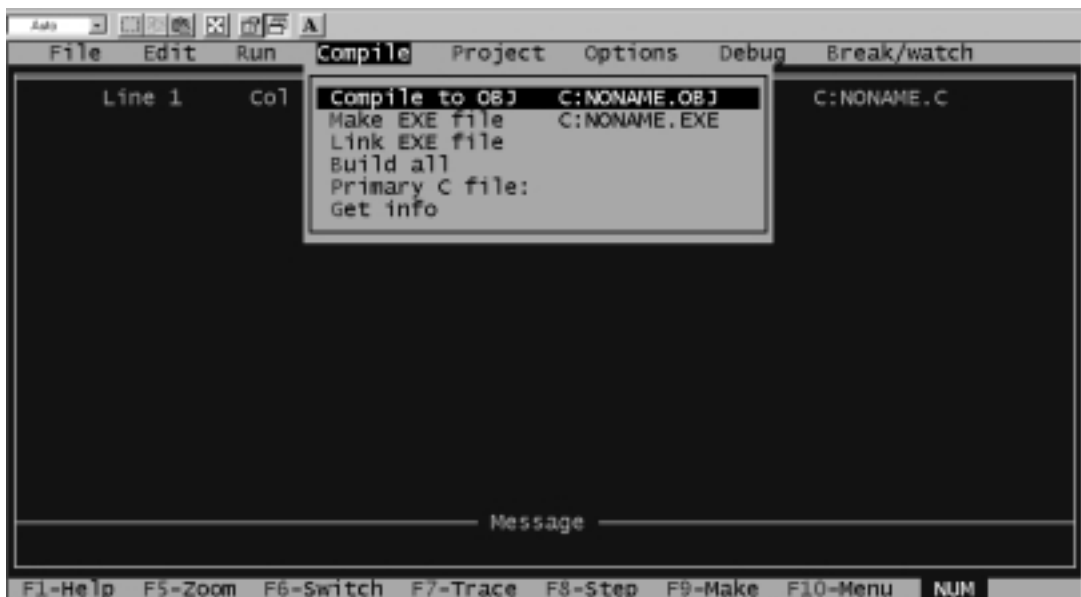


Pokretanjem opcije Edit iz glavnog menija ulazimo u mod za editovanje - pisanje izvornog programa C jezika.

Pokretanjem opcije Run iz glavnog menija ulazimo u mod za startovanje - pokretanje izvršnog programa i sagledavanje rezultata njegovog rada.



Pokretanjem opcije Compile iz glavnog menija ulazimo u mod za generisanje objektnog koda Compile to OBJ, u mod za automatsko kompajliranje i linkovanje programa Make EXE file i mod za linkovanje programa Link EXE file.



Pokretanjem opcije Debug iz glavnog menija ulazimo u mod za dibging programa, odnosno mod za postupno - automatsko otkrivanje grešaka.

2. Sada možemo započeti sa unosom programa u izvornom kodu, sa njegovim kompajliranjem i linkovanjem i naravno, ukoliko nema grašaka, i sa njegovim izvršenjem.

Napišimo jednostavan C program, koji na terminalu ili CRT ekranu ispisuje poruku "Moj prvi program u C jeziku" i uočimo osnovne gradivne elemente .

```
main()
```

```
{  
printf("Moj prvi program u C jeziku\n");  
}
```

Kada smo završili sa unosom programa u izvornom kodu pristupamo njegovom kompajliranju i linkovanju korišćenjem funkcijskog tastera F9 - Make. Vrlo je važno sagledati sve podatke koji se posle pokretanja opcije Make pojavljuju u izveštajnom prozoru, jer od toga zavisi da li je ovaj postupak ispravno izveden - odnosno da li je napisani program ispravan. U slučaju da je postupak uspešan - successful - i nema grešaka - no errors - , možemo pokrenuti izvršavanje programa opcijom iz glavnog menija Run. Rezultate rada programa možemo videti u korisničkom prozoru koji otvaramo istovremenim pritiskanjem kombinacije tastera Alt i F5 na funkcijskoj tastaturi. Kada smo prepisali sve rezultate rada programa povratak u naš Turbo C izvršićemo pritiskom na taster F5 na funkcijskoj tastaturi.

3.Napišimo jednostavan C program, koji izračunava razliku dva cela broja 100 i 90.

```
main()
```

```
{  
int razlika;  
razlika = 100 -90;  
printf("Razlika 100 -90= %d\n", razlika);  
}
```

Primenjujući već opisani postupak za kompajliranje i linkovanje dobijamo rezultat rada programa:

**Razlika 100 -90= 10**

Viša tehnička škola  
Niš

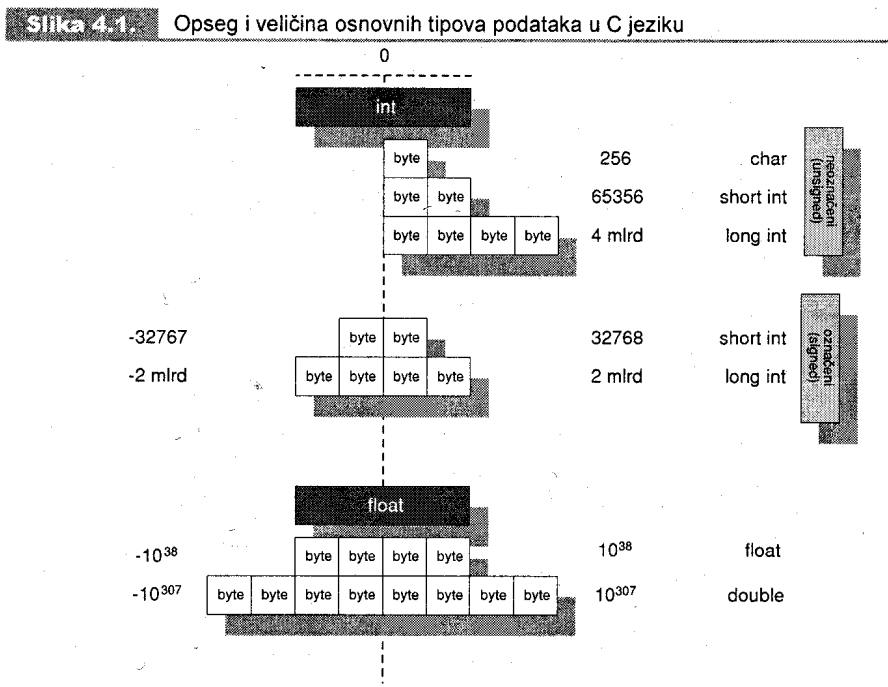
**DRUGA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I**

1. Upoznavanje sa osnovnim tipovima podataka i deklaracijom podataka u programskom jeziku C.

**Tabela 4.1.** Osnovni tipovi podataka u C jeziku

Oznaka tipa	Opis
char	Običan znak
signed char	Označeni znak (sa aritmetičkim izrazom)
unsigned char	Nenegativan znak
short int	Kratak ceo broj
int	Običan ceo broj (najčešće veličine registra)
long int	Dugačak ceo broj
unsigned short int	Nenegativan kratak ceo broj
unsigned int	Nenegativan običan ceo broj
unsigned long int	Nenegativan dugačak ceo broj
float	Realni broj obične preciznosti
double	Realni broj dvostruke preciznosti
long double	Realni broj proširene preciznosti

Osnovni opsezi i veličina podataka razlikuju se od računara do računara i uglavnom zavise od arhitekture procesora.



Konverzacioni znak	Opis
<code>%c</code>	Jedan karakter znakovnog tipa
<code>%d</code>	Označeni ceo broj
<code>%e</code>	Broj u pokretnom zarezu, e -notacija
<code>%E</code>	Broj u pokretnom zarezu, E -notacija
<code>%f</code>	Broj u pokretnom zarezu, decimalna not.
<code>%i</code>	Označeni ceo broj
<code>%X</code>	Neoznačeni heksadecimalni broj (A do F)
<code>%x</code>	Neoznačeni heksadecimalni broj (a do f)
<code>%p</code>	Pokazivač
<code>%s</code>	Karakterni niz

U programskom jeziku C direktno je moguća primena osnovnih matematičkih operadora `+`, `-`, `*`, `/` i operadora koji ne postoje u drugim programskim jezicima: inkrement i dekrement. Ostale matematičke operatore možemo uključiti iz skupa datoteka zaglavlja koji sadrži datoteku `<math.h>` i koja se po potrebi u procesu strukturnog programiranja uključuje u izvorni program naredbom `#include`.

Aritmetički operatori u C jeziku	Primedba
Sabiranje ( <code>+</code> )	
Oduzimanje ( <code>-</code> )	
Množenje ( <code>*</code> )	
Deljenje ( <code>/</code> )	Ostatak pri deljenju se gubi
Moduo ( <code>%</code> )	Ostatak celobrojnog deljenja
Inkrement ( <code>++</code> )	Efekti primene izraza su različiti
Dekrement ( <code>--</code> )	Efekti primene izraza su različiti

`#include <stdio.h>` zaglavlje programa - uključuje datoteku zaglavlja `stdio.h` koja je deo paketa C kompajlera i sadrži informacije o ulazno/izlaznim funkcijama za računar. Naziv potiče od STanDard Imput/Output. Informacije iz datoteke koje se koriste prilikom prevodenja izvornog programa, neće postati deo izvršnog programa, čime se on neće povećati.

2. Sastaviti program na programskom jeziku C za upotrebu specifikatora konverzije u odnosu na očekivane vrednosti celobrojnih promenljivih.

```
#include <stdio.h>
main()
{
    unsigned neoznaceni = 39000;
    long velik = 2000000000;
    unsigned long jako_velik = 2 * 2000000000;
    short mali_broj = 350;
    printf("neoznaceni = %u, i nije %d\n", neoznaceni, neoznaceni);
    printf("velik = %ld, i nije %d\n", velik, velik);
    printf("jako_velik = %lu, i nije %d\n", jako_velik, jako_velik);
    printf("mali_broj = %hd, i na ovom sistemu je %d\n", mali_broj, mali_broj);
}
```

Izlaz iz programa je:

```
neoznaceni=39000, i nije -25536
velik=2000000000 i nije -27648
jako_velik=4000000000 i nije 10240
mali_broj=200 i na ovom sistemu je 200
```

3. Sastaviti program na programskom jeziku C za upotrebu funkcije rand() kojom se uključuje generator slučajnih brojeva i štampa celobrojni slučajni broj i njegova dvostruka vrednost.

```
#include <stdio.h>
#include <math.h>
main()
{
  int slucajan, dvostruki;
  slucajan = rand();
  dvostruki = slucajan*2;
  printf("Slucajan broj je %d\n", slucajan);
  printf("Dvostruki slucajan broj je %d\n", dvostruki);
  printf("Naredni slucajan broj je %d\n", rand());
}
```

Prepišite izlaz iz programa:

4. Sastaviti program na programskom jeziku C za formatiranje ulaza u pokretnom zarezu.

```
#include <stdio.h>
main()
{
  /* deklaracija podataka */
  float f_pro;
  double d_pro;
  /*dodela vrednosti*/
  f_pro=106.11;
  d_pro=-0.0000654;
  /*štampanje vrednosti promenljivih */
  printf ("Promenljiva f_pro=%2f\n", f_pro);
  printf ("Promenljiva d_pro=%.11f\n", d_pro);
  printf ("Promenljiva f_pro=%e\n", f_pro);
  printf ("Promenljiva d_pro=%G\n", d_pro);
}
```

Izlaz iz programa je:

Promenljiva f\_pro=106.110001

Promenljiva d\_pro=-0.00006540000

Promenljiva f\_pro=1.061100e+02

Promenljiva d\_pro=-6.54E-05

5. Sastaviti program na programskom jeziku C za unos proizvoljnog karaktera i za štampanje ASCII koda tog karaktera.

```
#include<stdio.h>
main()
{
char ch;
printf("Unesite proizvoljan karakter.\n");
scanf("%c",&ch); /*naredba za unos podataka*/
printf("ASCII kod unetog karaktera %c je %d\n",ch,ch);
}
```

*Prepišite izlaz iz programa:*

Viša tehnička škola  
Niš

### TREĆA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

1. Napisati program na C jeziku za štampanje dva cela broja x i y u obliku njihovog inkrementa i dekrementa.

Napomena: Operator inkrementa se koristi da bi se povećala vrednost promenljive za jedan. Operator dekrementa se koristi da bi se smanjila vrednost promenljive za jedan. Pravila po kojima se primenjuje postfixni i prefiksni zapis operatora dekrementa na celobrojne promenljive i izraze su potpuno analogna pravilima upotrebe operatora inkrementa i mogu se sagledati iz navedenog primera.

```
main()
{
int x,y;
x=10;
y=10;
printf("Vrednost izraza ++x je %d\n",++x);
printf("Vrednost izraza y++ je %d\n",y++);
printf("Nakon inkrementiranja vrednost za x je %d\n",x);
printf("Nakon inkrementiranja vrednost za y je %d\n",y);
printf("Vrednost izraza --x je %d\n",--x);
printf("Vrednost izraza y-- je %d\n",y--);
printf("Nakon dekrementiranja vrednost za x je %d\n",x);
printf("Nakon dekrementiranja vrednost za y je %d\n",y);
}
```

Upisati rezultate rada programa :

2. Napisati program na C jeziku za rešavanje kvadratne jednačine  $x^2 + x - 2 = 0$  korišćenjem obrasca:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

i štampanje vrednosti za x1 i x2.

```
#include<stdio.h>
#include<math.h>
main()
{
int a,b,c;
float x1,x2,pom;
a=1;
b=1;
c=-2;
pom=sqrt(b*b-4*a*c);
x1=(-b+pom)/2*a;
x2=(-b-pom)/2*a;
printf("Vrednost prvog korena x1=%fn",x1);
printf("Vrednost drugog korena x2=%fn",x2);
}
```

Upisati rezultate rada programa :

2.1. Napisati isti program ali u opštem obliku kada se veličine a, b i c učitavaju sa tastature.

Napisati izvorni kod programa:



3. Napisati program na C jeziku za izračunavanje ukamaćene vrednosti ako je poznata kamatna stopa, period oročavanja i iznos glavnice. Štampati dobijene rezultate i proveriti matematičku postavku programa.

Napomena: Koristiti pow funkciju iz biblioteke math.h koja izračunava  $x$  na  $y$  ( $x^{**}y$ )

Njena deklaracija je:

- Real
- double pow(double x, double y);
  - long double pow(long double (x), long double (y));
- Complex
- complex pow(complex x, complex y);
  - complex pow(complex x, double y);
  - complex pow(double x, double y);

```
#include <stdio.h>
#include <math.h>
main()
{
double stopa, period, glavnica;
printf( "Unesite kamatnu stopu: " );
scanf( "%lf", &stopa ) ; /* ulaz u pokretnom zarezu */
/* konverzija u procenite */
stopa = stopa / 100.0;
/* kamata se izracunava mesecno */
stopa= stopa / 12.0;
printf( "Unesite glavnicu: " );
scanf( "%lf", &glavnica );
printf( "Unesite vreme orocavanja: " );
scanf( "%lf", &period ) ;

/*Stopa je konvertovana na mesecnu osnovicu*/
period = period * 12.0;
printf( "Ukamacena vrednost je = %.2f\n",
glavnica * pow( (1.0+stopa), period));
}
```

Proveriti izlaz iz programa:

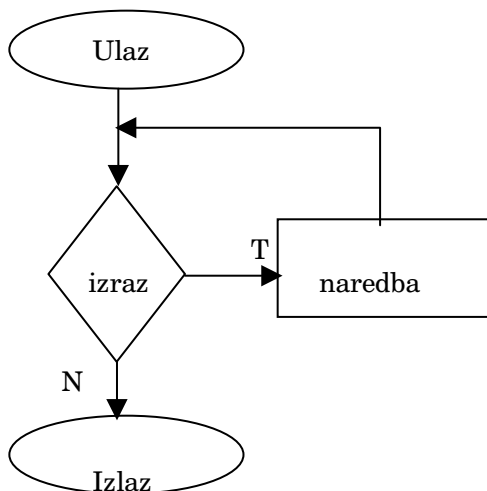
```
/*Programski kod ima sledece ulazno/izlazne poruke i vrednosti:*/

/*Unesite kamatnu stopu:8.5*/
/*Unesite glavnicu: 1000.00*/
/*Unesite vreme orocavanja u godinam:3.0*/
/*Ukamacena vrednost je:1289.30*/
```

Viša tehnička škola  
Niš

## ČETVRTA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

### 1. Osnovna petlja sa ulaznim uslovom: *while*



Upravljačku strukturu *while* karakterišu četiri osobine:

- ◆ petlja se ponavlja sve dok *izraz* koji se testira ne postane netačan ( logička 0 );
- ◆ petlja je sa *ulaznim uslovom* : odluka o još jednom prolasku kroz telo petlje se donosi pre izvršenja naredbi petlje;
- ◆ *while* petlja mora sadržavati promenu vrednosti *izraza* tako da postane lažan posle određenog broja iteracija, kako petlja ne bi postala beskonačna;
- ◆ jedna naredba se posmatra kao deo upravljačke strukture *while* petlje, bilo ona prosta ili složena.

1.1 Napisati program na C jeziku za izračunavanje zbira celih brojeva upotrebom *while* petlje.

```

#include <stdio.h>
main()
{
long num, sum = 0L;
int status;
printf("Unesite broj za sumiranje\n");
printf("Ili q za izlaz\n");
status = scanf("%ld", &num);
while (status==1) /*bez ; na kraju ako sledi blok naredbi i sa ; na kraju kao jedna naredba*/
{

```

```

sum = sum+num;

printf("Unesite naredni broj za sabiranje\n");
printf("Unesite q za quit!\n");
status=scanf("%ld", &num);
}
printf("Zbir unetih brojeva je %ld.\n", sum);
}

```

Napomena: Prvi broj za sabiranje ne sme biti 1, zašto?

Upisati rezultate rada programa :

2. Napisati program u C jeziku za permutovanje cifara celog broja - while iskaz: (npr. 54321 u 12345 )

Postupak:

celi broj 54321 delimo celobrojno sa 10	- rezultat je 1
celi broj 54321 delimo sa 10	- rezultat je 5432
celi broj 5432 delimo celobrojno sa 10	- rezultat je 2
celi broj 5432 delimo sa 10	- rezultat je 543
celi broj 543 delimo celobrojno sa 10	- rezultat je 3
celi broj 543 delimo sa 10	- rezultat je 54
celi broj 54 delimo celobrojno sa 10	- rezultat je 4
celi broj 54 delimo sa 10	- rezultat je 5
celi broj 5 delimo celobrojno sa 10	- rezultat je 5

Petlja se prekida jer broj postaje nula !!!

```

main()
{
int broj;
printf("Ukucajte ceo broj ? ");
scanf("%d", &broj );
printf("Permutovani broj je ");
while ( broj )
{
printf("%d", broj % 10);
broj = broj/10;
}
printf("\n");
}

```

**Programski izlaz**

Ukucajte ceo broj? 8612

Permutovani broj je 2168

3. Napisati program na C jeziku za određivanje srednje vrednosti n celih pozitivnih brojeva -while iskaz.

```
main()
{
int n, brojac = 0;
float suma = 0, x;
printf("Ukupno brojeva ? ");
scanf("%d", &n);
while (brojac < n)
{
printf("Ukucajte %d. broj ? ", brojac+1);
scanf("%f", &x);
suma += x;
brojac += 1;
}
printf("Srednja vrednost ovih brojeva je %f\n", suma/n);
}
```

**Programski izlaz**

Ukupno brojeva ? 4

Ukucajte 1. broj ? 34.345

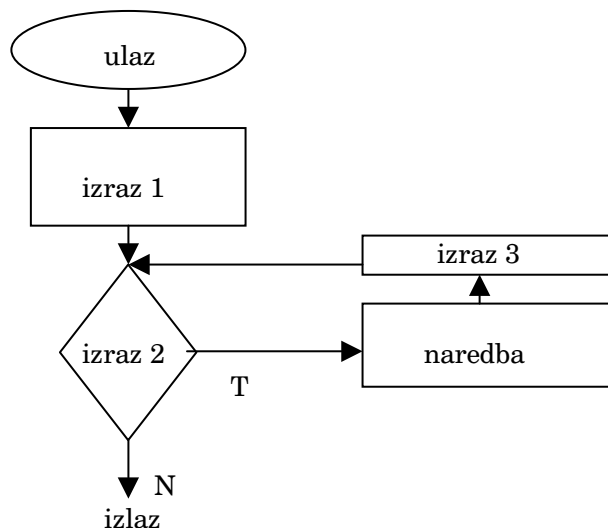
Ukucajte 2. broj ? 234.2876

Ukucajte 3. broj ? 45.98

Ukucajte 4. broj ? 453.32

Srednja vrednost ovih brojeva je 191.983154

4. Generalna petlja sa ulaznim uslovom: *for*



Upravljačke informacije kod for petlje, za razliku od while petlje, su smeštene na jednom mestu i to na vrhu iteracije. Petlja for je definisana, što znači da je njena osnovna konstrukcija sledeća:

- ◆ inicijalizacije vrednosti brojača;
- ◆ komparacija brojača sa limitirajućom vrednošću;
- ◆ ažuriranje brojača pri svakoj iteraciji.

Prema tome, opšti oblik naredbe for je: for ( izraz1, izraz2, izraz3 )  
naredba

uz napomenu da neki od izraza u for petlji može ostati prazan.

4.1 Napisati program na C jeziku za prikaz neparnih brojeva manjih ili jednakih 20 korišćenjem for petlje.

```
#include <stdio.h>
main()
{
  int broj;
  for (broj=1;broj<=20;broj=broj+2)
  printf("\n%d",broj);
}
```

5. Proveriti upravljačku strukturu *if* naredbe:

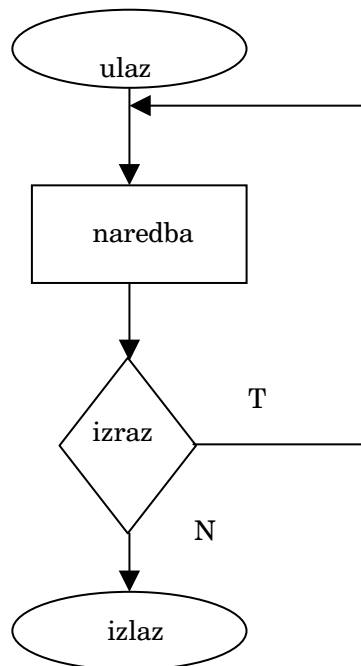
```
#include <stdio.h>
main()
{
  int x, y, z;
  x = 10;
  y = 3;
  z = (x / y) * y;
  if (x == z)
  {
    printf("Vrednosti x-a i z-a su jednake\n");
    printf("Vrednost x-a je %d\n", x);
    printf("Vrednost z-a je %d\n", z);
  }
  if (x <= z)
  {
    printf("Vrednost x-a je < ili = od vrednosti z-a\n");
    printf("Vrednost x-a je %d\n", x);
    printf("Vrednost z-a je %d\n", z);
  }
  if (x >= z)
  {
    printf("Vrednost x-a je > ili = od vrednosti z-a\n");
    printf("Vrednost x-a je %d\n", x);
    printf("Vrednost z-a je %d\n", z);
  }
}
```

Viša tehnička škola  
Niš

## PETA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

### 1. Petlja sa ulaznim uslovom *do while* :

U slučaju *do while* petlje obavezno izvršavanje bar jedne iteracije se postiže tako što je upravljački izraz petlje na samom dnu petlje. Tako se uslov petlje proverava nakon izvršenih naredbi tela petlje. Strukturni dijagram toka može se prikazati kao:



#### 1.1. Napisati program za izračunavanje $n!$ primenom *do while* strukture

```

/* Program za izracunavanje faktorijela */
main()
{
  int i, fak;
  long n ;
  i = 1; n = 1;
  printf("Izracunavanje n!\nUkucajte broj ? ");
  scanf("%d", &fak);
  do {
    n *= i;
    i++;
  } while (i <= fak);
  printf("%d! = %ld\n", fak,n);
}

```

2. Bezuslovno grananje : *break* i *continue*

Prekid izvršenja naredbi tela petlje kontrolnih upravljačkih struktura se može realizovati i pre nego što uslovni izraz dobije logičku netačnu vrednost. Naredba kojom je moguće realizovati iskakanje iz upravljačke strukture se realizuje korišćenjem ključne reči *break*. Ova naredba inicira sledeća dva procesa:

- ♦ Prekid izvršenja naredbi tela petlje u upravljačkim strukturama *while*, *for* i *do while* iskakanjem iz tela petlje. Programski tok se nastavlja neposredno na prvoj naredbi iza naredbe upravljačke strukture.
- ♦ Preskakanje preostalih naredbi unutar višestrukog grananja, koje se ostvaruju naredbom *switch*, skakanjem na prvu naredbu iza naredbe selekcije.

2.1 Primer programa u kome se koristi naredba *break* u kombinaciji sa naredbom *switch* za izračunavanje broja samoglasnika u delu proizvoljnog teksta:

```
#include <stdio.h>
main()
{
    char ch;
    int a_ct,e_ct,i_ct,o_ct,u_ct;
    a_ct=e_ct=i_ct=o_ct=u_ct=0;
    printf("Unesi priozvoljan tekst; Unesi # za izlaz.\n");
    while((ch=getchar())!='#')
    {
        switch (ch)
        {
            case 'a' :
            case 'A' : a_ct++;
                break;
            case 'e' :
            case 'E' : e_ct++;
                break;
            case 'i' :
            case 'I' : i_ct++;
                break;
            case 'o' :
            case 'O' : o_ct++;
                break;
            case 'u' :
            case 'U' : u_ct++;
                break;
            default:
                break;
        }
        /* kraj switch */
    }
    /* dok petlji nije kraj */
    printf("Broj samoglasnika: A E I O U\n");
    printf("          %4d %4d %4d %4d %4d\n",
           a_ct,e_ct,i_ct,o_ct,u_ct);
}
```

2. Sastaviti program na C jeziku za rešavanje sistema linearnih jednačina primenom Kramerovog pravila:

$$x_1 = \frac{\Delta_1}{\Delta}, x_2 = \frac{\Delta_2}{\Delta}, x_3 = \frac{\Delta_3}{\Delta}, \dots, x_n = \frac{\Delta_n}{\Delta}$$

i praktično rešiti sistem jednačina:

$$\begin{aligned} x_1 + x_2 + 2x_3 &= 9 \\ 4x_1 - x_2 + 3x_3 &= 11 \\ -2x_1 + 2x_2 + x_3 &= 5 \end{aligned}$$

gde je :

$$\Delta = \begin{vmatrix} 1 & 1 & 2 \\ 4 & -1 & 3 \\ -2 & 2 & 1 \end{vmatrix}, \Delta_1 = \begin{vmatrix} 9 & 1 & 2 \\ 11 & -1 & 3 \\ 5 & 2 & 1 \end{vmatrix}, \Delta_2 = \begin{vmatrix} 1 & 9 & 2 \\ 4 & 11 & 3 \\ -2 & 5 & 1 \end{vmatrix}, \Delta_3 = \begin{vmatrix} 1 & 1 & 9 \\ 4 & -1 & 11 \\ -2 & 2 & 5 \end{vmatrix}$$

Proveriti rešenja ovog sistema linearnih jednačina za:  $x_1=1$ ,  $x_2=2$  i  $x_3=3$



Viša tehnička škola  
Niš

## ŠESTA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

### 1. Potprogrami i struktura programa

Potprogrami predstavljaju mehanizam koji direktno podržava funkcionalnu dekompoziciju kao jednu od osnovnih metoda strukturnog programiranja. Oni su samostalni segmenti programskog koda koji se pozivaju radi obavljanja konkretno specificiranog zadatka. Potprogram može biti neograničeno puta pozivan iz istog ili u okviru različitih programa. Programski jezik C sadrži module u kojima se nalaze grupe deklaracija promenljivih i deklaracija potprograma kao jedinice fizičke dekompozicije. Ovakav pristup za direktnu posledicu ima stvaranje opštih programskih rešenja i mogućnost višestrukog korišćenja istog programskog koda u različitim softverskim proizvodima. Svaki potprogram se sastoji od:

- ◆ deklaracije kojom se definiše interfejs potprograma prema programu, koja uključuje ime potprograma, listu parametara koja ne mora biti nepravna i opciono tip vraćene vrednosti
- ◆ deklaraciju lokalnih promenljivih koje su dostupne samo naredbama tela potprograma
- ◆ tela potprograma koje se sastoji od deklaracija i naredbi.

Postoje dva tipa potprograma:

**FUNKCIJA:** je segment programskog koda koji na osnovu nijednog, jednog ili više argumenata, uvek daje kao rezultat jednu tačno određenu vrednost koja se naziva vrednost funkcije.

**PROCEDURA:** je tip potprograma koji takođe može prihvatiti nijednu, jednu ili više vrednosti u obliku argumenata. Na osnovu prihvaćene vrednosti rezultat procedure može biti nijedna ili više vrednosti.

U deklaraciji ovakvog tipa funkcija, što važi generalno i za procedure, koristi se rezervisana reč void kao oznaka za vraćene vrednosti.

Definicija funkcije u opštem slučaju ima sledeći oblik:

```

vra}eni_tip    ime_funkcije    ( lista_argumenata )
                {
                deklaracija ( a ) .....
                naredba
                .....
                naredba_povratka
                }

```

Izvršavanje se može prekinuti naredbama return ili exit.

### 1.1 Poziv funkcija bez argumenata

```
#include<stdio.h>
#define IME "Visa tehnicka skola"
#define ADRESA "Beogradska 20"
#define MESTO "18000 Nis"
#define LIMIT 65
main()
{
    void zvezde(void); /*deklaracija funkcije bez argumenata*/
    zvezde(); /*poziv korisnicke funkcije*/
    printf("%s \n",IME); /*poziv funkcije iz standardne biblioteke*/
    printf("%s \n",ADRESA);
    printf("%s \n",MESTO);
    zvezde();
}
/*definicija korisnickih funkcija*/
void zvezde() /*potprogramska funkcija nema argumenata*/
{
    int brojac;
    for(brojac=1;brojac<=LIMIT;brojac++)
        putchar('*');
        putchar('\n');
}
```

1.2 Sagledati tok glavnog programa i potprograma i utvrditi u kojoj programskoj liniji nastaju " tačke prekida " glavnog programa kada se kontrola prebacuje na izvršavanje potprograma. Napisati rezultate rada programa:



1.3 Koristeći isti program umesto konstanti IME, ADRESA i MESTO ubaciti stvarne podatke programera. Kompajlirati, linkovati i izvršiti ovako modifikovan program.

2. Sastaviti program na C jeziku za izračunavanje minimuma dva cela broja. U glavnom programu obezbediti štampanje oba broja i njihovog minimuma, a u potprogramu izraziti samu funkciju minimuma.

```
#include<stdio.h>
main()/* glavni program */
{
    int broj1,broj2,vrati;
    int imin(int,int);
    vrati=scanf("%d %d",&broj1,&broj2);
    if(vrati==2)
        printf("Manji od %d i %d je %d\n",broj1,broj2,imin(broj1,broj2));
}
int imin(n,m)/* potprogram */
int n,m;
{
    int min;
    if (n<m)
        min=n;
    else
        min=m;
    return min;
}
```

3.Sastaviti program korišćenjem struktura IF...THEN....ELSEIF za izračunavanje funkcije  $y$ . U glavnom programu obezbediti štampanje rezultata a u potprogramu definisati vrednosti funkcije po određenim uslovima. Brojevi su integer tipa.

$$y = \begin{cases} x_1 + x_2 & x_1 < x_2 \\ x_1 * x_2 & x_1 = x_2 \\ x_1 - x_2 & x_1 > x_2 \end{cases}$$

Napisati program

Viša tehnička škola  
Niš

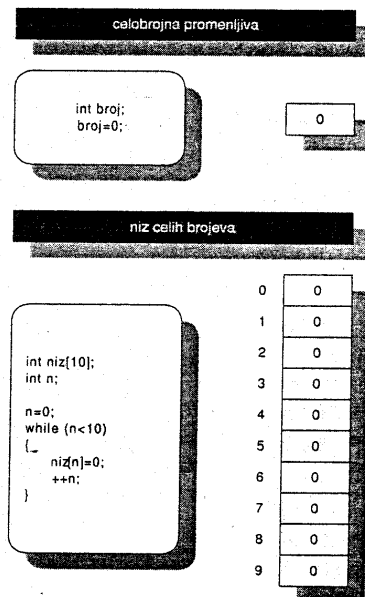
## SEDMA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

### 1. Izvedeni tipovi podataka - nizovi

Niz je homogena struktura u kojoj su svi objekti istog tipa. Objekti u nizu se nazivaju elementi niza. Tip niza predstavlja implicitnu konsekvencu tipa svojih elemenata. Korišćenjem indeksa u nizu se veoma lako pretražuju homogeni tipovi podataka, vrši se njihovo sortiranje ili prestruktuiranje elemenata na neki drugi način. Nizovi mogu biti: nizovi celih, nizovi realnih brojeva, nizovi pokazivača, nizovi karaktera, nizovi korisničkih tipova itd.

Opšti oblik deklaracije niza je: `tip ime_niza [veličina]`

Deklaracija i eksplicitna dodela vrednosti celobrojnoj promenljivoj u nizu.



U programskom jeziku C, se saglasno vrstama memorijskih klasa koje se navode pri deklaraciji nizova,- razlikuju četiri vrste inicijalizacije nizova, od toga su prve dve implicitne, a poslednje dve eksplicitne.

1. Nizovi koji su deklarirani kao spoljašnje promenljive su istovremeno i globalne promenljive, koje se prema tome automatski inicijalizuju na nulu. Ovakva inicijalizacija se naziva automatskom inicijalizacijom.

2. Deklaracijom niza kao statičke promenljive se realizuje automatska inicijalizacija svih članova niza na nulu, ukoliko se eksplicitno ne zadaju vrednosti članovima niza prilikom deklaracije. Ovakva inicijalizacija se naziva statička inicijalizacija niza, koja obezbeđuje i zadržavanje poslednjih vrednosti za sve elemente niza prilikom izlaska izopsega dejstva funkcije.

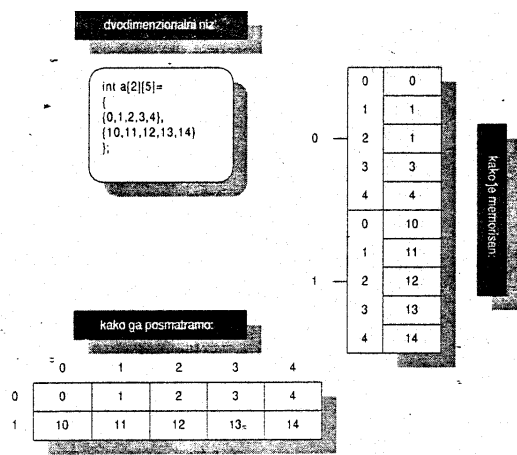
3. Prosledivatljive podataka za inicijalizaciju koji se koriste sa naredbom za deklaraciju niza.

4. Prosledivanje podataka za elemente niza tokom izvršenja naredbi iz toka programa. Vrednosti se u ovakvoj vrsti inicijalizacije prenose u elemente niza dodeljivanjem ili kopiranjem.

Primer organizovanja dvodimenzionalnog niza bi bio:

## 2. Pokazivači

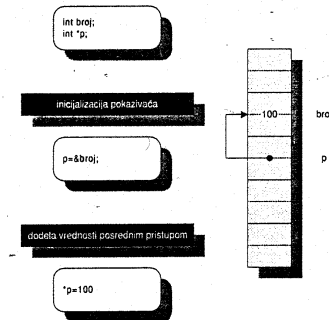
Pokazivač je promenljiva koja sadrži adresu objekta ili funkcije. Rad sa pokazivačkim promenljivama je veoma teško razumeti bez razumevanja koncepta indirekcije, odnosno koncepta posrednog pristupa. Objekti programskog rešenja se uvek



memorišu u određenom programskom bloku, i to na tačno određenoj memorijskoj lokaciji. Pokazivači omogućuju indirektni pristup vrednostima programskih promenljivih korišćenjem njihovih memorijskih adresa. U neposrednoj vezi sa pokazivačima su dva specijalna operatora programskog jezika. operator adresa-od u oznaci & je operator koji daje memorijsku adresu objekta na koga je primenjen; posredni operator \* je operator kojim se upravlja memorijskim lokacijama upokazivačkim promenljivama.

Adresni operator & je unarni operator vrlo visokog prioriteta, što je slučaj i sa ostalim unarnim operatorima. Ovaj operator predstavlja način da se sazna memorijska lokacija gde se objekat smešten u memoriji računara. Asocijativnost ovog unarnog operatora je s desna na levo, a opseg vrednosti koji može da poprimi je skup nenegativnih celih brojeva (unsigned int). Operator indirekcije \* omogućava deklarisanje pokazivačke promenljive i indirektni pristup do vrednosti za koje su korišćene pokazivačke promenljive. U razvoju programskih rešenja se vrlo često informacija o memorijskoj adresi promenljive ili funkcije čuva i kasnije koristi. Za implementaciju ovog koncepta je neophodno obezbediti mesto u memoriji gde se smešta

informacija o adresi programskog objekta, koju dobijamo deklarisanjem pokazivačke promenljive. Paradigma pokazivačkih promenljivih u C jeziku bi izgledala:



Pokazivači se često koriste u sprezi sa nizovima kako bi se lakše dolazilo do određenog člana niza što predstavlja adekvatnu upotrebu aritmetičkih operacija nad pokazivačima. Dozvoljene operacije su sabiranje i oduzimanje celobrojnih podataka, dodela vrednosti jednog pokazivača drugom, poređenje dva pokazivača, poređenje pokazivača i nule kao celobrojne vrednosti, oduzimanje vrednosti pokazivača istog tipa.

3. Uraditi program za prikaz niza i množenje elemenata niza konstantnom vrednošću.

```
#include <stdio.h>
#define VELICINA 5
void prikaz_niza(double niz[], int n);
void uvecaj_niz(double broj, double *pok, int vel);
void main()
{
    static double dip[VELICINA] = {20.0, 17.66, 8.2, 15.3, 22.22}; prikaz_niza(dip, VELICINA);
    uvecaj_niz(2.5, dip, VELICINA);
    prikaz_niza(dip, VELICINA);
}
void prikaz_niza(ar, n)
double ar[];
int n;
{
    int i;
    putchar('\n');
    for(i = 0; i < n; i++)
        printf("%8.3f ", ar[i]);
    putchar('\n');
}

void uvecaj_niz(mnozi, ar, granica)
double mnozi;
double *ar;
int granica;
{
    int i;
    for(i = 0; i < VELICINA; i++)
        *ar++ *= mnozi;
}
```

4. Primer prenosa argumenata funkcije pokazivačkim promenljivim za realizaciju sabiranja članova niza.

```
#define VELIC 10
long sump(int *pok, int vel) ;
void main()
{
static int niz[VELIC] = {20,10,5,39,4,16,19,26,31,20};
long odqovor;
odgovor = sump(niz, VELIC);
printf("Suma niza je: %ld.\n", odqovor);
}
long sump(ar, n) /* koristi pokazivacku aritmetiku */
int *ar; /* ar je pokazivac */
int n;
{
int i ;
long ukupno = 0;
for(i = 0; i < n; i++)
{
ukupno += *ar; /* dodaje vrednost na ukupno */
ar++; /* pomera pokazivac na sledeci elemenat */
}
return ukupno;
}
```

– Pokazivači kao argumenti funkcija, omogućuju između ostalog i vraćanje više vrednosti iz pozivajuće funkcije. Naredni primer programskog koda ilustruje eophodnost ovakvog pristupa za jednostavnu operaciju zamene vrednosti dve promenljive.

5. Sastaviti program za prenos argumenata pozivom po vrednosti.

```
#include <stdio.h>
void promeni(int p , int q) ;
void main ( )
{
int x = 5, y = 10;
printf("U programu oriqinalne vrednosti: x = %d a y = %d.\n", x, y) ;
promeni(x,y) ;
printf("U programu nakon poziva funkcije je: x = %d a
y = %d.\n", x, y) ;
}
void promeni(int u,int v)
{
int temp;
printf("U funkciji originalne vrednosti su: u = %d a
v= %d.\n", u, v);
temp=u;
u=v;
v=temp;
printf("U funkciji nakon promene vrednosti su: u = %d a v= %d.\n", u, v);
}
```

Napisati rezultate rada programa:





Viša tehnička škola  
Niš

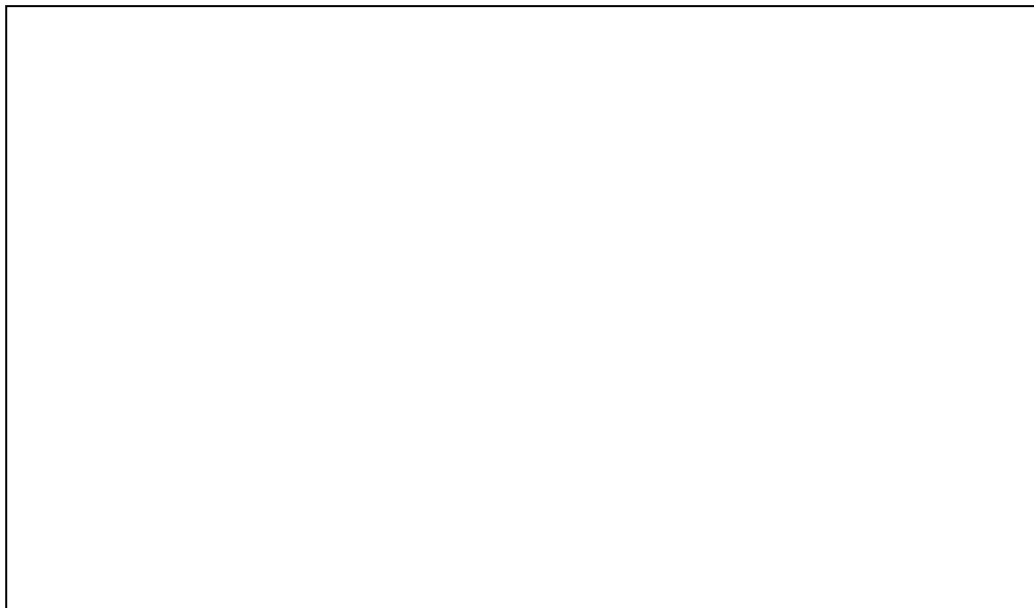
## OSMA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

8.1. Razmotriti rad programa pisanog u C programskom jeziku, koji sortira dati vektor  $ar=\{7,3,9,2,11,20,17,19,6,4\}$  korišćenjem famozne metode "bubble sort" algoritma. Program se najpre sastoji u pisanju funkcije "swap" koja svapuje vrednosti i smešta ih u dve memorijske lokacije. Zatim tu funkciju koristimo da bi izvršili sortiranje datog vektora. "Bubble sort" ispituje svake dve ćelije vektora i ako nisu sortirane - svapuje ih. Ako se obezbedi upoređenje i svapovanje n-1 put nad svim ćelijama vektora, on će biti kompletno sortiran.

```
#include <stdio.h>
void swap (int *a, int *b);
main()
{
    int ar[10],i,j,n;
    ar[0]=7,ar[1]=3,ar[2]=9,ar[3]=2,ar[4]=11;ar[5]=20;ar[6]=17;ar[7]=19;ar[8]=6;ar[9]=4;
    printf("Vektor pre sortiranja:\n\n");
    for(i=0;i<10;i++)
        printf("ar[%d]=%d\n",i,ar[i]);
    n=10;          /*broj članova u vektoru koji se sortira*/
    for(i=0;i<n-1;i++)
        for(j=0;j<n-1;j++)
        {
            if(ar[j]>ar[j+1])
                swap(&ar[j],&ar[j+1]);
        }
    printf("Vektor posle sortiranja:\n\n");
    for(i=0;i<10;i++)
        printf("ar[%d]=%d\n",i,ar[i]);
    return 0;
}

void swap ( int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

Prepisati rezultate rada programa:



8.2. Sastaviti program na programskom jeziku C za ispisivanje tablice ASCII kodova za sve štampanje znake.

ASCII kod sa brojem 32 je blanko simbol ili razmak. Prvih 32 ASCII kodova i svi ASCII kodovi veći ili jednaki 127 predstavljaju specijalne simbole koji se koriste u različite svrhe i najčešće nemaju neko jasno slovno značenje. Koristili su se za iscrtavanje DOS prozora, i na taj način simulirali grafiku u tekstualnom režimu rada. Konkretno prozori razvojnog okruženja Turbo C su napravljeni na ovaj način! ASCII kodovi između 33 i 126 su štampanje znaci.

Znaci se prikazuju po kolonama. U jednoj koloni se prikazuje po 19 znakova, a u jednom redu se prikazuje po 5 znakova ( $5 \times 19 = 95$ ). Prikazaćemo sve ASCII kodove od 32 do 126.

```
#include <stdio.h>
main()
{
    char c=' ';
    int i;

    printf ("\t\tTablica ASCII kodova \n \n");
linija:
    i=0;
    znak:
    printf ("%3d %c   ",c+i,c+i);
    i=i+19;
    if (i<95) goto znak;
    printf ("\n");
    c=c+1;
    if (c<' '+19) goto linija;
}
```

8.3 Sastaviti program na programskom jeziku C koji učitava srednje temperature po mesecima za 12 meseci i na osnovu njih izračuna i ispiše srednju temperaturu za celu godinu.

```
#include <stdio.h>
#define BROJ_MESECI 12
main()
{
    enum meseci {JAN=1,FEB,NAR,APR,MAJ,JUN,JUL,AVG,SEP,OKT,NOV,DEC};
    enum meseci mesec=JAN;
    /* Moguce je umesto gornje dve linije napisati samo jednu:
    enum meseci {JAN=1,FEB,NAR,APR,MAJ,JUN,JUL,AVG,SEP,OKT,NOV,DEC} mesec=JAN;
    */

    float temperature[BROJ_MESECI];
    float srednja_temp=0;
    while (1)
    {
        printf("Temperatura za mesec %2d: ",mesec);
        scanf("%f",&temperature[mesec - 1]); /* niz u C-u uvek ide od 0 */
        srednja_temp+=temperature[mesec - 1];
        if (mesec==DEC) break;
        mesec++;
    }

    srednja_temp/=BROJ_MESECI;
    printf("Srednja temperatura je %.2f\n",srednja_temp);
}
/*
    Jasno je da nam niz nije potreban za racunanje srednje vrednosti
    dovoljno je da mesecnu temperaturu ucitavamo u neku pomocnu - float
    promenljivu i da nju dodajemo na srednja_temp
*/
```

Prikaz nekih vrednosti adresa i niza temperature:

```
&temperature,p: 22FC(SS):0FCC
&temperature[0],p: 22FC(SS):0FCC - adresa clana 0
&temperature[1],p: 22FC(SS):0FD0 - adresa clana 1
```

```
temperature: { 3.0,8.0,12.0,15.0,20.0,26.0,29.0,30.0,26.0,20.0,13.0,7.0 }
- prikaz celog niza
```

```
temperature[0]: 3.0 - vrednost clana sa indeksom 0
```

Ovakav prikaz se može dobiti pomoću Watch prozora razvojnog okruženja Turbo C. Najpre se navodi izraz čija se vrednost posmatra, a onda način kako ona treba da bude ispisana. Uz pomoć ovog dodatka, može se isti izraz posmatrati na više načina.

```
izraz,d - kao broj
izraz,c - kao karakter
izraz,p - kao pokazivac (adresa)
```

8.4 Sastaviti program na programskom jeziku C koji učitava decimalan pozitivan celi broj u obliku niza znakova i ispisuje njegovu vrednost u binarnom obliku. Pretpostaviti da se za interno predstavljanje celih brojeva koristi 16 bitova.

```
#include <stdio.h>
main()
{
    char dec[10];
    short int bin,i;
    printf("Unesite decimalan broj: ");
    scanf("%s",dec); /* učitava string sa standardnog ulaza (dec=&dec[0]) */
    /* atoi vraća 0 ukoliko nije uspela konverzija
    Ukoliko je strlen(dec)=0 onda se drugi deo USLOVA (iza && operatora)
    neće ni proveravati. Po postavci zadatka očekuje se pozitivan broj,
    i takav uslov se jednostavno, ovim putem, može proveriti.
    */
    if (strlen(dec) && (bin=atoi(dec)))
    {
        printf("Binarni broj je: ");
        i=-1;
        while (++i<16)
        {
            putchar((bin & 0x8000) ? '1' : '0');
            /* 0x8000 ima jedinicu na najvišem 15-om bitu
            Gornja naredba će ispisati 15-i bit broja bin!
            Najpre ispisujemo najviše bitove, jer takav prikaz želimo na ekranu
            bit15 bit14 ... bit2 bit1 bit0 */
            bin <<= 1 ; /* bin = bin shl 1 ; pomeramo bin ulevo za 1 bit */
            if (i%4 == 3)
                putchar(' '); /* prikaz razmaka između svake polovine bajta */
        }
        printf("\n");
    }
    else
        printf("Neispravan broj ili nula\n");
}
```

Komentar:

16-bitni binarni broj može da ima najveću vrednost 65535, ukoliko se tretira kao **unsigned**, odnosno 32767, ukoliko se tretira kao **signed**. To konkretno znači da je za ovaj 16-bitni prikaz dovoljno da koristimo niz brojeva **char dec[6]** ( 5 za cifre i 1 za NULL terminator ). Sa druge strane, ako u ovom zadatku unesemo neki broj veći od 65535/32767 rezultat neće biti ispravan ( biće prikazano samo najnižih 16-bitova zadatog broja ako je **short int** na datom računaru duži od 16 bita). Funkcija **atoi** se može koristiti i za konverziju negativnih vrednosti.

Treba imati na umu i sledeću činjenicu: ako unesemo više od 10 znakova nepredvidiv je ishod izvršavanja ovog programa, kao i ostalih programa operativnog sistema koji su trenutno u memoriji! Jer, niz 'dec' ima dodeljeno svega 10 bajtova u memoriji i ništa preko toga! Ostatak memorije pripada drugim podacima ili drugom programskom kodu, tako da se upisivanjem van granica niza može izazvati dosta problema.

Viša tehnička škola  
Niš

## DEVETA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

9.1 Sastaviti program na programskom jeziku C koji štampa sadržaj kvadratne matrice  $n * n$  ( broj  $n$  se unosi sa tastature ) vrstu po vrstu, a potom sporednu dijagonalu, sleva-udesno. Matrica  $n * n$  formirati generatorom slučajnih brojeva RAND()

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int **a, n, i, j;
    printf("N="); scanf("%d", &n);
    a = calloc(n, sizeof(int*)); /*alocira memoriju za n pokazivaca na vrste*/
    for (i=0; i<n; i++)
    {
        *(a+i) = calloc(n, sizeof(int));
        /*alocira memoriju za n elemenata vrste koji su tipa int*/
        for (j=0; j<n; j++) *(*(a+i)+j) = rand()/((double)RAND_MAX + 1) * 10;
    }
    for (i=0; i<n; printf("\n"), i++)
        for (j=0; j<n; j++)
            printf(" %d", *(*(a+i)+j));
        printf("\n");
    for (i=n-1; i>=0; i--) printf(" %d", *(*(a+i)+n-1-i));
    putchar('\n');
}
```

Prepisati rezultate rada programa:



9.2 Sastaviti program na jeziku C za učitavanje imena gradova uz njihovo uređivanje po abecednom redosledu i ispisivanje rezultata. U svakom redu se učitava po jedno ime sve dok se ne učitava prazan red.

```
/* program za sortiranje imena gradova */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_GRAD 100
#define MAX_DUZ 30

main()
{
    char *adrese[MAX_GRAD], *adresa;
    int znak, br_grad = 0, duz, i;

    do
    {
        adresa = calloc(MAX_DUZ, sizeof(char));
        for (duz = 0; duz < MAX_DUZ - 1; duz++)
            if ((znak = getchar()) != '\n')
                *(adresa + duz) = znak;
            else
                break;
        *(adresa + duz) = '\0';
        if (!duz)
        {
            free(adresa); break;
        }
        adresa = realloc(adresa, duz + 1);
        for (i = br_grad - 1; i >= 0; i--)
            if (strcmp(adrese[i], adresa) > 0)
                adrese[i + 1] = adrese[i];
            else
                break;
        adrese[i + 1] = adresa;
    } while (br_grad++ < MAX_GRAD);

    for (i = 0; i < br_grad; i++)
        printf("%s\n", adrese[i]);
}
```

Upisati rezultate rada programa:

9.3 Napisati program na programskom jeziku C koji ponavlja sledeću sekvencu operacija:

1. učitava informaciju o tipu podataka sa kojim će raditi (ceo broj ili znak);
2. učitava dužinu niza podataka;
3. učitava niz podataka zadatog tipa i dužine u dinamički alociranu memoriju;
4. ispisuje u heksadecimalnom obliku sadržaj učitanih podataka, bajt po bajt.

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    void *blok_p, *bajt_p;
    unsigned int vel, n;
    char o, *format_p;

    while(1)
    {
        o = 'n';
        while (o != 'c' && o != 'z' && o != 'k')
        {
            printf("Tip brojeva: (c)eli/(z)naci, ili (k)raj?");
            scanf("\n%c", &o);
            switch(o)
            {
                case 'c': vel = sizeof(int);
                    format_p = "%d";
                    break;
                case 'z': vel = sizeof(char);
                    format_p = "\n%c";
                    break;
                case 'k': break;
                default: printf("Neispravan unos! Ponovite.\n");
            }
        }
        if (o == 'k') break;
        printf("Broj podataka?");
        scanf("%d", &n);
        blok_p = malloc(vel*n);
        printf("Podaci:\n");
        for (bajt_p=blok_p; bajt_p<(char*)blok_p+n*vel; (char*)bajt_p+=vel)
        /* C75.c(34) : warning C4133: '<': incompatible types - from 'char *' to 'void *' */
        { printf(" "); scanf(format_p, bajt_p); }
        printf("Bajtovi:\n");
        for (bajt_p = blok_p; bajt_p<(char*)blok_p+n*vel; ((char*)bajt_p)++)
            printf("%x%c", *(char *)bajt_p,
                (((char*)bajt_p-(char*)blok_p)%vel == vel-1)? '\n': ' ');
        printf("\n");
        free(blok_p);
    }
}
```

9.4 Napisati program na programskom jeziku C koji učitava dva znakovna niza, čije se dužine unose kao podatak sa tastature, izvrši nadovezivanje drugog na prvi, okrene naopako dobijeni niz i ispiše ga na standardnom izlaznom uređaju.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

main()
{
    int n;
    char c, *d, *p, *prvi, *drugi;
    printf("Maksimalna dužina za string: ");
    scanf("%d", &n);
    while(getchar() != '\n') /* preskacemo sve do kraja linije gde je bio zadat broj */
        ;
    p = calloc(2*n+1, sizeof(char)); /* alociramo memoriju */
    d = calloc(n+1, sizeof(char));
    if ((d == NULL) || (p == NULL))
    {
        printf("\nNije moguca alokacija!\n");
        exit(1); /* losije resenje je sa: goto kraj; */
    }
    prvi = p; drugi = d;
    /* učitavanje prvog, pa drugog stringa znak po znak */
    while ((*p = getchar()) != '\n')
        p++;
    *p = '\0';
    while ((*d = getchar()) != '\n')
        d++;
    *d = '\0';
    p = prvi; d = drugi;
    /* konkatencija */
    while (*p)
        p++;
    while (*p++=*d++)
        ;
    /* okretanje */
    p = prvi;
    for (d=p+(strlen(p)-1); p < d; p++, d--)
        c=*p, *p=*d, *d=c;
    /* ispis */
    printf("\n%s\n", prvi);
    /* koristili bi labelu kraj;; da smo radili sa goto */
}
```

Napisati rezultate rada programa:

ulazni nizovi:

obrnuti izlazni niz



Viša tehnička škola  
Niš

## DESETA LABORATORIJSKA VEŽBA IZ PROGRAMSKIH JEZIKA I

### STRUKTURE

Strukture predstavljaju kompleksne tipove podataka koje mogu sadržati promenljive istog ili različitog tipa. One čine pogodno sredstvo za rad sa podacima koji su u međusobnoj vezi, jer se mogu grupisati pod istim imenom. Znači, ako je potrebno napraviti strukturu pod nazivom *knjige* i sa promenljivim *naslov*, *autor* i *cena*, slog bi izgledao ovako:

```
struct knjige {
char naslov[MAXTIT]; /*Deklaracija šablona strukture se obavezno završava znakom ;
char autor[MAXAUT];
float cena;
};
```

Ovaj format opisuje strukturu načinjenu od dva niza tipa char i jedne promenljive tipa float. To su strukturne promenljive. Razmotrićemo program koji ima za zadatak da formira strukture knjige i biblioteke i da posle unosa određenih podataka izlista sve unete promene. Deklaracijom *struc knjiga biblioteka* kreira se biblioteka kao strukturna promenljiva tipa knjiga.

Program 10.1:

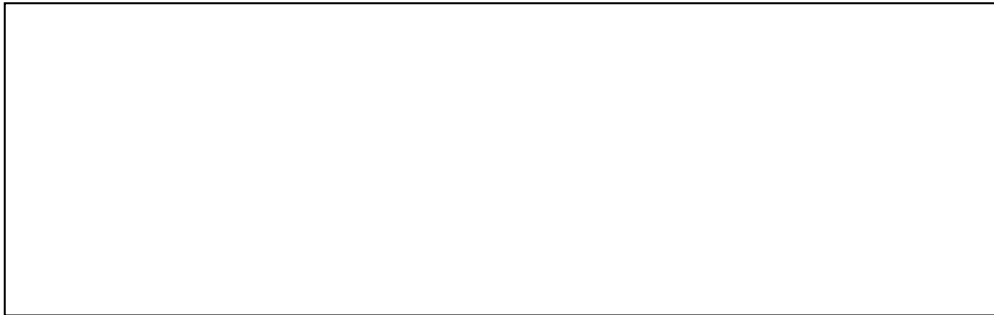
```
#include <stdio.h>
#define MAXTIT 40 /*naslov knjige max. 40 karaktera*/
#define MAXAUT 40 /*ime autora max. 40 karaktera*/
#define MAXBKS 4 /*broj knjiga u biblioteci max. 4*/
#define STOP "" /*nula string, završava unos*/
struct knjiga {
char naslov[MAXTIT];
char autor[MAXAUT];
float cena;
};

void main()
{
struct knjiga biblioteka[MAXBKS]; /*niz struktura knjiga */
int brojac=0;
int index;
printf("Unesite naslov knjige.\n");
printf("Pritisnite ENTER na pocetku linije za prekid unosa.\n");
while(brojac<MAXBKS && strcmp(gets(biblioteka[brojac].naslov),STOP) !=0)
{
printf("Sada unesite ime autora.\n");
gets(biblioteka[brojac].autor);
printf("Sada unesite cenu.\n");
```

```
scanf("%f",&biblioteka[brojac++].cena);
while(getchar()!='\n');
if(brojac < MAXBKS)
printf("Unesi sledeci naslov.\n");
}
printf("Ovo je lista vasih knjiga.\n");
for(index=0; index < brojac; index++)
printf("%s je napisao %s: %2f dinara\n",biblioteka[index].naslov,
biblioteka[index].autor, biblioteka[index].cena);
}
```

Napomena: Unećemo četiri bloka podataka: naziv knjige ime autora i cena jer je po definiciji ( MAXBKS = 4, konstanta ), a kao izlaz dobijamo listu unetih promena.

Prepisati rezultate rada programa:



*Program 10.2* može biti korišćen za odbrojavanje vremena u računaru. Računari koriste osnovni klok ( clock ) koji može biti osnova za odbrojavanje vremena ( sat, minut, sekund ). Program za odbrojavanje vremena startuje se klockom, izvršavajući se svakog sekunda, pri čemu se koriguju memorijske lokacije koje sadrže vrednosti za sate, minute i sekunde. Koristi se sledeći algoritam:

- ◆ Ako je broj sekunda jednak 60, broj minute se inkrementira za 1, a broj sekunda postavlja na 0.
- ◆ Ako je broj minuta jednak 60, broj sati se inkrementira za 1, a broj minuta postavlja na 0.
- ◆ Ako je broj sati jednak 23, broj sati, minuta i sekunda se postavljaju na 0.

Program učitava vreme, koriguje vrednosti za sate, minute i sekunde i štampa korigovano vreme. Funkcija scanf koristi konverzioni niz %d:%d:%d: koji specificira da treba učitati tri celobrojne vrednosti razdvojene tačkom. Učitane vrednosti se memorišu u strukturnoj promenljivoj tekuce\_vreme. Nakon učitavanja tekućeg vremena program koriguje vrednosti za sate, minute i sekunde u struktuiranoj promenljivoj naredno\_vreme. Funkcija štampa vrednost korigovanog vremena. Za štampu se koristi konverzacioni niz %02d:%02d:%02d: koji sugerije da se štampaju tri celobrojne vrednosti u polju širine dva karaktera, pri čemu se prazna mesta popunjavaju nulama.

```
/*Program 10.2 za korekciju tekuceg vremena*/
main()
{
struct vreme{
    int sat;
    int minut;
    int sekund;
} tekuce_vreme, naredno_vreme;

printf("Unesite tekuce vreme ?[cc:mm:ss] :");
scanf("%d:%d:%d",&tekuce_vreme.sat,&tekuce_vreme.minut,&tekuce_vreme.sekund);

    naredno_vreme=tekuce_vreme;
if(++naredno_vreme.sekund==60)
{
naredno_vreme.sekund=0;
    if(++naredno_vreme.minut==60)
    {
        naredno_vreme.minut=0;
if(++naredno_vreme.sat==24)
naredno_vreme.sat=0;
}
}
printf("Naredno vreme je
%02d:%02d:%02d\n",naredno_vreme.sat,naredno_vreme.minut,naredno_vreme.sekund);
}
```

Prepisati rezultate rada programa:



*Program 10.3* može biti korišćen za sagledavanje pristupa članova strukture uz pomoć pokazivača na strukturu. Strukture u sprezi sa pokazivačima čine veoma moćno sredstvo za rešavanje problema iz različitih domena strukturnog razvoja softvera. U programskom jeziku C je moguće pored same strukture proslediti i adresu na kojoj se ta struktura nalazi, što predstavlja veliku pogodnost. Za korišćenje pokazivača na strukture postoji nekoliko razloga:

- ◆ sa pokazivačima na strukture se mnogo lakše manipuliše nego sa samim strukturama
- ◆ sve verzije programskog prevodioca podržavaju prosleđivanje pokazivača na strukturu kao argumenta funkcije
- ◆ ogroman broj reprezentacija podataka predstavlja strukture koje sadrže druge strukture, što je idealna mogućnost za korišćenje pokazivača

```

#include <stdio.h>
#define LEN 20
struct imena_st{
    char ime[LEN];
    char prezime[LEN];
};
struct prijat_st{
    struct imena_st imenik;
    char posao[LEN];
    float prihod;
};

void main()
{
    static struct prijat_st drug[2] = {
        {{ "Borivoje", "Milosevic" },
        "profesor PJ I",
        15000.00},
        {{ "Zoran", "Milivojevic" },
        "profesor DOS",
        16000.00}
    };
    struct prijat_st *prij;    /*pokazivac u strukturi gde prijavuje na bilo koju strukturu
                               tipa prijat_st*/
    printf("adresa #1: %u #2: %u\n", &drug[0], &drug[1]);
    prijav=&drug[0]; /* govori pokazivacu na koju adresu da ukazuje*/
    printf("pokazivac #1: %u #2: %u\n", prijav, prijav+1);
    printf("prij->prihod je s%.2f: (*prij).prihod je s%.2f\n",prij->prihod,(*prij).prihod);
    prijav++;
    printf("prij->posao je %s: prijav->imenik.prezime je %s\n",prij->posao,prij->imenik.prezime);
}

```

# ZBIRKA REŠENIH ZADATAKA

# PRIMERI PROGRAMA ZA PRIPREMU PRVOG KOLOKVIJUMA

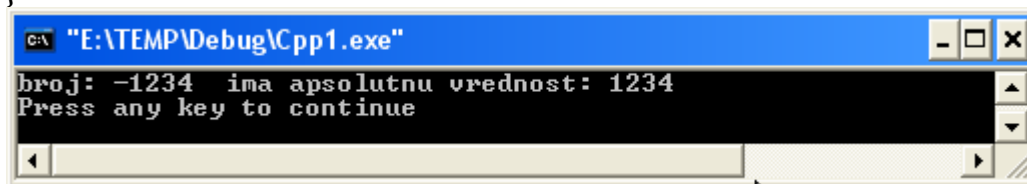
## 1. Sastaviti program na C jeziku za izračunavanje:

*/\* abs funkcija za PRIMER APSOLUTNE VREDNOSTI CELOG BROJA \*/*

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

int main(void)
{
    int number = -1234;

    printf("broj: %d ima apsolutnu vrednost: %d\n", number, abs(number));
    return 0;
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
broj: -1234 ima apsolutnu vrednost: 1234
Press any key to continue
```

## 2. Sastaviti program na C jeziku za izračunavanje:

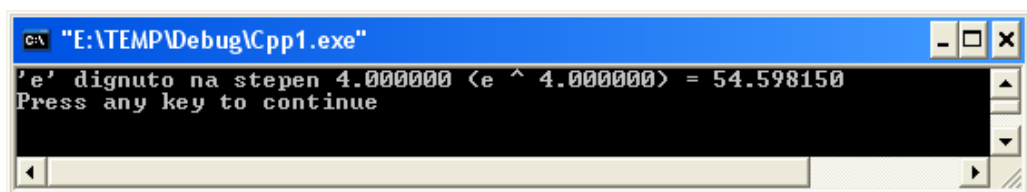
*/\* exp funkcija za IZRAČUNAVANJE BROJA e NA REALNI STEPEN \*/*

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

int main(void)
{
    double result;
    double x = 4.0;

    result = exp(x);
    printf("'e' dignuto na stepen %lf (e ^ %lf) = %lf\n",
           x, x, result);

    return 0;
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
'e' dignuto na stepen 4.000000 (e ^ 4.000000) = 54.598150
Press any key to continue
```

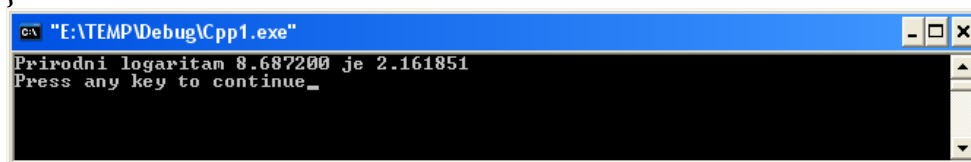
**3. Sastaviti program na C jeziku za izračunavanje:****/\* log funkcija za PRIMER LOGARITMA REALNOG BROJA \*/**

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

int main(void)
{
    double result;
    double x = 8.6872;

    result = log(x);
    printf("Prirodni logaritam %lf je %lf\n", x, result);

    return 0;
}
```



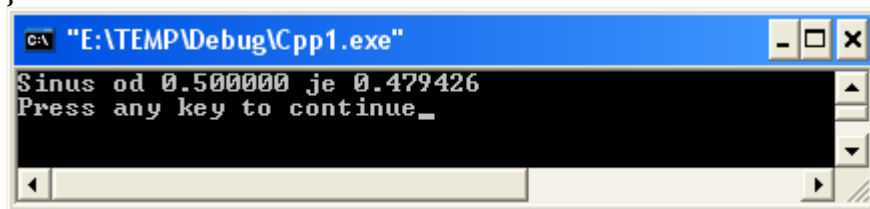
```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Prirodni logaritam 8.687200 je 2.161851
Press any key to continue_
```

**4. Sastaviti program na C jeziku za izračunavanje:****/\* sin funkcija za PRIMER IZRAČUNAVANJA SINUSA REALNOG BROJA \*/**

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

int main(void)
{
    double result, x = 0.5;

    result = sin(x);
    printf("Sinus od %lf je %lf\n", x, result);
    return 0;
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Sinus od 0.500000 je 0.479426
Press any key to continue_
```

**5. Sastaviti program na C jeziku za izračunavanje:****/\* poly funkcija za PRIMER IZRAČUNAVANJA VREDNOSTI POLINOMA \*/**

```
#include <stdio.h>
#include <math.h>
/* program za resavanje vrednosyi polinoma */
/* polynomial: x**3 - 2x**2 + 5x - 1 */
```

```
int main(void)
{
    double array[] = {-1.0, 5.0, -2.0, 1.0};
    double result;

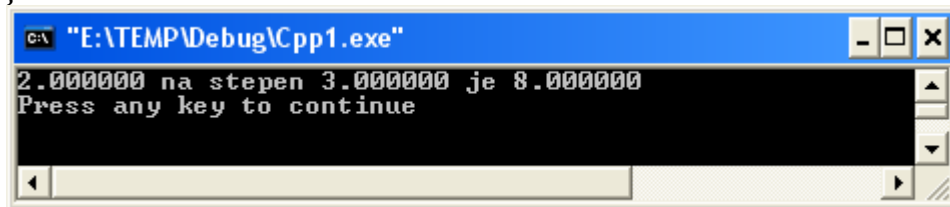
    result = poly(2.0, 3, array);
    printf("Polinom: x**3 - 2.0x**2 + 5x - 1 na 2.0 je %lf\n", result);
    return 0;
}
```

#### 6. Sastaviti program na C jeziku za izračunavanje:

**/\* pow funkcija za PRIMER IZRAČUNAVANJA REALNOG BROJA NA STEPEN \*/**

```
#include <math.h>
#include <stdio.h>
int main(void)
{
    double x = 2.0, y = 3.0;

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



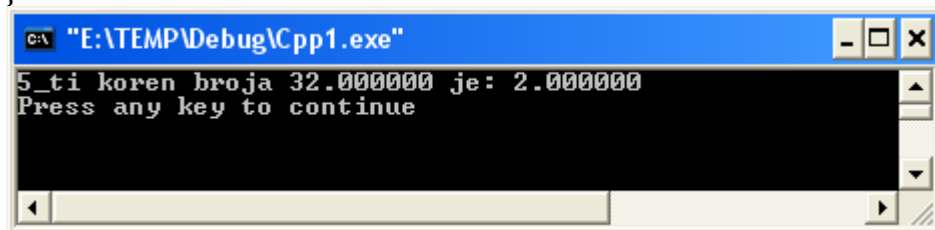
```
C:\ "E:\TEMP\Debug\Cpp1.exe"
2.000000 na stepen 3.000000 je 8.000000
Press any key to continue
```

#### 7. Sastaviti program na C jeziku za izračunavanje:

**/\* pow funkcija za PRIMER IZRAČUNAVANJA n-tog KORENA REALNOG BROJA \*/**

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

float main(void)
{int n=5;
double x = 32.0, rezultat;
rezultat=pow(x,1./n);
printf("%d_t koren broja %lf je: %lf\n",n, x,rezultat);
return 0;
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
5_t koren broja 32.000000 je: 2.000000
Press any key to continue
```

NAPOMENA: u opštem slučaju  $rezultat = \sqrt[n]{x^m} = pow(x, m./n)$

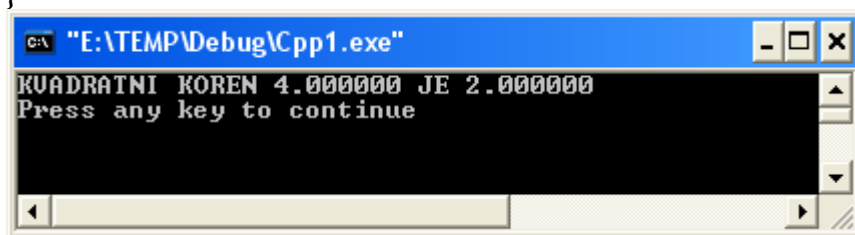


**8. Sastaviti program na C jeziku za izračunavanje:**

```
/* sqrt funkcija kao PRIMER IZRACUNAVANJA KVADRATNOG KORENA REALNOG BROJA */
```

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

int main(void)
{
    double x = 4.0, rezultat;
    rezultat = sqrt(x);
    printf("KVADRATNI KOREN %lf JE %lf\n", x, rezultat);
    return 0;
}
```



The screenshot shows a Windows command prompt window titled "E:\TEMP\Debug\Cpp1.exe". The output of the program is displayed as follows:

```
KVADRATNI KOREN 4.000000 JE 2.000000
Press any key to continue
```

**9. Sastaviti program na C jeziku za izračunavanje:**

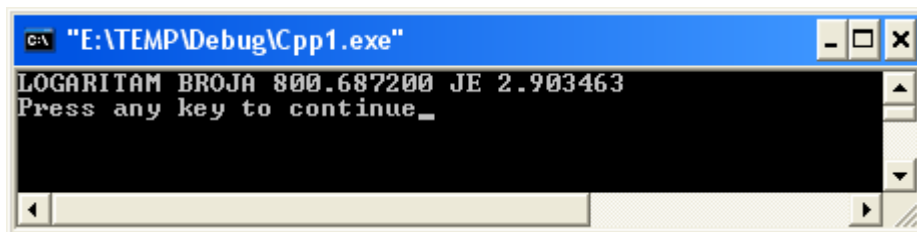
```
/* log10 funkcija za PRIMER LOGARITMA ZA OSNOVU 10 REALNOG BROJA */
```

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

int main(void)
{
    double result;
    double x = 800.6872;

    result = log10(x);
    printf("LOGARITAM BROJA %lf JE %lf\n", x, result);

    return 0;
}
```



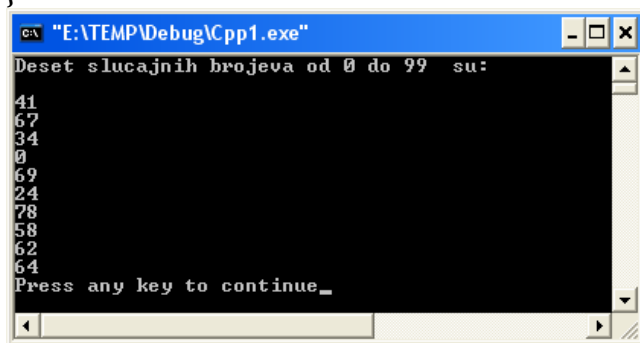
The screenshot shows a Windows command prompt window titled "E:\TEMP\Debug\Cpp1.exe". The output of the program is displayed as follows:

```
LOGARITAM BROJA 800.687200 JE 2.903463
Press any key to continue_
```

**10. Sastaviti program na C jeziku za izračunavanje:****/\* rand funkcija za PRIMER GENERISANJA SLUCAJNIH BROJEVA \*/**

```
#include <stdlib.h>
#include <stdio.h>

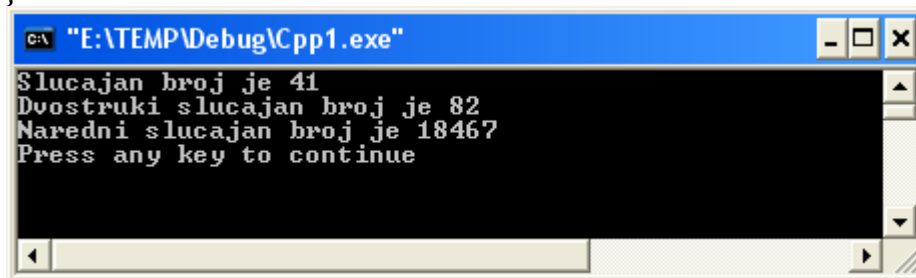
int main(void)
{
    int i;
    printf("Deset slucajnih brojeva od 0 do 99 su:\n\n");
    for(i=0; i<10; i++)
        printf("%d\n", rand() % 100);
    return 0;
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Deset slucajnih brojeva od 0 do 99 su:
41
67
34
0
69
24
78
58
62
64
Press any key to continue_
```

**11. Sastaviti program na C jeziku za izračunavanje:****Generisanje slučajnog broja, izračunati njegovu dvostruku vrednost i prikazati sledeći slučajni broj.**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void main()
{
    int slucajan, dvostruki;
    slucajan = rand();
    dvostruki = slucajan*2;
    printf("Slucajan broj je %d\n", slucajan);
    printf("Dvostruki slucajan broj je %d\n", dvostruki);
    printf("Naredni slucajan broj je %d\n", rand());
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Slucajan broj je 41
Dvostruki slucajan broj je 82
Naredni slucajan broj je 18467
Press any key to continue
```

**12. Sastaviti program na C jeziku za izračunavanje:**

**/\* Rad sa specijalnim karakterima \*/**

```
#include <stdio.h>
int main(void)
{printf("Alert: zvučni ili vizuelni signal \a\n");
printf("Alert: NESTO SE CUJE \n");
printf("Alert: PONOVO zvučni ili vizuelni signal \a\n");
printf("Po\bvratnik ili b\backspace: sakrij me\b \n");
printf("Form feed. \f\n");
printf("Horizontalni\ttab\n");
return 0;}
```

**13. Sastaviti program na C jeziku za izračunavanje:**

**/\*Stampanje vrednosti stepena Farenhajta i Celzijusa pocev od 0 do 300 sa korakom 20 \*/**

```
#include <stdio.h>
int main(void)
{
float fahr, celsius; /*vrednosti stepena Celzijusa, Farenhajta*/
int lower, upper, step; /* donja vrednost skale - lower, gornja vrednost skale -upper, korak
skaliranja */
/*inicijalizacije*/
lower = 0;
upper = 300;
step = 20;
/*stampanje zaglavlja "tabele" */
printf("F C\n\n");
/*stampanje vrednosti stepena pocev
od 0 do 300 sa korakom 20 */
fahr = lower;
while(fahr <= upper)
{
celsius = (5.0 / 9.0) * (fahr - 32.0);
printf("%3.0f %6.1f\n", fahr, celsius);
fahr = fahr + step;
}
return 0;
}
```

F	C
0	-17.8
20	-6.7
40	4.4
60	15.6
80	26.7
100	37.8
120	48.9
140	60.0
160	71.1
180	82.2
200	93.3
220	104.4
240	115.6
260	126.7
280	137.8
300	148.9

**drugi način**

obrnuto

```
#include <stdio.h>

/* print Fahrenheit-Celsius table */
int
main()
{
    int fahr;

    for (fahr = 300; fahr >= 0; fahr = fahr - 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));

    return 0;
}
```

```
C:\Brisi\Debug\Cpp1.exe
300 148.9
280 137.8
260 126.7
240 115.6
220 104.4
200 93.3
180 82.2
160 71.1
140 60.0
120 48.9
100 37.8
 80 26.7
 60 15.6
 40  4.4
 20 -6.7
  0 -17.8
Press any key to continue
```

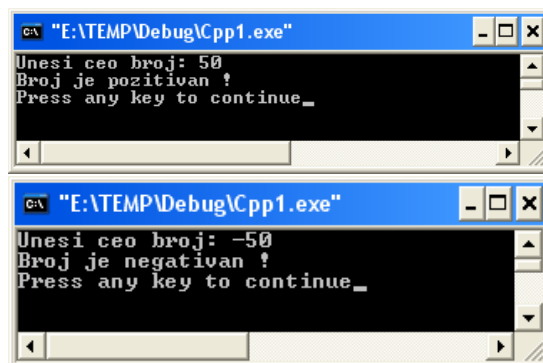
**14. Sastaviti program na C jeziku za izračunavanje:****Za štampanje rezultata množenja (od 1\*1 do 9\*9 u redovima po 9).**

```
#include<stdio.h>
void main()
{
    int i,j;
    for(i=1;i<10;i++)
    {
        for(j=1;j<10;j++)
            printf("%3d",i*j);
        printf("\n");
    }
}
```

```
E:\TEMP\Debug\Cpp1.exe
 1  2  3  4  5  6  7  8  9
 2  4  6  8 10 12 14 16 18
 3  6  9 12 15 18 21 24 27
 4  8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
Press any key to continue
```

### 15. Sastaviti program na C jeziku za izračunavanje: Određivanje znaka unetog broja

```
#include<stdio.h>
void main()
{
int n;
printf("Unesi ceo broj: ");
scanf("%d",&n);
if(n>=0)
printf("Broj je pozitivan !\n");
if(n<0)
printf("Broj je negativan !\n");
}
```



Drugi način sa if else strukturom :

```
#include<stdio.h>
void main()
{
int n;
printf("Unesi ceo broj: ");
scanf("%d",&n);
if(n>=0)
printf("Broj je pozitivan !\n");
else
printf("Broj je negativan !\n");
}
```

### 16. Sastaviti program na C jeziku za izračunavanje: Rešavanje kvadratne jednačine $ax^2+bx+c=0$ .

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void main()
{
float delta,a,b,c,x1,x2;
printf("Unesite a : ");
scanf("%f",&a);
printf("Unesite b : ");
scanf("%f",&b);
printf("Unesite c : ");
scanf("%f",&c);
delta=b*b-(4*a*c);
printf("Podkorena velicina delta je:%f",delta);
if(delta<0)
{
printf("Jednacina nema resenja !\n");
exit(0);
}
if(delta==0)
```

```

{
x1=-b/(2*a);
printf("E Jednacina ima dva ista resenja !\n");
printf("x1=x2=%f",x1);
exit (0);
}

```

```

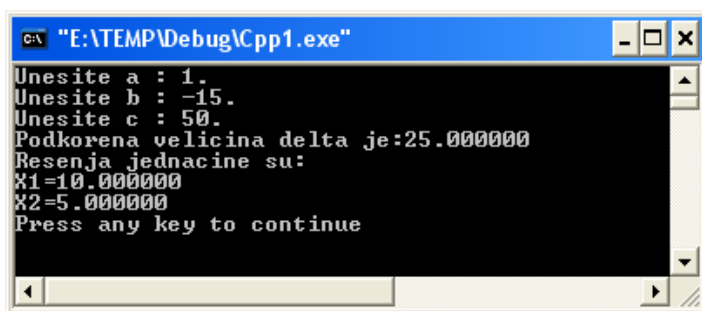
x1=(-b+sqrt(delta))/(2*a);
x2=(-b-sqrt(delta))/(2*a);

```

```

printf("\nResenja jednacine su:");
printf("\nX1=%f",x1);
printf("\nX2=%f\n",x2);
}

```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite a : 1.
Unesite b : -15.
Unesite c : 50.
Podkorena velicina delta je:25.000000
Resenja jednacine su:
X1=10.000000
X2=5.000000
Press any key to continue

```

### 17. Sastaviti program na C jeziku za izračunavanje: programa sa menijem:

- 1- Program matematika
- 2- Program finansija
- 3- Program zabave
- 4- Exit

upotrebom while naredbe i if, else if strukture za startovanje svih stavki iz menija.

```

#include<stdio.h>
#include<stdlib.h>
void main()
{
int choice;

while(1)
{
printf("\n\nMeni:\n");
printf("1- Program matematika\n2- Program finansija\n");
printf("3- Program zabave\n4- Exit");
printf("\n\nVas izbor -> ");
scanf("%d",&choice);

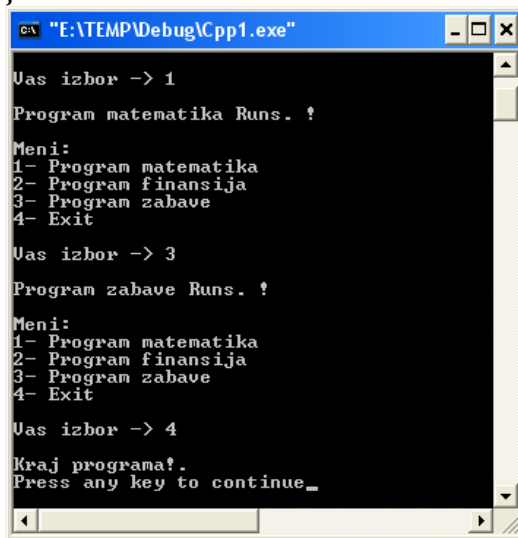
if(choice==1)
printf("\nProgram matematika Runs. !");

```

```

else if(choice==2)
    printf("\nProgram finansija Runs. !");
else if(choice==3)
    printf("\nProgram zabave Runs. !");
else if(choice==4)
    {
    printf("\nKraj programa!\n");
    exit(0);
    }
else
    printf("\nPogresan izbor");
}
}

```



```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Uas izbor -> 1
Program matematika Runs. !
Meni:
1- Program matematika
2- Program finansija
3- Program zabave
4- Exit
Uas izbor -> 3
Program zabave Runs. !
Meni:
1- Program matematika
2- Program finansija
3- Program zabave
4- Exit
Uas izbor -> 4
Kraj programa!.
Press any key to continue_

```

Isti zadatak samo upotrebom naredbe switch i case.

```

#include<stdio.h>
#include<stdlib.h>
void main()
{
int choice;

while(1)
{
printf("\n\nMeni:\n");
printf("1- Program matematika\n2- Program finansija\n");
printf("3- Program zabave\n4- Exit");
printf("\n\nUas izbor -> ");
scanf("%d",&choice);

switch(choice)
{ case 1 :
    printf("\nProgram matematika Runs. !");
break;
case 2 :

```

```

    printf("\nProgram finansija Runs. !");
break;
case 3 :
    printf("\nProgram zabave Runs. !");
break;
case 4 :
    {
        printf("\nKraj programa!.n");
        exit(0);
    }
default:
printf("\nPogresan izbor");
}
}
}

```

### 18. Sastaviti program na C jeziku za izračunavanje:

funkcije  $y = \frac{x^2 \cdot \sin(x) - 1}{\sqrt[3]{x}}$ .

```

#include <stdio.h>
#include <math.h>
void main()
{
double x,y,pom;
printf("Unesite broj\n");
scanf("%lf",&x);
pom=pow(x,1./3.);
y=(x*x*sin(x)-1)/pom;
printf("Vrednost izraza je %lf a vrednost korena %lf\n",y,pom);
}

```

### 19. Sastaviti program na C jeziku za izračunavanje:

//I primenu pperatora inkrementa

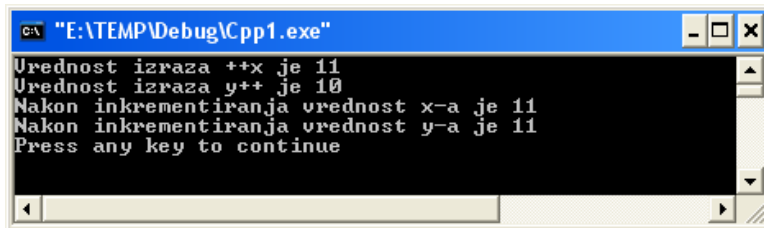
```

#include <stdio.h>
void main()
{
int x, y;
x = 10;
y =10;
printf("Vrednost izraza ++x je %d\n", ++x);

```



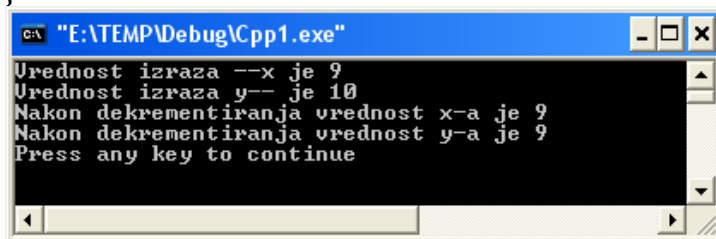
```
printf("Vrednost izraza y++ je %d\n", y++);
printf("Nakon inkrementiranja vrednost x-a je %d\n", x);
printf("Nakon inkrementiranja vrednost y-a je %d\n", y);
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Vrednost izraza ++x je 11
Vrednost izraza y++ je 10
Nakon inkrementiranja vrednost x-a je 11
Nakon inkrementiranja vrednost y-a je 11
Press any key to continue
```

## 20. Sastaviti program na C jeziku za izračunavanje: //operatora dekrementa

```
#include <stdio.h>
void main()
{
int x, y;
x = 10;
y = 10;
printf("Vrednost izraza --x je %d\n", --x);
printf("Vrednost izraza y-- je %d\n", y--);
printf("Nakon dekrementiranja vrednost x-a je %d\n", x);
printf("Nakon dekrementiranja vrednost y-a je %d\n", y);
}
```

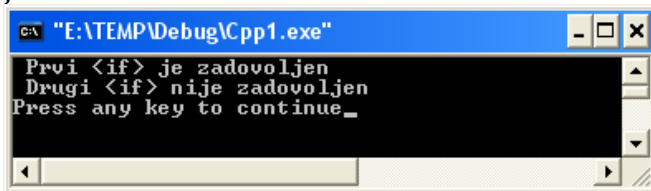


```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Vrednost izraza --x je 9
Vrednost izraza y-- je 10
Nakon dekrementiranja vrednost x-a je 9
Nakon dekrementiranja vrednost y-a je 9
Press any key to continue
```

## 21. Sastaviti program na C jeziku za izračunavanje: //operatora dodeljivanja

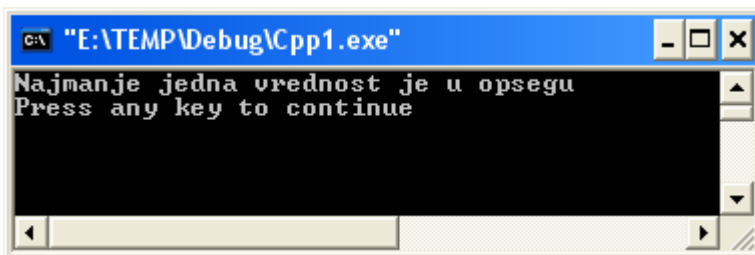
```
#include <stdio.h>
void main()
{
int x, y;
x = 6;
y = 6;
if ( x = 7)
{
printf(" Prvi <if> je zadovoljen\n" );
}
```

```
}  
else  
{  
printf( " Prvi <if> nije zadovoljen\n" );  
}  
if ( y == 7 )  
{  
printf( " Drugi <if> je zadovoljen\n" );  
}  
else  
printf( " Drugi <if> nije zadovoljen\n" );  
}
```



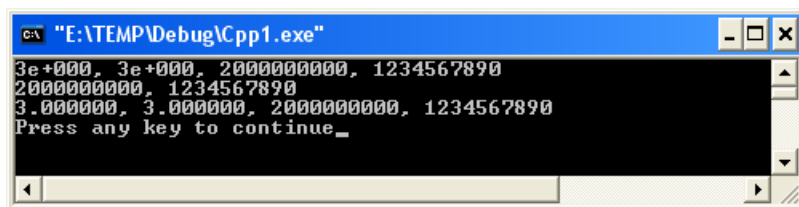
## 22. Sastaviti program na C jeziku za izračunavanje: //relacionih izraza i logickih operatora

```
#include <stdio.h>  
void main()  
{  
int minimum,maximum;  
int pr1, pr2;  
minimum=10;  
maximum = 1000;  
pr1 = 110;  
pr2 = 1123;  
if( (pr1 < maximum && pr1 > minimum) || (pr2 < maximum && pr2 > minimum) )  
printf( "Najmanje jedna vrednost je u opsegu\n" );  
if( (pr1 < maximum && pr1 > minimum) && (pr2 < maximum && pr2 > minimum) )  
{  
printf( "Obe vrednosti su u opsegu\n" ) ;  
}  
}
```



**23. Sastaviti program na C jeziku za izračunavanje:****//poziva funkcije stdio.h i primene kod izlaznog tipa promenljive.**

```
#include <stdio.h>
void main()
{
float n1=3.0;
double n2=3.0;
long n3=2000000000;
long n4=1234567890;
printf("%.1e, %.1e, %ld, %ld\n", n1,n2,n3,n4);
printf("%ld, %ld\n", n3, n4);
printf("%f, %f, %ld, %ld\n", n1,n2,n3,n4);
}
```

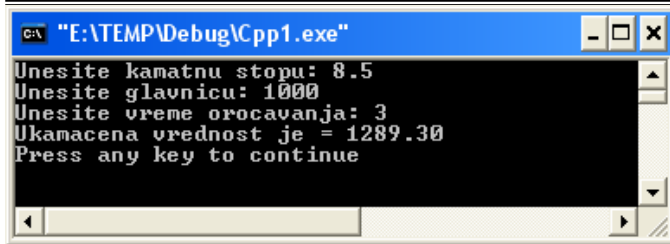


```
C:\ "E:\TEMP\Debug\Cpp1.exe"
3e+000, 3e+000, 2000000000, 1234567890
2000000000, 1234567890
3.000000, 3.000000, 2000000000, 1234567890
Press any key to continue_
```

**24. Sastaviti program na C jeziku za izračunavanje:****//Obracun kamata u pokretnom zrezu**

```
#include <stdio.h>
#include <math.h>
void main( )
{
double stopa, period, glavnica;
printf( "Unesite kamatnu stopu: " );
scanf( "%lf", &stopa ); /* ulaz u pokretnom zrezu */
/* konverzija u procenete */
stopa = stopa / 100.0;
/* kamata se izracunava mesечно */
stopa= stopa / 12.0;
printf( "Unesite glavnica: " );
scanf( "%lf", &glavnica );
printf( "Unesite vreme orocavanja: " );
scanf( "%lf", &period );

/*Stopa je konvertovana na mesecnu osnovicu*/
period = period * 12.0;
printf( "Ukamacena vrednost je = %.2f\n",
glavnica * pow( (1.0+stopa), period));
}
```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite kamatnu stopu: 8.5
Unesite glavnicu: 1000
Unesite vreme orocavanja: 3
Ukamacena vrednost je = 1289.30
Press any key to continue

```

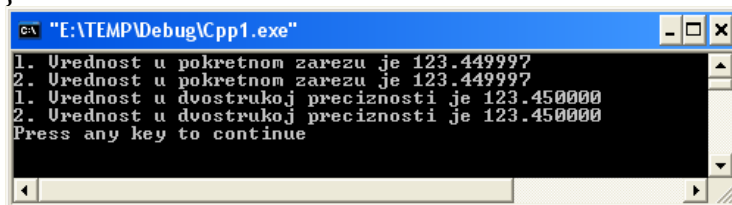
## 25. Sastaviti program na C jeziku za izračunavanje:

### //zaokruzivanja vrednosti

```

#include <stdio.h>
void main()
{
float fvr_pz;
double dvr_pz;
fvr_pz= 123.45;
printf("1. Vrednost u pokretnom zarezu je %f\n", fvr_pz);
fvr_pz /= 3.30;
fvr_pz *= 3.30;
printf("2. Vrednost u pokretnom zarezu je %f\n", fvr_pz);
dvr_pz = 123.45;
printf("1. Vrednost u dvostrukoj preciznosti je %f\n", dvr_pz);
dvr_pz /= 3.30;
dvr_pz *= 3.30;
printf("2. Vrednost u dvostrukoj preciznosti je %f\n", dvr_pz);
}

```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
1. Vrednost u pokretnom zarezu je 123.449997
2. Vrednost u pokretnom zarezu je 123.449997
1. Vrednost u dvostrukoj preciznosti je 123.450000
2. Vrednost u dvostrukoj preciznosti je 123.450000
Press any key to continue

```

## 26. Sastaviti program na C jeziku za izračunavanje:

### //Formatiranje ulaza u pokretnom zarezu

```

#include <stdio.h>
void main()
{
/* deklaracija podataka */
float f_pro;
double d_pro;
/*dodela vrednosti*/
f_pro=106.11;
d_pro=-0.0000654;
/*stampanje vrednosti promenljivih */
printf ("Promenljiva f_pro=%2f\n", f_pro);
}

```

```
printf ("Promenljiva d_pro=%.11f\n", d_pro);
printf ("Promenljiva f_pro=%e\n", f_pro);
printf ("Promenljiva d_pro=%G\n", d_pro);
}
```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Promenljiva f_pro=106.110001
Promenljiva d_pro=-0.00006540000
Promenljiva f_pro=1.061100e+002
Promenljiva d_pro=-6.54E-005
Press any key to continue_

```

## 27. Sastaviti program na C jeziku za izračunavanje:

//upravljacke struktura if

```
#include <stdio.h>
void main()
{
int x, y, z;
x = 10;
y= 3;
z = ( x / y ) * y;
if (x == z)
{
printf ( "Vrednosti x-a i z-a su jednake\n");
printf ( "Vrednost x-a je %d\n", x );
printf ( "Vrednost z-a je %d\n", z );
}
if ( x <= z)
{
printf ("Vrednost x-a je < ili = od vrednosti z-a\n");
printf( "Vrednost x-a je %d\n", x );
printf( "Vrednost z-a je %d\n", z );
}
if ( x >=z)
{
printf ( "Vrednost x-a je > ili = od vrednosti z-a\n" );
printf ( "Vrednost x-a je %d\n", x );
printf ( "Vrednost z-a je %d\n", z );
}
}
```

```

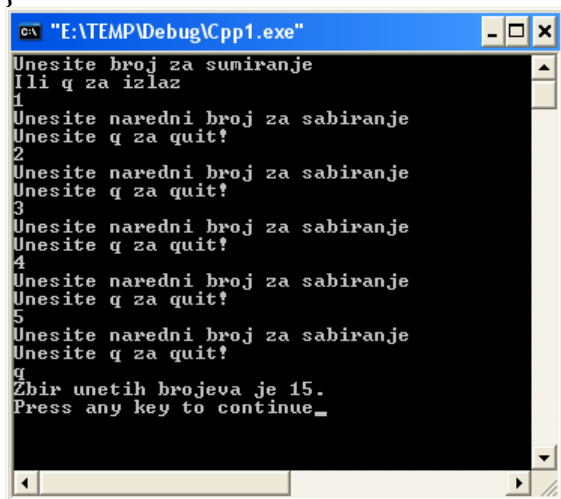
C:\ "E:\TEMP\Debug\Cpp1.exe"
Vrednost x-a je > ili = od vrednosti z-a
Vrednost x-a je 10
Vrednost z-a je 9
Press any key to continue_

```

**28. Sastaviti program na C jeziku za izračunavanje:  
//formiranje zbira celih brojeva while petljom**

```
#include <stdio.h>
void main()
{
long num, sum = 0L;
int status;
printf("Unesite broj za sumiranje\n");
printf("Ili q za izlaz\n");
status = scanf("%ld", &num);
while (status==1)
{
sum = sum+num;

printf("Unesite naredni broj za sabiranje\n");
printf("Unesite q za quit!\n");
status=scanf("%ld", &num);
}
printf("Zbir unetih brojeva je %ld.\n", sum);
}
```



```
c:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite broj za sumiranje
Ili q za izlaz
1
Unesite naredni broj za sabiranje
Unesite q za quit!
2
Unesite naredni broj za sabiranje
Unesite q za quit!
3
Unesite naredni broj za sabiranje
Unesite q za quit!
4
Unesite naredni broj za sabiranje
Unesite q za quit!
5
Unesite naredni broj za sabiranje
Unesite q za quit!
q
Zbir unetih brojeva je 15.
Press any key to continue_
```

**29. Sastaviti program na C jeziku za izračunavanje:  
//Neparnih brojeva sa for petljom**

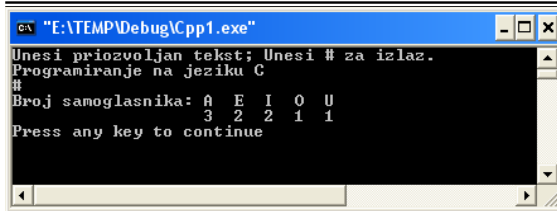
```
#include <stdio.h>
void main()
{
int broj;
for (broj=1;broj<=20;broj=broj+2)
printf("\n%d",broj);
}
```

### 30. Sastaviti program na C jeziku za izračunavanje: //naredbe break sa naredbom switch

```

#include <stdio.h>
void main()
{
char ch;
int a_ct,e_ct,i_ct,o_ct,u_ct;
a_ct=e_ct=i_ct=o_ct=u_ct=0;
printf("Unesi priozvoljan tekst; Unesi # za izlaz.\n");
while((ch=getchar())!='#')
{
switch (ch)
{
case 'a' :
case 'A' : a_ct++;
break;
case 'e' :
case 'E' : e_ct++;
break;
case 'i' :
case 'I' : i_ct++;
break;
case 'o' :
case 'O' : o_ct++;
break;
case 'u' :
case 'U' : u_ct++;
break;
default:
break;
} /* kraj switch */
} /* dok petlji nije kraj */
printf("Broj samoglasnika: A E I O U\n");
printf("          %d %d %d %d %d\n", a_ct,e_ct,i_ct,o_ct,u_ct);
}

```



```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Unesi proizvoljan tekst; Unesi # za izlaz.
Programiranje na jeziku C
#
Broj samoglasnika: A E I O U
3 2 2 1 1
Press any key to continue

```

### 31. Sastaviti program na C jeziku za izračunavanje: //ispisivanje brojeva Morzeovom azbukom

```

#include <stdio.h>
int main()
{ int broj;
printf("Unesi jedan broj izmedju 0 i 9:");
scanf("%d",&broj);
if((broj<0)||(broj>9))
{ printf("Broj nije u opsegu izmedju 0 i 9\n");}
else
{ printf("Broj %d prikazan Morzeovom azbukom je:",broj);}
if(broj==0) printf("----");
if(broj==1) printf("----");
if(broj==2) printf("..---");
if(broj==3) printf("...--");
if(broj==4) printf("....-");
if(broj==5) printf(".....");
if(broj==6) printf("-....");
if(broj==7) printf("--...");
if(broj==8) printf("---..");
if(broj==9) printf("----.");
return 0;
}

```



```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Unesi jedan broj izmedju 0 i 9:8
Broj 8 prikazan Morzeovom azbukom je:---..Press any key to continue

```

### 32. Sastaviti program na C jeziku za izračunavanje: sume prvih 20 prirodnih brojeva.

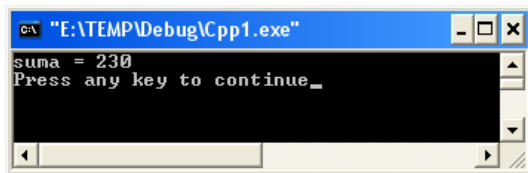
```

#include <stdio.h>
int main(void) /* suma prvih 20 prirodnih brojeva */
{
int brojac,suma; /* deklaracijska naredba */
brojac=1; /* naredba pridruzivanja */
suma =0; /* isto */
while(brojac++ < 21) /* while */
suma += brojac; /* naredba */
}

```



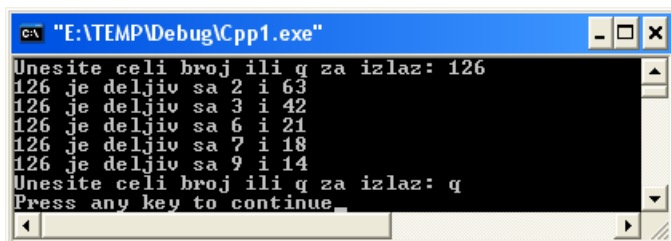
```
printf("suma = %d\n",suma); /* funkcijska naredba */
return 0;
}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
suma = 230
Press any key to continue_
```

### 33. Sastaviti program na C jeziku za izračunavanje: za zadani prirodan broja treba ispisati sve njegove delitelje.

```
#include <stdio.h>
int main(void)
{
    long num; // broj koji proveravamo
    long div; // potencijalni delitelj
    unsigned flag = 0; // prim broj?
    printf("Unesite celi broj ili q za izlaz: ");
    while(scanf("%ld",&num) == 1)
    {
        for(div=2; (div*div) <= num; ++div)
        {
            if(num % div == 0)
            {
                if(div * div != num)
                    printf("%d je deljiv sa %d i %d\n", num, div,
                        num/div);
                else
                    printf("%d je deljiv s %d.\n", num, div);
                flag=1;
            }
        }
        if(flag == 0) printf("%ld je prom broj.\n",num);
        printf("Unesite celi broj ili q za izlaz: ");
    }
    return 0;
}
```

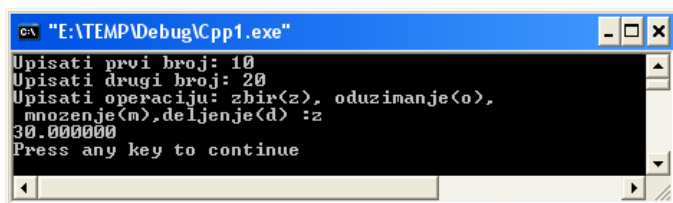


```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite celi broj ili q za izlaz: 126
126 je deljiv sa 2 i 63
126 je deljiv sa 3 i 42
126 je deljiv sa 6 i 21
126 je deljiv sa 7 i 18
126 je deljiv sa 9 i 14
Unesite celi broj ili q za izlaz: q
Press any key to continue_
```

**34. Sastaviti program na C jeziku za izračunavanje:**

Učitavaju se dva broja i jedan znak koji predstavlja izbor računске operacije. U zavisnosti od učitanoг znaka izvršava se jedna od četiri računске operacije + - \* /.

```
#include <stdio.h>
int main(void)
{
float a,b;
char operacija;
printf("Upisati prvi broj: ");
scanf(" %f",&a);
printf("Upisati drugi broj: ");
scanf(" %f",&b);
printf("Upisati operaciju: zbir(z), oduzimanje(o),\n");
printf(" mnozenje(m),deljenje(d) :");
scanf(" %c",&operacija);
if(operacija=='z')
printf("%f\n",a+b);
else if(operacija=='o')
printf("%f\n",a-b);
else if(operacija=='m')
printf("%f\n",a*b);
else if(operacija=='d')
printf("%f\n",a/b);
else
printf("Nedopustena operacija!\n");
return 0;
}
```



```

c:\ E:\TEMP\Debug\Cpp1.exe
Upisati prvi broj: 10
Upisati drugi broj: 20
Upisati operaciju: zbir(z), oduzimanje(o),
mnozenje(m), deljenje(d) :z
30.000000
Press any key to continue

```

**35. Sastaviti program na C jeziku za izračunavanje:**

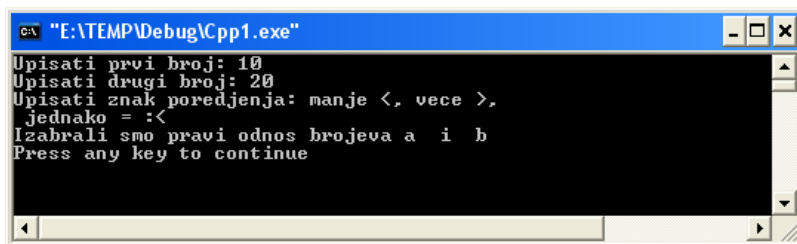
Učitavaju se dva broja i jedan znak koji predstavlja izbor njihovog odnosa (< manje, > veće, = jednako). U zavisnosti od učitanoг znaka ispituje se jedan od tri odnosa za poređenje ova dva broja. Ako je odnos tačan štampati poruku "Izabrali smo pravi odnos brojeva a i b", a ako nije "Izabrali smo pogresan odnos brojeva a i b". U slučaju da se unese pogrešan znak za poređenje štampati poruku "Nedopusteni znak za odnos brojeva!".

```
#include<stdio.h>
#include <stdio.h>
int main(void)
{
float a,b;
char znak;
printf("Upisati prvi broj: ");
scanf(" %f",&a);
```

```

printf("Upisati drugi broj: ");
scanf(" %f",&b);
printf("Upisati znak poredjenja: manje <, vece >,\n");
printf(" jednako = :");
scanf(" %c",&znak);
if(znak=='<')
{if(a<b) printf("Izabrali smo pravi odnos brojeva a i b\n");
else printf("Izabrali smo pogresan odnos brojeva a i b\n");}
else if(znak=='>')
{if(a>b) printf("Izabrali smo pravi odnos brojeva a i b\n");
else printf("Izabrali smo pogresan odnos brojeva a i b\n");}
else if(znak=='=')
{if(a==b) printf("Izabrali smo pravi odnos brojeva a i b\n");
else printf("Izabrali smo pogresan odnos brojeva a i b\n");}
else
{printf("Nedopusteni znak za odnos brojeva!\n");}
return 0;
}

```



```

E:\TEMP\Debug\Cpp1.exe
Upisati prvi broj: 10
Upisati drugi broj: 20
Upisati znak poredjenja: manje <, vece >,
jednako = :<
Izabrali smo pravi odnos brojeva a i b
Press any key to continue

```

### 36. Sastaviti program na C jeziku za izračunavanje: Srednje vrednostin upisanih brojeva.

```

#include <stdio.h>
/* Srednja vrednost upisanih brojeva (razlicitih od 0). */
int main(void)
{
int i=0;
double sum=0.0,x[];
printf(" Upisite niz brojeva !=0, ili nulu za kraj.\n");
printf(" x[0]= "); scanf("%lf",&x);
while (x!=0.0){
sum+=x;
printf(" x[%d]= ",++i);
scanf("%lf",&x);
}
sum/=i;
printf(" Srednja vrednost = %f\n",sum);
return 0;
}

```

```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Upisite niz brojeva !=0, ili nulu za kraj.
x[0]= 1
x[1]= 2
x[2]= 3
x[3]= 4
x[4]= 5
x[5]= 6
x[6]= 7
x[7]= 8
x[8]= 0
Srednja vrednost = 4.500000
Press any key to continue_

```

**37. Sastaviti program na C jeziku za izračunavanje:**

**/\* najveceg zajednickog delioca dva nenegativna cela broja \*/**

```

#include<stdio.h>
void main()
{
int prvi, drugi, pomocni;
printf("Ukucajte dva nenegativna cela broja ?\n");
scanf("%d%d", &prvi, &drugi );
while (drugi==0)
{
pomocni = prvi % drugi;
prvi = drugi;
drugi = pomocni;
}
printf("Najveci zajednicki delilac ovih brojeva je %d\n", prvi);
}

```

```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Ukucajte dva nenegativna cela broja ?
10
20
Najveci zajednicki delilac ovih brojeva je 10
Press any key to continue_

```

**38. Sastaviti program na C jeziku za izračunavanje:**

**/\* permutovanje cifara celog broja - while iskaz \*/**

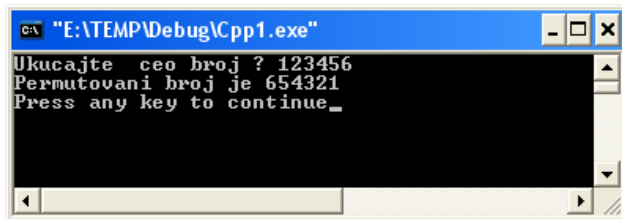
```

#include<stdio.h>
void main()
{
int broj;

printf("Ukucajte ceo broj ? ");
scanf("%d", &broj );
printf("Permutovani broj je ");
while ( broj )
{

```

```
printf("%d", broj % 10);  
broj = broj/10;  
}  
printf("\n");  
}
```

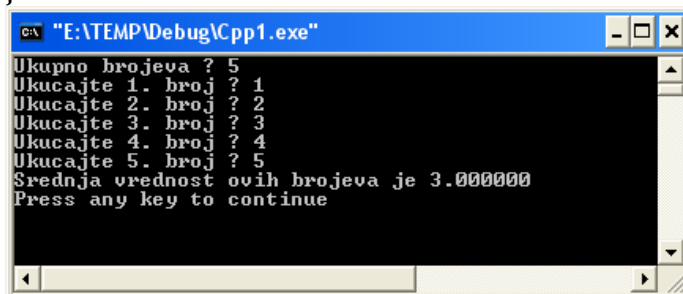


```
C:\ "E:\TEMP\Debug\Cpp1.exe"  
Ukucajte ceo broj ? 123456  
Permutovani broj je 654321  
Press any key to continue_
```

### 39. Sastaviti program na C jeziku za izračunavanje:

*/\* srednje vrednosti n celih pozitivnih brojeva -while iskaz \*/*

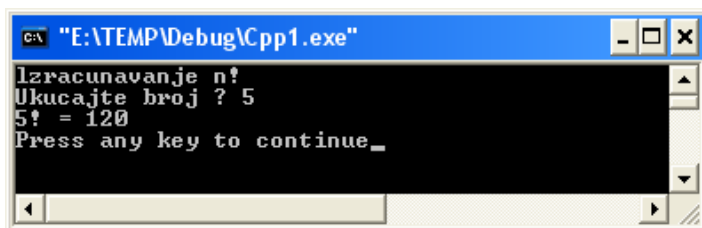
```
#include<stdio.h>  
void main()  
{  
int n, brojac = 0;  
float suma = 0, x;  
printf("Ukupno brojeva ? ");  
scanf("%d", &n);  
while (brojac < n)  
{  
printf("Ukucajte %d. broj ? ", brojac+1);  
scanf("%f", &x );  
suma += x ;  
brojac += 1;  
}  
printf("Srednja vrednost ovih brojeva je %f\n", suma/n);  
}
```



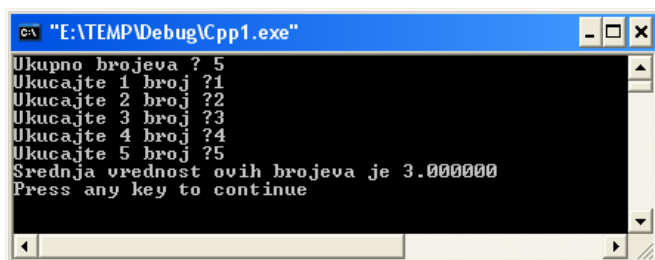
```
C:\ "E:\TEMP\Debug\Cpp1.exe"  
Ukupno brojeva ? 5  
Ukucajte 1. broj ? 1  
Ukucajte 2. broj ? 2  
Ukucajte 3. broj ? 3  
Ukucajte 4. broj ? 4  
Ukucajte 5. broj ? 5  
Srednja vrednost ovih brojeva je 3.000000  
Press any key to continue
```

**40. Sastaviti program na C jeziku za izračunavanje:****/\* faktoriijela \*/**

```
#include <stdio.h>
#include <math.h>
void main()
{
    int i, n;
    long fak = 1;
    printf("Izracunavanje n!\nUkucajte broj ? ");
    scanf("%d", &n);
    for ( i=1; i<=n; ++i)
        fak *= i;
    printf("%d! = %ld\n", n, fak);
}
```

**41. Sastaviti program na C jeziku za izračunavanje:****/\* srednje vrednosti n celih pozitivnih brojeva -for iskaz \*/**

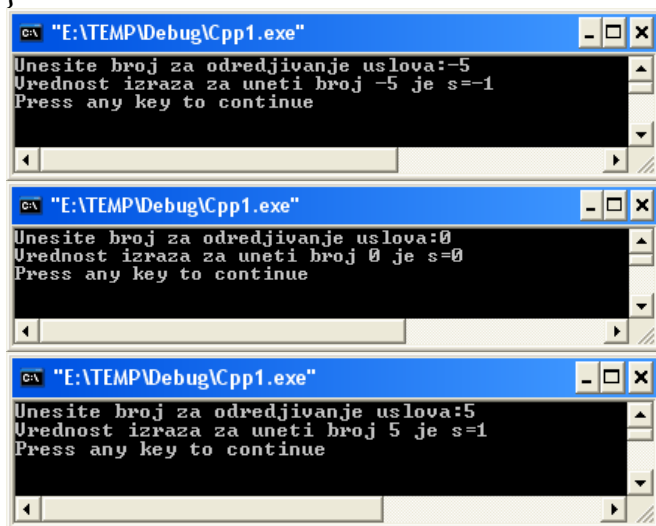
```
#include<stdio.h>
void main()
{
    int n, brojac =1;
    float suma =0, x ;
    printf("Ukupno brojeva ? ");
    scanf("%d", &n);
    for ( ; brojac<=n; ++brojac, suma+=x )
    {
        printf("Ukucajte %d broj ?",brojac);
        scanf("%f",&x);
    }
    printf("Srednja vrednost ovih brojeva je %f\n", suma/n);
}
```



**42. Sastaviti program na C jeziku za izračunavanje: funkcije primenom operatora uslovnog izraza:**

$$s = \begin{cases} -1 & broj < 0 \\ 0 & broj = 0 \\ 1 & broj > 0 \end{cases}$$

```
#include<stdio.h>
void main()
{
int s, broj;
printf("Unesite broj za odredjivanje uslova:");
scanf("%d",&broj);
s=(broj < 0) ? -1 : ((broj==0) ? 0 : 1);
printf("Vrednost izraza za uneti broj %d je s=%d\n",broj,s);
}
```


**43. Sastaviti program na C jeziku za izračunavanje: funkcije primenom operatora uslovnog izraza:**

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

```
#include <stdio.h>
#include <iomanip.h>
#include <math.h>
int main()
{double x;
printf("Unesite vrednost za x:");
scanf("%lf",&x);
if (x==0.)
{ printf("Greska. Pogresan unos argumenta x !\n");
```

```

return 1;
} // Unesena je validna vrednost za x
double y;
y=(x>=1?log10(x) + 1.82:x*x + 7*x + 8.82);
//cout << setprecision(3) << setiosflags(ios :: fixed);
printf("Za unetu vrednost %lf x, vrednost izraza je y= %lf ",x,y);
return 0;
}

```

```

C:\temp\Debug\Cpp1.exe
Unesite vrednost za x:-2
Za unetu vrednost -2.000000 x, vrednost izraza je y= -1.180000
continue

```

#### 44. Sastaviti program na C jeziku za izračunavanje: funkcije primenom operatora uslovnog izraza:

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

```

// Program za izracunavanje vrednosti funkcije
#include<stdio.h>
void main()
{
int s, x;
printf("Unesite broj za odredjivanje uslova:");
scanf("%d",&x);
s=(x <= 2) ? x : ((x>3) ? x-1 : 2);
printf("Vrednost izraza za uneti broj %d je s=%d\n",x,s);
}

```

```

C:\temp\Debug\Cpp1.exe
Unesite broj za odredjivanje uslova:1
Vrednost izraza za uneti broj 1 je s=1
Press any key to continue

C:\temp\Debug\Cpp1.exe
Unesite broj za odredjivanje uslova:3
Vrednost izraza za uneti broj 3 je s=2
Press any key to continue

C:\temp\Debug\Cpp1.exe
Unesite broj za odredjivanje uslova:4
Vrednost izraza za uneti broj 4 je s=3
Press any key to continue

```

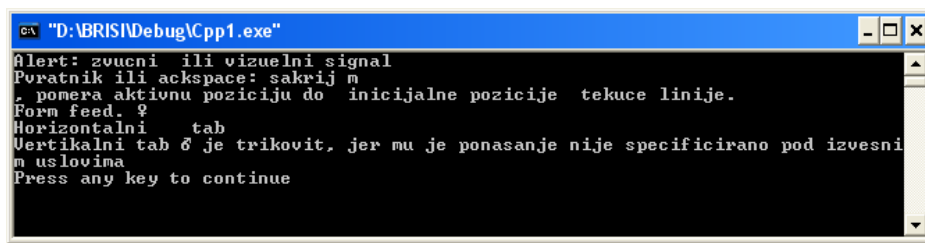


# PRIMERI PROGRAMA ZA PRIPREMU DRUGOG KOLOKVIJUMA

## 1. RAD SA NIZOVIMA KARAKTERA

### //PROGRAM 1. Rad sa specijalnim karakterima.

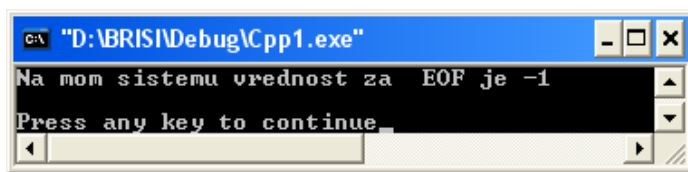
```
#include <stdio.h>
int main()
{
printf("Alert: zvucni ili vizuelni signal \a\n");
printf("Po\bvratnik ili b\backspace: sakrij me\b \n");
printf("Upravljacki znak carriage return, \r, pomera aktivnu poziciju do inicijalne pozicije tekuce linije.\n");
printf("Form feed. \f\n");
printf("Horizontalni\ttab\n");
printf("Vertikalni tab \v je trikovit, jer mu je ponasanje nije specificirano pod izvesnim uslovima\n");
return 0;
}
```



/\* Posle unosa sa tastature se za uneti EOF (uglavnom control-D ili control-Z karakter, ali ne obavezno), stampa 0. Inace, stampa se 1. Akoje Vas input stream baferovan (a verovatno jeste), onda morate pritisnuti ENTER taster pre nego dobijete izlaznu poruku \*/

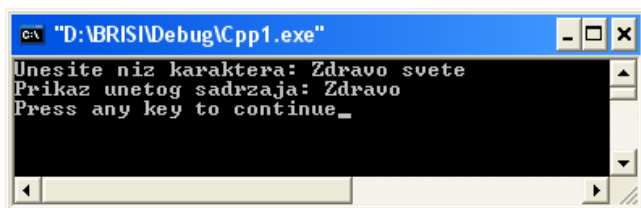
### //PROGRAM 2. Karakteristične vrednosti.

```
#include <stdio.h>
int main()
{
printf("Na mom sistemu vrednost za EOF je %d\n\n", EOF);
return 0;
}
```

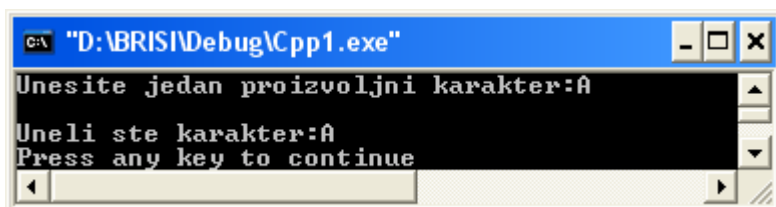
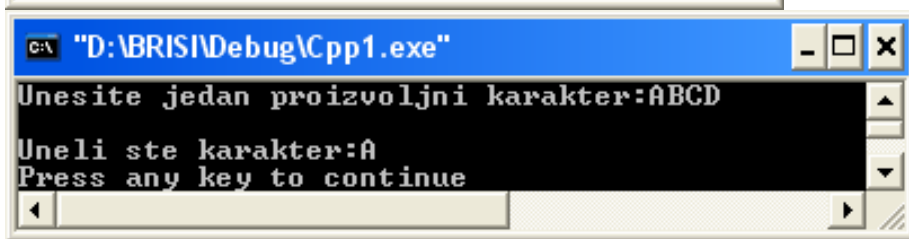


**//PROGRAM 3.**//Unos i štampanje karaktera

```
#include<stdio.h>//PRIMER 1.
int main()
{
char recenica[80];
printf("Unesite niz karaktera: ");
scanf("%s",&recenica);
printf("Prikaz unetog sadrzaja: %s\n",recenica);
return 0;
}
```


**//PROGRAM 4.**//Unos i štampanje karaktera

```
#include<stdio.h>
void main() {
char c;
printf("Unesite jedan proizvoljni karakter:");
c = getchar();
putchar('\n');//Prelazak u novi red
printf("Uneli ste karakter:");
putchar(c);//Stampanje unetog karaktera
putchar('\n');//Prelazak u novi red
}
```

**//PROGRAM 5.**//Unos i štampanje karaktera

```
#include<stdio.h>
void main() {
    char c[30];
    printf("Unesite niz proizvoljnih karaktera:");
    gets(c);
    printf("\n");//Prelazak u novi red
    printf("Uneli ste karakter:");
    puts(c);//Stampanje unetog karaktera
    printf("\n");//Prelazak u novi red
}
```

**//PROGRAM 6.**//Unos i štampanje karaktera

```
#include <stdio.h>
main()
{ int vrednost;
  vrednost='A';
  printf("%s\nkarakter=%3c\nvrednost=%3d\n","Veliko slovo",vrednost,vrednost);
  vrednost='a';
  printf("%s\nkarakter=%3c\nvrednost=%3d\n","Malo slovo",vrednost,vrednost); }
```

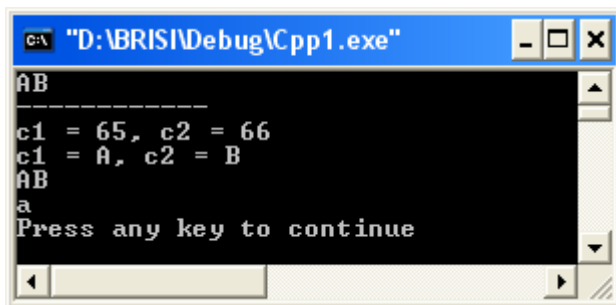
**//PROGRAM 7.**//Unos i štampanje karaktera

```
#include <stdio.h>
void main()
{
int c1, c2;
```

```

c1 = getchar();
printf("-----\n");
c2 = getchar();
printf("c1 = %d, c2 = %d\n",c1, c2);
printf("c1 = %c, c2 = %c\n",c1, c2);
putchar(c1); /* isto je kao i printf("%c",c1); */
putchar(c2); /* isto je kao i printf("%c",c2); */
putchar('\n');
/* Za ispisivanje karaktera a */
putchar('a');
/* dozvoljeno je : printf("abc"); printf("a"); */
/* nedozvoljeno je : printf('a'); putchar('abc'); putchar("abc"); */
}

```



```

D:\BRISI\Debug\Cpp1.exe
AB
c1 = 65, c2 = 66
c1 = A, c2 = B
AB
a
Press any key to continue

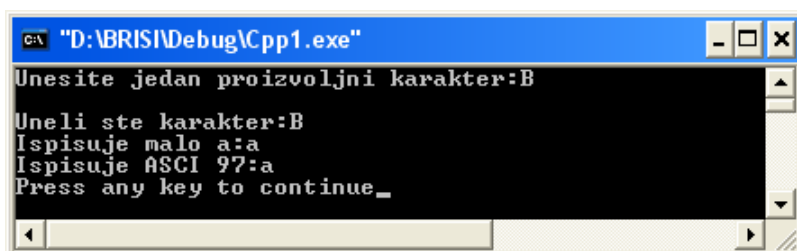
```

**//PROGRAM 8.**//Program čita jedan karakter i ispisuje ga.

```

#include <stdio.h>
void main()
{
int c; /* Karakter - obratiti paznju na int */
printf("Unesite jedan proizvoljni karakter:");
c = getchar(); /* cita karakter sa standardnog ulaza */
printf("\nUneli ste karakter:");
putchar(c); /* pise karakter zapamćen u promenljivoj c na standardni izlaz */
putchar('\n'); /* prelazak u novi red */
printf("Ispisuje malo a:");
putchar('a'); /* ispisuje malo a */
printf("\nIspisuje ASCI 97:");
putchar(97); /* ekvivalentno prethodnom */
putchar('\n'); /* prelazak u novi red */
}

```



```

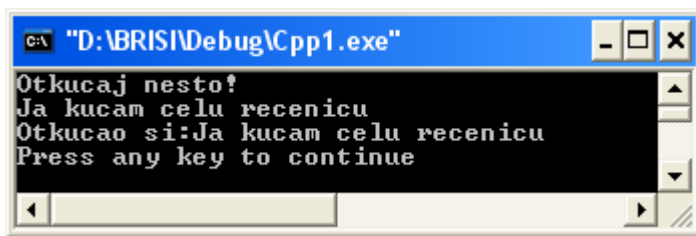
D:\BRISI\Debug\Cpp1.exe
Unesite jedan proizvoljni karakter:B
Uneli ste karakter:B
Ispisuje malo a:a
Ispisuje ASCI 97:a
Press any key to continue_

```

**//PROGRAM 9.**

//Program čita ceo izraz i ispisuje ga.

```
#include <stdio.h>
int main()
{
    char my_string[50];
    printf("Otkucaj nesto!\n");
    gets(my_string);
    printf("Otkucao si:%s\n", my_string);
    return 0;
}
```



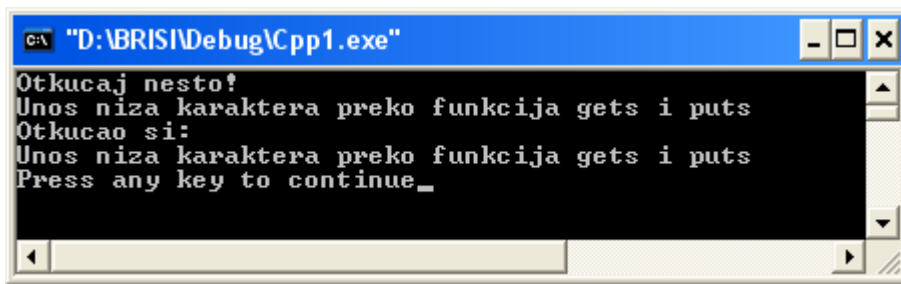
```
C:\ "D:\BRISI\Debug\Cpp1.exe"
Otkucaj nesto!
Ja kucam celu recenicu
Otkucao si:Ja kucam celu recenicu
Press any key to continue
```

**//PROGRAM 10.**

//Program čita ceo izraz i ispisuje ga pomoću funkcija gets i puts.

```
#include <stdio.h>

int main()
{
    char my_string[500];
    printf("Otkucaj nesto!\n");
    gets(my_string);
    printf("Otkucao si:\n");
    puts(my_string);
    return 0;
}
```



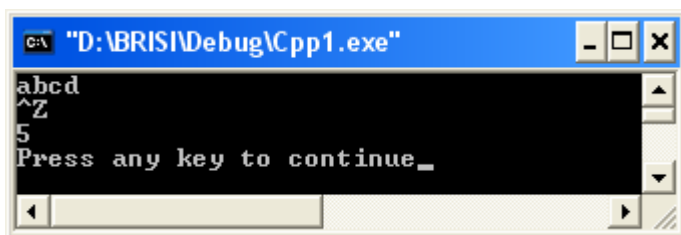
```
C:\ "D:\BRISI\Debug\Cpp1.exe"
Otkucaj nesto!
Unos niza karaktera preko funkcija gets i puts
Otkucao si:
Unos niza karaktera preko funkcija gets i puts
Press any key to continue_
```

**//PROGRAM 11.*****//Unos i štampanje karaktera***

Program učitava broj karaktera sa ulaza sve dok se ne otkuca znak za kraj unosa EOF. EOF se može generisati kucanjem Control /Z.

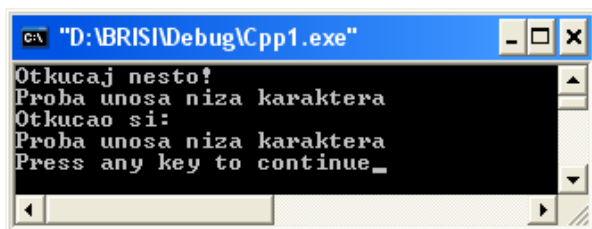
```
#include <stdio.h>
main()
{ int ch, i = 0;

  while((ch = getchar()) != EOF)
    i ++;
  printf("%d\n", i);
}
```

**//PROGRAM 12.*****//Unos i štampanje niza karaktera***

/\*Program koristi getchar da uči liniju sa standardnog ulaza stdin, postavlja takav ulaz u buffer, onda završava string pre štampanja linije na ekranu. \*/

```
#include <stdio.h>
void main( void )
{
  char buffer[81];
  int i, ch;
  printf( "Enter a line: " );
  /* Read in single line from "stdin": */
  for( i = 0; (i < 80) && ((ch = getchar()) != EOF)
        && (ch != '\n'); i++)
    buffer[i] = (char)ch;
  /* Terminate string with null character: */
  buffer[i] = '\0';
  printf( "%s\n", buffer );
}
```

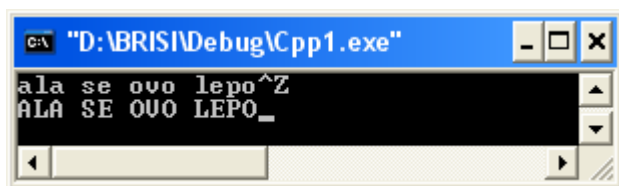


**//PROGRAM 13.****//Unos i štampanje karaktera**

Program konvertuje ulazne karaktere u velika slova. Da bi ovo učinili moramo dodati funkciju toupper iz biblioteke za konverziju karaktera ctype.h

```
#include <ctype.h> /* For definition of toupper */
#include <stdio.h> /* For definition of getchar, putchar, EOF */

void main()
{ int ch;
  while((ch = getchar()) != EOF)
    putchar(toupper(ch));
}
```

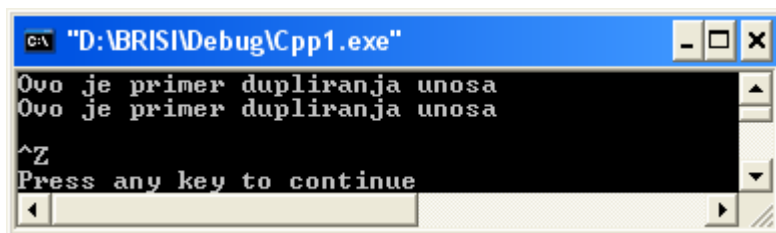
**//PROGRAM 14.****//Unos i štampanje karaktera**

Program koji koristi gets i puts to double space typed input.

```
#include <stdio.h>

void main()
{ char line[256]; /* Definiše string dužine 256 karaktera koji se pamti kao ulazna linija */

  while(gets(line) != NULL) /* Čitanje linije */
  { puts(line); /* Print linije */
    printf("\n"); /* Print prazne linije */
  }
}
```



**//PROGRAM 15.****//Unos i štampanje karaktera**

//Program stampa broj karaktera pre ispisivanja linije:

```
#include<stdio.h>
void main() {
    int n;
    char line[100];
    n = 0;
    while( (line[n++]=getchar()) != '\n' );
    line[n] = '\0';
    printf("%d:\t%s", n, line);
}
```

**//PROGRAM 16.****//Unos i štampanje karaktera**

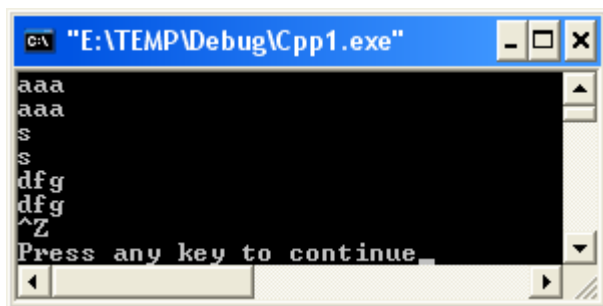
```
#include <stdio.h>
#include <string.h>
main()
{char linija_teksta[81];
while( fgets(linija_teksta,81, stdin)!= NULL)
/* Unos u liniju teksta se prekida i petlja sa kontrolnim karakterom Ctrl/z */
{continue;}
printf("Linija je : %s\n",linija_teksta);}
```



**//PROGRAM 17.**//Unos i štampanje karaktera

/\*(Funkcije getchar(), putchar() ) NCP koji sadržaj standardnog ulaza štampa znak po znak na standardni izlaz sve do markera kraja ulaza.\*/

```
#include <stdio.h>
main()
{   int znak;
    znak=getchar(); /*promenljivoj znak se dodeljuje znak sa ulaza */
    while( znak !=EOF) //End of file Ctrl/Z
    {       putchar(znak); /*dodeljeni znak se kopira na izlaz */
        znak=getchar(); } }
```

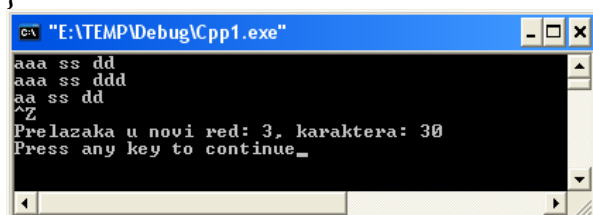


```
c:\ "E:\TEMP\Debug\Cpp1.exe"
aaa
aaa
s
s
dfg
dfg
^Z
Press any key to continue
```

**//PROGRAM 18.**//Unos i štampanje karaktera

/\* NCP koji će ispisati na standardni izlaz ukupan broj karaktera standardnog ulaza do markera kraja, kao i broj prelazaka u novi red. Pretpostaviti da je za ukupan broj karaktera (zajedno sa enter i blanko) i ukupan broj prelazaka u novi red dovoljan opseg long promenljivih.\*/

```
#include <stdio.h>
void main()
{
    int znak; /*prihvata znak sa ulaza */
    long linije=0; /*brojac linija */
    long br_znak=0; /*brojac znakova na ulazu */
    while ( (znak=getchar() ) != EOF)
    { br_znak++;
      if (znak=='\n') linije ++;
    }
    printf("Prelazaka u novi red: %ld, karaktera: %ld \n",linije,br_znak);
}
```



```
c:\ "E:\TEMP\Debug\Cpp1.exe"
aaa ss dd
aaa ss ddd
aa ss dd
^Z
Prelazaka u novi red: 3, karaktera: 30
Press any key to continue
```

**//PROGRAM 19.**

```

//Unos i štampanje karaktera
//Testira duzinu i jednakost dva stringa
#include <stdio.h>
#include <string.h>
void main()
{
char a[3], b[3],ch[3];
int i=0;
printf("Unesi priozvoljan tekst; Unesi # za izlaz!\n");
ch[i]=getchar();
while(ch[i]!='#')//izlazak iz petlje kada je uneto #
{putchar(ch[i]);i=i+1;//ispisuje trenutno uneti karakter direktno na izlaz
ch[i]=getchar();}
printf("\nNiz ch je: %s, kojih ima ukupno %d\n",ch,i) ;
strcpy(a, "abc");//kopiranje niza karaktera u vektor a
strcpy(b, "abc");//kopiranje niza karaktera u vektor b
printf("Ovako izgleda a niz:%s\n",a) ;
printf("Ovako izgleda b niz:%s\n",b) ;
printf("Ovako izgleda clan a[0] niza:%c\n", a[0]);
printf("Ovako izgleda clan a[1] niza:%c\n", a[1]);
printf("\nDuzina izraza a po broju karaktera je:%d\n",strlen(a)) //izracunavanje duzine niza
printf("\nDuzina izraza b po broju karaktera je:%d\n",strlen(b)) //izracunavanje duzine
niza
if (strcmp(a,b) == 0) //komparacija-uporedjivanje dva niza
{ printf("Ovo su jednaki NIZOVI!\n") ;}
else
printf("Ovo nisu jednaki NIZOVI!\n");
}

```

```

C:\Documents and Settings\Borivoje Milosevic\Desktop\Debug\Cpp1.exe
Unesi priozvoljan tekst; Unesi # za izlaz!
Ana voli Milovana#
Ana voli Milovana
Niz ch je: Ana voli Milovana#0, kojih ima ukupno 17
Ovako izgleda a niz:abc
Ovako izgleda b niz:abc
Ovako izgleda clan a[0] niza:a
Ovako izgleda clan a[1] niza:b

Duzina izraza a po broju karaktera je:3
Duzina izraza b po broju karaktera je:3
Ovo su jednaki NIZOVI!
Press any key to continue_

```

**//PROGRAM 20.****//Unos i štampanje karaktera****//Program za prikazivanje tablice ASCII kodova:**

```

#include <stdio.h>
main()
{
    char c=' ';
    int i;
    printf ("\t\tTablica ASCII kodova \n \n");
linija:
    i=0;
znak:
    printf ("%3d %c  ",c+i,c+i);
    i=i+19;
    if (i<95) goto znak;
    printf ("\n");
    c=c+1;
    if (c<' '+19) goto linija;
}

```

```

D:\BRIS\Debug\Cpp1.exe
Tablica ASCII kodova
32      51 3      70 F      89 Y      108 l
33     ?      52 4      71 G      90 Z      109 m
34     "      53 5      72 H      91 [      110 n
35     #      54 6      73 I      92 \      111 o
36     $      55 7      74 J      93 ]      112 p
37     %      56 8      75 K      94 ^      113 q
38     &      57 9      76 L      95 _      114 r
39     '      58 :      77 M      96 `      115 s
40     <      59 ;      78 N      97 a      116 t
41     >      60 <      79 O      98 b      117 u
42     *      61 =      80 P      99 c      118 v
43     +      62 >      81 Q     100 d      119 w
44     ,      63 ?      82 R     101 e      120 x
45     -      64 @      83 S     102 f      121 y
46     .      65 A      84 T     103 g      122 z
47     /      66 B      85 U     104 h      123 <
48     0      67 C      86 V     105 i      124 |
49     1      68 D      87 W     106 j      125 }
50     2      69 E      88 X     107 k      126 ~
Press any key to continue_

```

**//PROGRAM 21.****//NCP koji pronalazi najduzu liniju sa ulaza. Nije poznat ukupan broj linija, ali svaka linija nema vise od 80 karaktera ( izlaz CTRL/Z).**

```

#include <stdio.h>
#include <string.h>
#define MAX_DUZINA_LINIJE 81

main()
{
char linija[MAX_DUZINA_LINIJE];

```

```

char najduza_linija[MAX_DUZINA_LINIJE];
int duzina_linije = 0;
int duzina_najduze_linije = 0;
najduza_linija[0] = '\0';
while( fgets(linija, MAX_DUZINA_LINIJE, stdin) != NULL)
{
    duzina_linije=strlen(linija);
    if (duzina_linije>duzina_najduze_linije)
    {strcpy(najduza_linija, linija);
    duzina_najduze_linije = duzina_linije;}
}
printf("Najduza linija je : %s\n",najduza_linija);
}

```

**//PROGRAM 22.**

// NCP koji stampa liniju sa ulaza pod uslovom da linija nema vise od 80 karaktera.

```

#include <stdio.h>
void main( void )
{
    char buffer[81];
    int i, ch;
    printf( "Unesi liniju teksta: " );

    /* Citanje linije karaktera sa standardnog "stdin": */
    for( i = 0; (i < 80) && ((ch = getchar()) != EOF)
        && (ch != '\n'); i++)
        buffer[i] = (char)ch;
    /* String se terminira sa NULL karakterom: */
    buffer[i] = '\0';
    printf( "%s\n", buffer );
}

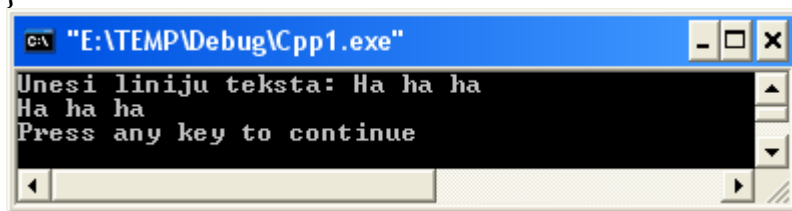
```

**//PROGRAM 23.**

//NCP koji stampa liniju konstantnog niza karaktera.

```
#include <stdio.h>
void main( void )
{
    char buffer[81];
    int i, ch;
    printf( " Unesi liniju teksta: " );

    /* Citanje linije karaktera sa standardnog "stdin": */
    for( i = 0; (i < 80) && ((ch = getchar()) != EOF)
        && (ch != '\n'); i++)
        buffer[i] = (char)ch;
    /* String se terminira sa NULL karakterom: */
    buffer[i] = '\0';
    printf( "%s\n", buffer );
}
```

**//PROGRAM 24.**

/\*Uraditi program za unos niza karaktera "Moram da položim drugi kolokvijum iz Programskih jezika I" u vektor Niz1. Odštampati ovaj niz karaktera u potpunoj formi, i izračunati dužinu ovog unetog niza. Iskopirati sadržaj vektora Niz1 u vektor Niz2 i utvrditi dali su to isti nizovi. Prebrojati koliko praznina (blanko karaktera) ima u nizu Niz1.\*/

```
#include <stdio.h>
#include <string.h>
void main()
{ int brojznak=0,i=0;
  char Niz1[81],Niz2[81];
  printf("Unesi clanove niza Niz1, za kraj Ctrl/z/");
  while( fgets(Niz1,81, stdin)!= NULL)
  /* Unos u liniju teksta se prekida i petlja sa kontrolnim karakterom Ctrl/z */
  {continue;}
  printf("Linija niza Niz1 je : %s\n",Niz1);
  printf("\nDuzina niza po broju karaktera je:%d\n",(strlen(Niz1)-1));
  strcpy(Niz2,Niz1);
  printf("Linija niza Niz2 je : %s\n",Niz2);
  if (strcmp(Niz1,Niz2)==0)//Komparacija-uporedjivanje dva niza
  { printf("Ovo su jednaki NIZOVI!\n");}
  else
  printf("Ovo nisu jednaki NIZOVI!\n");
  while (Niz1[i]!='\n')
  { if(Niz1[i]==' ')brojznak++;i=i+1;}
  printf("Broj blanko znakova u nizu Niz1 je:%ld \n",brojznak); }
```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesi clanove niza Niz1, za kraj Ctrl/z/Moram da polozi drugi kolokvijum iz Programskih jezika 1
^Z
Linija niza Niz1 je : Moram da polozi drugi kolokvijum iz Programskih jezika 1

Duzina niza po broju karaktera je:57
Linija niza Niz2 je : Moram da polozi drugi kolokvijum iz Programskih jezika 1

Ovo su jednaki NIZOUI?
Broj blanko znakova u nizu Niz1 je:8
Press any key to continue_

```

**//PROGRAM 25.**

/\*Sastaviti program koji iz skupa unetih karaktera ( naPRIMER: rečenice "Ja sam rođen 1990. god u 12. mesecu, i u petak!!" izračunava broj unetih slova, brojeva i ostalih karaktera.\*/

```

#include<stdio.h>
#include<conio.h>
main()
{
int slovo, broj, ostalo, c;
slovo = broj = ostalo = 0;
while( (c=getchar()) != EOF )
if( ('A'<=c && c<='Z') || ('a'<=c && c<='z') )
++slovo;
else if( '0'<=c && c<='9' ) ++broj;
else ++ostalo;
printf("Uneli ste:%d slova, %d brojeva,%d ostalih\n", slovo, broj, ostalo);
}

```

```

C:\ "E:\BMilosevic Stari Disk\Programski jezici C++\C Lessons\Debug\Funk1.exe"
Ja sam rođen 1990. god u 12. mesecu, i u petak!!
^Z
Uneli ste:28 slova, 6 brojeva,16 ostalih
Press any key to continue

```

**//PROGRAM 26.**

```

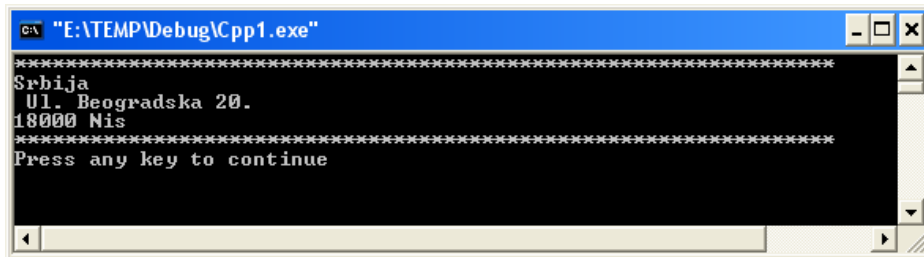
//Formatiranje izlaza
#include <stdio.h>
#define IME "Srbija"
#define ADRESA "Ul. Beogradska 20."
#define MESTO "18000 Nis"
#define LIMIT 65
void main()
{void zvezde(void);/*deklaracija funkcije bez argumenata*/
zvezde(); /*poziv korisnicke funkcije */
printf("%s\n",IME); /*poziv funkcije iz standardne biblioteke*/
}

```

```

printf("%s\n", ADRESA);
printf("%s\n", MESTO);
zvezde(); /*definicija korisnicke funkcije*/
void zvezde() /*funkcija nema argumenata*/
{ int brojac;
for(brojac=1;brojac<=LIMIT;brojac++)
putchar ('*');
putchar ('\n'); }

```



```

C:\ "E:\TEMP\Debug\Cpp.exe"
*****
Srbija
Ul. Beogradska 20.
18000 Nis
*****
Press any key to continue

```

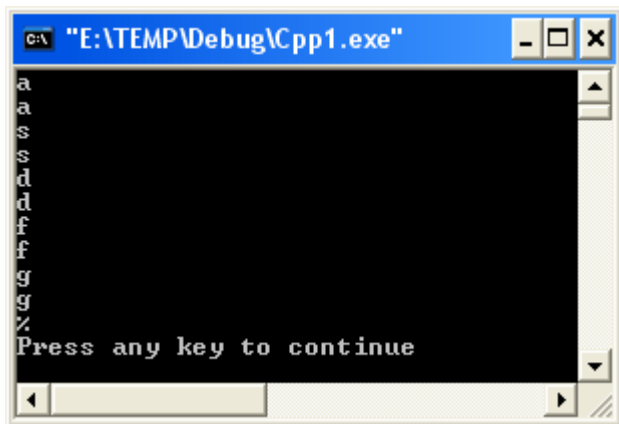
**//PROGRAM 27.**

//Program koji kopira ulaz svakog karaktera direktno na izlaz. '%' označava kraj ulaza.

```

#include <stdio.h>
void main() {
    char c;
    while( (c=getchar()) != '%' )
        putchar(c);
}

```



```

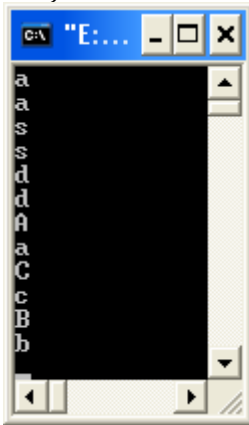
C:\ "E:\TEMP\Debug\Cpp1.exe"
a
a
s
s
d
d
f
f
g
g
%
Press any key to continue

```

**//PROGRAM 28.**

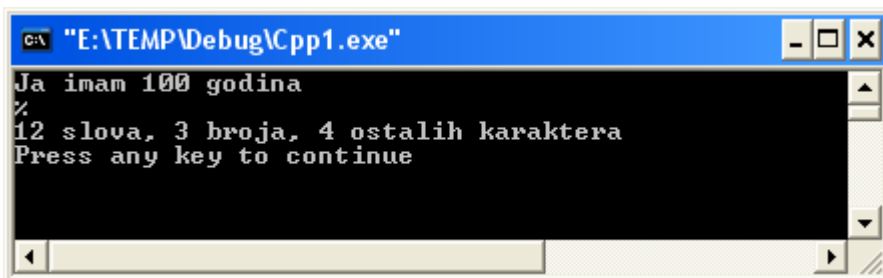
//Program konvertuje velika u mala slova

```
#include <stdio.h>
void main() {
    char c;
    while( (c=getchar()) != '%' )
        if( 'A'<=c && c<='Z' )
            putchar(c+'a'-'A');
        else
            putchar(c);
}
```

**//PROGRAM 29.**

//Program broji karaktere, brojeve i praznine u tekstu. Završava se unosom %.

```
#include <stdio.h>
void main() {
    int let, dig, other, c;
    let = dig = other = 0;
    while( (c=getchar()) != '%' )
        if( ('A'<=c && c<='Z') || ('a'<=c && c<='z') )
            ++let;
        else if( '0'<=c && c<='9' ) ++dig;
        else ++other;
    printf("%d slova, %d broja, %d ostalih karaktera\n", let, dig, other);
}
```





**//PROGRAM 30.****//Izračunavanje dužine zapsanog stringa, IZLAZ ENTER.**

```

#include <stdio.h>
void main() {
    int n, c;
    char line[100];
    n = 0;
    while( (c=getchar()) != '\n' ) {
        if( n < 100 )
            line[n] = c;
        n++;
    }
    printf("duzina stringe je: = %d\n", n);
}

```

**//PROGRAM 31.****//Dužina stringa sa izlazom ENTER**

```

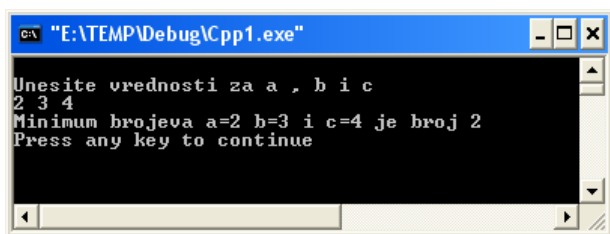
void main() {
    int n;
    char line[100];
    n = 0;
    while( (line[n++]=getchar( )) != '\n' );// Karakter \n kao ENTER za kraj
    line[n] = '\0';
    printf("Duzine stringa je %d:\t%s", n, line);
}

```

## 2. OPŠTI ZADACI

### /\* 2.1 MINIMUM TRI CELA BROJA \*/

```
#include<stdio.h>
#include<math.h>
void main()
{int a,b,c,m;
printf("\nUnesite vrednosti za a , b i c\n");
scanf("%d%d%d",&a,&b,&c);
m=a;
if(b<m) m=b;
if(c<m) m=c;
printf("Minimum brojeva a=%d b=%d i c=%d je broj %d\n",a,b,c,m);}
```

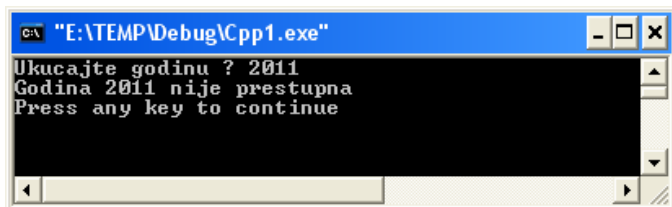


```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite vrednosti za a , b i c
2 3 4
Minimum brojeva a=2 b=3 i c=4 je broj 2
Press any key to continue
```

### /\* 2.2 Program za odredjivanje da li je godina prestupna ili nije\*/

```
#include<stdio.h>
#include<math.h>
void main()
{
int godina,ost_4,ost_100,ost_400;
printf("Ukucajte godinu ? ");
scanf("%d", &godina);
ost_4=godina%4;
ost_100=godina%100;
ost_400=godina%400;
if (ost_4 ==0 && ost_100 != 0 || ost_400== 0)
printf("Godina %d je prestupna\n", godina);
else
printf("Godina %d nije prestupna\n", godina);
}
```



```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Ukucajte godinu ? 2011
Godina 2011 nije prestupna
Press any key to continue
```

**/2.3\* Program za permutovanje cifara  
nenegativnog celog broja - do-while iskaz \*/**

```
#include<stdio.h>
#include<math.h>
void main()
{int broj;
printf("Ukucajte ceo broj ? ");
scanf("%d", &broj );
printf("Permutovani broj je ");
do
{printf("%d", broj%10);
broj =broj/10;}
while (broj);
printf("\n"); }
```

**/2.4\* Program za odredjivanje srednje vrednosti n celih pozitivnih brojeva -for iskaz \*/**

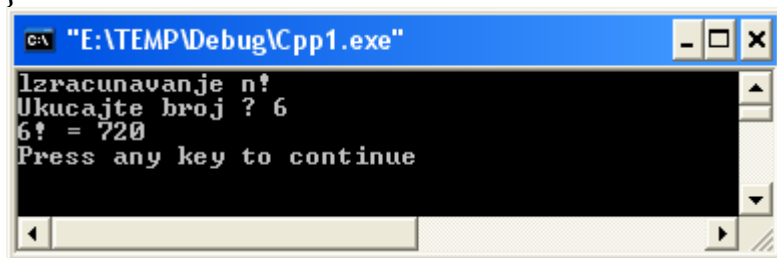
```
#include<stdio.h>
#include<math.h>
void main()
{
int n,brojac;
float suma =0, x ;
printf("Ukupno brojeva ? ");
scanf("%d", &n);
for (brojac=1 ; brojac<=n; ++brojac, suma+=x )
{
printf("Ukucajte %d broj ?",brojac);
scanf("%f",&x);
}
printf("Srednja vrednost ovih brojeva je %f\n", suma/n);
}
```

**/2.5\* Program za izracunavanje faktoriijela \*/**

```

#include <stdio.h>
#include <math.h>
void main()
{
int i, n;
long fak = 1;
printf("Izracunavanje n!\nUkucajte broj ? ");
scanf("%d", &n);
for ( i=1; i<=n; ++i)
fak *= i;
printf("%d! = %ld\n", n, fak);
}

```

**/2.6\* Program za izracunavanje prostih aritmetickih izraza u formi operand1 operator operand2 \*/**

```

#include <stdio.h>
#include <math.h>
void main()
{
float operand1, operand2;
char op;
printf("Ukucajte Izraz ? \n");
scanf("%f%c%f", &operand1, &op, &operand2);
switch (op)
{
case '+':
printf("%f\n", operand1+operand2);
break;
case '-':
printf("%f\n", operand1-operand2 );
break;
case '*':
printf("%f\n", operand1*operand2 );
break;
case '/':
printf("%f\n", operand1/operand2);
break;
default:
printf("Nepoznat operator\n");
}
}

```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Ukucajle izraz ?
123.9-3.9
120.0000001
Press any key to continue_

```

**//2.7\*Odredjivanje u ulaznom tekstu broja praznih karaktera, kao i broja znakova: tacka, zarez, dvotacka i tacka-zarez ( naredbe switch i break) - za izlaz CTRL/Z\*/**

```

#include <stdio.h>
void main()
{ int c, prazno=0, interp=0;
while((c=getchar())!=EOF)
switch(c)
{ case ' ':
prazno++;
break;

case '.':
case ',':
case ':':
case ';':
interp++;
break;

default:
break; }
printf("\nBroj praznina: %d", prazno);
printf("\nBroj znakova tacka, zarez, dvotacka i tacka-zarez: %d\n", interp); }

```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Naprimjer: prvi, drugi, treci
^Z
Broj praznina: 3
Broj znakova tacka, zarez, dvotacka i tacka-zarez: 3
Press any key to continue_

```

**//2.8 Štampanje elemenata matrice**

```

#include<stdio.h>
#include<math.h>
void main()
{
static int mat[2][3]={{0,1,2},{3,4,5}};
int i,j;
for(i=1;i<=2;++i)

```

```

{
printf("Elementi %d.reda su:\n",i);
for(j=1;j<=3;++j)
printf("kolona%d:%d\n",j,mat[i-1][j-1]);
}
printf("\n");
}

```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Elementi 1.reda su:
kolona1:0
kolona2:1
kolona3:2
Elementi 2.reda su:
kolona1:3
kolona2:4
kolona3:5
Press any key to continue_

```

## 2.9 Neka je data kvadratna matrica u formatu:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Članove matrice uneti u program kao konstantne celobrojne veličine, a kao izlaz iz programa predvideti štampanje ove matrice u pokazanom formatu kao polazne matrice i štampanje nove matrice u istom obliku ali čiji su svi članovi sada pomnoženi skalarom 2.

```

/* 1 STAMPANJE CLANOVA MATRICE I CLANOVA MATRICE POMNOZENE
SKALAROM*/

```

```

#include<stdio.h>
#include<math.h>
void main()
{int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int i,j;
printf("\nPolazna matrica je\n");
for(i=0;i<4;i++)
{
for(j=0;j<4;j++)
{printf("%2d ",mat[i][j]);printf("\n");}
printf("\nNova matrica je\n");
for(i=0;i<4;i++)
{
for(j=0;j<4;j++)
{printf("%2d ",2*mat[i][j]);printf("\n");}}
}
}

```

```

E:\TEMP\Debug\Cpp1.exe
Polazna matrica je
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Nova matrica je
2 4 6 8
10 12 14 16
18 20 22 24
26 28 30 32
Press any key to continue_

```

## 2.10 Neka je data kvadratna matrica u formatu:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Članove matrice uneti u program kao konstantne celobrojne veličine, a kao izlaz iz programa predvideti štampanje ove matrice u pokazanom formatu kao polazne matrice i štampanje nove matrice u istom obliku ali sada unazad od člana  $a_{44}, a_{43}, a_{42}, a_{41}, \dots, a_{11}$ .

```
/* 2 STAMPANJE CLANOVA MATRICE I CLANOVA MATRICE UNAZAD*/
```

```
#include<stdio.h>
```

```
#include<math.h>
```

```
void main()
```

```
{int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
```

```
int i,j;
```

```
printf("\nPolazna matrica je\n");
```

```
for(i=0;i<4; i++)
```

```
{
```

```
for(j=0;j<4; j++)
```

```
{printf("%2d ",mat[i][j]);printf("\n");}
```

```
printf("\nNova matrica je\n");
```

```
for(i=3;i>=0; i--)
```

```
{
```

```
for(j=3;j>=0; j--)
```

```
{printf("%2d ",mat[i][j]);printf("\n");}}
```

```

E:\TEMP\Debug\Cpp1.exe
Polazna matrica je
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Nova matrica je
16 15 14 13
12 11 10 9
8 7 6 5
4 3 2 1
Press any key to continue

```

**2.11 Napisati program na C++ jeziku koji izračunava sumu svih elemenata u svakom redu pojedinačno, pravougaone matrice [4x4].**

A=	1	2	3	4	←
	5	6	7	8	←
	9	10	11	12	←
	13	14	15	16	

```

#include<stdio.h>
#include<math.h>
void main()
{int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int i,j,s=0;
printf("\nPolazna matrica je\n");
for(i=0;i<4; i++)
{
for(j=0;j<4; j++)
{printf("%2d ",mat[i][j]);printf("\n");}

for(i=0;i<4; i++)
{printf("\nSuma %d reda je:",i+1);
for(j=0;j<4; j++)

{s=s+mat[i][j];}
printf("%2d\n",s);s=0;}
return 0;}

```

```

E:\TEMP\Debug\Cpp1.exe
Polazna matrica je
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

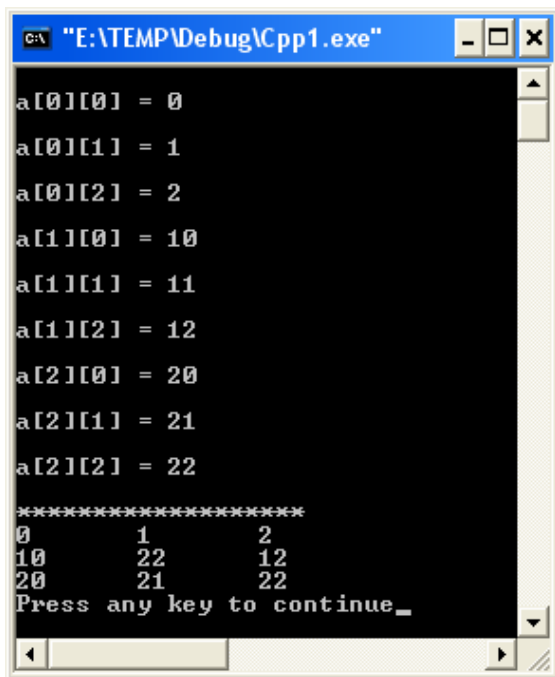
Suma 1 reda je:10
Suma 2 reda je:26
Suma 3 reda je:42
Suma 4 reda je:58
Press any key to continue_

```



2.12 NCP koji sa standardnog ulaza učitava 3x3 matricu i ispisuje je na standardni izlaz tako da centralno polje  $a[1][1]$  bude jednako sumi gornjeg levog ( $a[0][0]$ ) i donjeg desnog polja ( $a[2][2]$ ). Pre učitavanja popuniti matricu proizvoljnim vrednostima.

```
#include <stdio.h>
void main()
{
int a[3][3] = {{0, 1, 2}, {10, 11, 12}, {20, 21, 22}}; /* deklaracija i inicijalizacija matrice a, može
se uraditi za konkretan primer date matrice*/
int i, j; /*brojaci u ciklusu */
/* unos elemenata matrice */
for(i=0; i<3; i++)
for(j=0; j<3; j++)
{
printf("\na[%d][%d] = ", i, j);
scanf("%d", &a[i][j]);
}
printf("\n*****\n");
a[1][1] = a[0][0] + a[2][2]; /* 0 + 22 = 22 */
/*ispis matrice */
for(i=0; i<3; i++)
{
for(j=0; j<3; j++) printf("%d\t", a[i][j]);
printf("\n");
}
}
```



```
"E:\TEMP\Debug\Cpp1.exe"
a[0][0] = 0
a[0][1] = 1
a[0][2] = 2
a[1][0] = 10
a[1][1] = 11
a[1][2] = 12
a[2][0] = 20
a[2][1] = 21
a[2][2] = 22
*****
0      1      2
10     22     12
20     21     22
Press any key to continue_
```

**2.13 Neka je data kvadratna matrica u formatu:**

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

Uneti članove matrice u program kao celobrojne veličine, a kao izlaz iz programa predvideti štampanje takve matrice u pokazanom formatu kao i štampanje izračunate vrednosti njene determinante uradene preko funkcije.

```
#include<stdio.h>
#include<math.h>
#include<iomanip.h>
main()
{static int mat[4][4];
int i,j;
float delta;
printf("Unesite proizvoljne elemente pravougle matrice:\n");
for(i=0;i<4;++i)
for(j=0;j<4;++j)
{printf("a[%d][%d]=",i,j);
scanf("%d",&mat[i][j]);printf("\n");}
printf("\nPolazna matrica je\n");
for(i=0;i<4;++i)
{for(j=0;j<4;++j)
printf("%3d ",mat[i][j]);printf("\n");}
printf("\nDeterminanta sistema je:\n");
delta=mat[0][0]*((mat[1][1]*mat[2][2]*mat[3][3]
+mat[1][2]*mat[2][3]*mat[3][1]
+mat[2][1]*mat[3][2]*mat[1][3])
-(mat[1][3]*mat[2][2]*mat[3][1]
+mat[1][1]*mat[2][3]*mat[3][2]
+mat[1][2]*mat[2][1]*mat[3][3]))-
mat[0][1]*((mat[1][0]*mat[2][2]*mat[3][3]
+mat[1][2]*mat[2][3]*mat[3][0]
+mat[2][0]*mat[3][2]*mat[1][3])
-(mat[1][3]*mat[2][2]*mat[3][0]
+mat[1][0]*mat[2][3]*mat[3][2]
+mat[1][2]*mat[2][0]*mat[3][3]))+
mat[0][2]*((mat[1][0]*mat[2][1]*mat[3][3]
+mat[1][3]*mat[2][0]*mat[3][1]
+mat[2][0]*mat[3][1]*mat[1][3])
-(mat[1][3]*mat[2][1]*mat[3][0]
+mat[3][1]*mat[2][3]*mat[1][0]
+mat[1][1]*mat[2][0]*mat[3][3]))-
mat[0][3]*((mat[1][0]*mat[2][1]*mat[3][2]
+mat[1][1]*mat[2][2]*mat[3][0]
+mat[2][0]*mat[3][1]*mat[1][2])
-(mat[1][2]*mat[2][1]*mat[3][0]
+mat[3][1]*mat[2][2]*mat[1][0]
+mat[1][1]*mat[2][0]*mat[3][2]));
printf("Vrednost determinante je :%f",delta);
return 0;}
```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
a[2][0]=9
a[2][1]=1
a[2][2]=2
a[2][3]=3
a[3][0]=4
a[3][1]=5
a[3][2]=6
a[3][3]=7

Polazna matrica je
 1  2  3  4
 5  6  7  8
 9  1  2  3
 4  5  6  7

Determinanta sistema je:
Vrednost determinante je :864.000000Press any key to continue

```

#### 2.14 Sastaviti program za izračunavanje sume:

$$S = \prod_{i=1}^n i!$$

**Napomena:** radi se o integer veličinama ali voditi računa da ne dođe do preopterećenja memorije!

```

#include<stdio.h>
#include<math.h>
main()
{int i,n;
float F=1,S=1;
printf("Unesite verdnost za n\n n=");
scanf("%d",&n);
for(i=2;i<=n;i++)
{F=F*i;
S=S*F;};
printf("\nRezultat proizvoda i! je S=%lf\n",S);
return 0;}

```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite verdnost za n
n=5

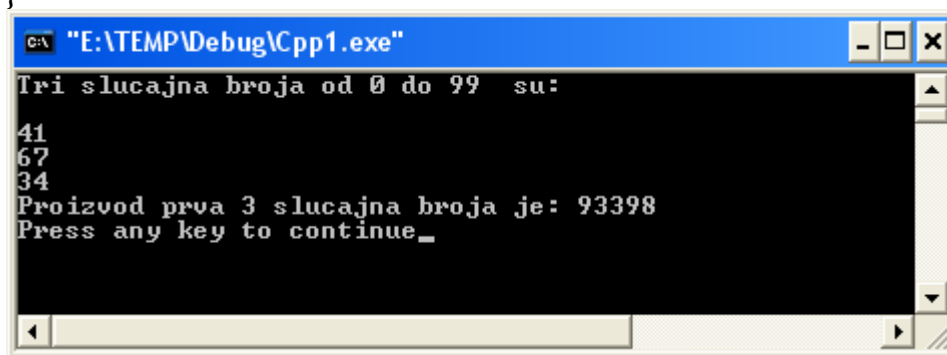
Rezultat proizvoda i! je S=34560.000000
Press any key to continue

```

**2.15 Napisati program za nalaženje proizvoda prva 3 slučajna broja.**

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int i,R;
    long int P=1;
    printf("Tri slucajna broja od 0 do 99 su:\n\n");
    for(i=0; i<3; i++)
        {R= rand()% 100;
        printf("%d\n",R );
        P=R*P;}
    printf("Proizvod prva 3 slucajna broja je: %d\n", P);
    return 0;
}
```



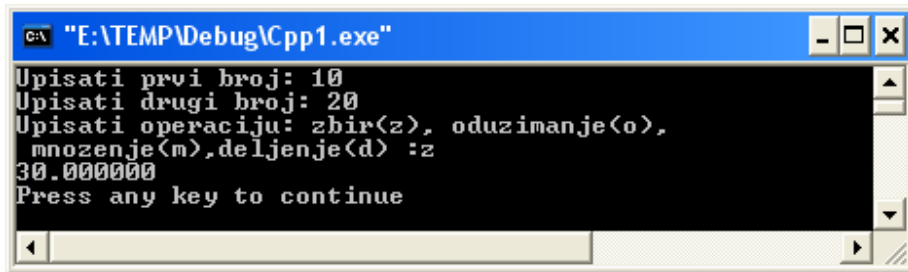
**2.16 U sledećem primeru učitavaju se dva broja i jedan znak koji predstavlja izbor njihovog odnosa (< manje, > veće, = jednako). U zavisnosti od učitanoog znaka ispituje se jedan od tri odnosa za poredenje ova dva broja. Ako je odnos tačan štampati poruku "Izabrali smo pravi odnos brojeva a i b", a ako nije "Izabrali smo pogresan odnos brojeva a i b". U slučaju da se unese pogrešan znak za poredenje štampati poruku "Nedopusteni znak za odnos brojeva!"**

```
#include <stdio.h>
int main(void)
{
    float a,b;
    char operacija;
    printf("Upisati prvi broj: ");
    scanf(" %f",&a);
    printf("Upisati drugi broj: ");
    scanf(" %f",&b);
    printf("Upisati operaciju: zbir(z), oduzimanje(o),\n");
    printf(" mnozenje(m),deljenje(d) :");
    scanf(" %c",&operacija);
    if(operacija=='z')
        printf("%f\n",a+b);
    else if(operacija=='o')
        printf("%f\n",a-b);
    else if(operacija=='m')
        printf("%f\n",a*b);
    else if(operacija=='d')
        printf("%f\n",a/b);
    else
        printf("Nedopusteni znak za odnos brojeva!\n");
}
```

```

printf("%f\n",a*b);
else if(operacija=='d')
printf("%f\n",a/b);
else
printf("Nedopustena operacija!\n");
return 0;
}

```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Upisati prvi broj: 10
Upisati drugi broj: 20
Upisati operaciju: zbir(z), oduzimanje(o),
mnozenje(m), deljenje(d) :z
30.000000
Press any key to continue

```

## 2.17 Sastaviti program na C jeziku za izračunavanje površine trougla po obrascu

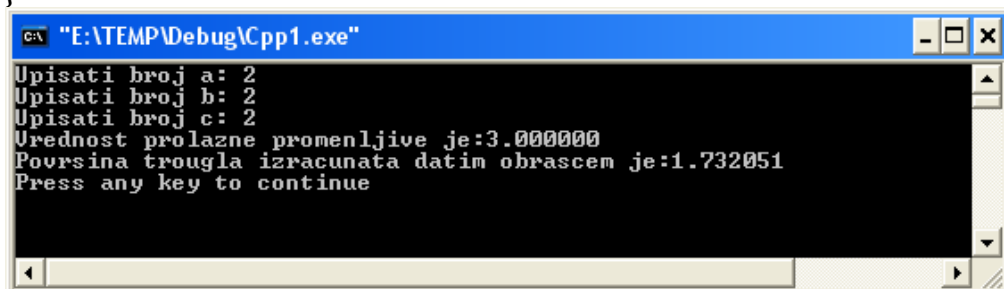
$$PTROUGLA = \sqrt{p(p-a)(p-b)(p-c)} \quad \text{gde je } p = \frac{a+b+c}{2}$$

```

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

float main(void)
{
float a,b,c,p,PTROUGLA;
printf("Upisati broj a: ");
scanf(" %f",&a);
printf("Upisati broj b: ");
scanf(" %f",&b);
printf("Upisati broj c: ");
scanf(" %f",&c);
p=(a+b+c)/2.;
printf("Vrednost prolazne promenljive je:%f\n",p);
PTROUGLA=sqrt(p*(p-a)*(p-b)*(p-c));
printf("Povrsina trougla izracunata datim obrascem je:%f\n",PTROUGLA);
return 0;
}

```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Upisati broj a: 2
Upisati broj b: 2
Upisati broj c: 2
Vrednost prolazne promenljive je:3.000000
Povrsina trougla izracunata datim obrascem je:1.732051
Press any key to continue

```

2.18 U sledećem primeru učitavaju se dva broja i jedan znak koji predstavlja izbor njihovog odnosa (< manje, > veće, = jednako). U zavisnosti od učitanoog znaka ispituje se jedan od tri odnosa za poredenje ova dva broja. Ako je odnos tačan štampati poruku "Izabrali smo pravi odnos brojeva a i b", a ako nije "Izabrali smo pogresan odnos brojeva a i b". U slučaju da se unese pogrešan znak za poredenje štampati poruku "Nedopusteni znak za odnos brojeva!".

```
#include<stdio.h>
#include <stdio.h>
int main(void)
{
float a,b;
char znak;
printf("Upisati prvi broj: ");
scanf(" %f",&a);
printf("Upisati drugi broj: ");
scanf(" %f",&b);
printf("Upisati znak poredjenja: manje <, vece >,\n");
printf(" jednako = :");
scanf(" %c",&znak);
if(znak=='<')
{if(a<b) printf("Izabrali smo pravi odnos brojeva a i b\n");
else printf("Izabrali smo pogresan odnos brojeva a i b\n");}
else if(znak=='>')
{if(a>b) printf("Izabrali smo pravi odnos brojeva a i b\n");
else printf("Izabrali smo pogresan odnos brojeva a i b\n");}
else if(znak=='=')
{if(a==b) printf("Izabrali smo pravi odnos brojeva a i b\n");
else printf("Izabrali smo pogresan odnos brojeva a i b\n");}
else
{printf("Nedopusteni znak za odnos brojeva!\n");}
return 0;
}
```

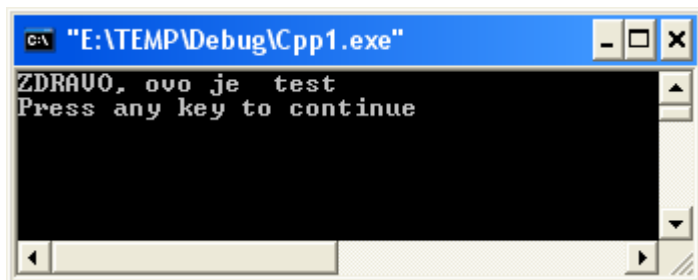
```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Upisati prvi broj: 10
Upisati drugi broj: 20
Upisati znak poredjenja: manje <, vece >,
jednako = :<
Izabrali smo pravi odnos brojeva a i b
Press any key to continue
```

```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Upisati prvi broj: 10
Upisati drugi broj: 20
Upisati znak poredjenja: manje <, vece >,
jednako = :+
Nedopusteni znak za odnos brojeva!
Press any key to continue
```

# FUNKCIJE

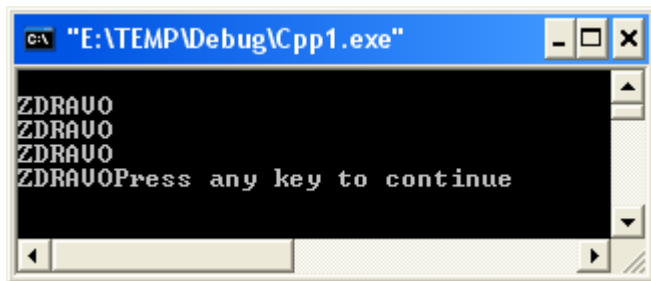
## 1. Pokazati kako funkcioniše prototip funkcije.

```
#include<stdio.h>
myfunc();/*inicijalizacija funkcije bez argumenata*/
void main()
{
    myfunc();/*pozivanje funkcije bez argumenata*/
}
myfunc();/*definisanje funkcije bez argumenata*/
{
    printf("ZDRAVO, ovo je test\n");
return 0;}
```



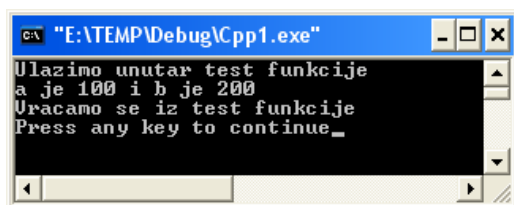
## 2. Korišćenje prototipa funkcije sa argumentom.

```
#include<stdio.h>
napisi(int count);
void main()
{napisi(4);}
napisi(int count)
{
    int c;
    for(c=0;c<count;c++)
        printf("\nZDRAVO");
return 0;}
```



**3. Test funkcije**

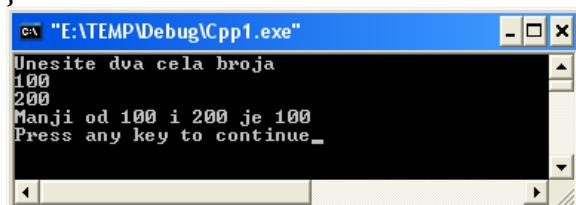
```
#include<stdio.h>
test();
void main()
{
printf("Ulazimo unutar test funkcije\n");
test();
printf("\nVracamo se iz test funkcije\n");
}
test()
{
int a,b;a=100;
b=a+100;
printf("a je %d i b je %d",a,b);
return 0;}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Ulazimo unutar test funkcije
a je 100 i b je 200
Vracamo se iz test funkcije
Press any key to continue_
```

**4. Naći minimum od dva cela broja. U glavnom programu obezbediti štampanje, a u funkciji njihovo poređenje.**

```
#include<stdio.h>
int imin(int n,int m);
void main()/* glavni program */
{
int broj1,broj2,vrati;
printf("Unesite dva cela broja\n");
vrati=scanf("%d %d",&broj1,&broj2);
if(vrati==2)
printf("Manji od %d i %d je %d\n",broj1,broj2,imin(broj1,broj2));
int imin(int n,int m)/* potprogram */
{
int min;
if (n<m)
min=n;
else
min=m;
return min;
}
```



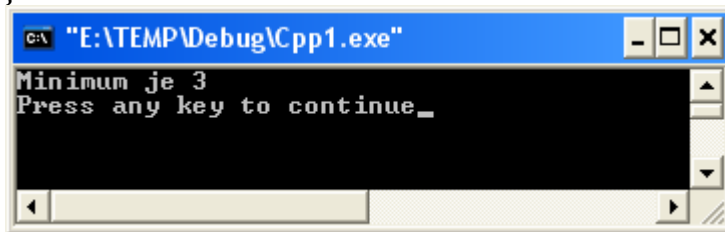
```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite dva cela broja
100
200
Manji od 100 i 200 je 100
Press any key to continue_
```



**5. Isti kao 4. samo na drugi način.**

```
#include<stdio.h>
int min(int a, int b);
main()
{
int m;
m=min(3,6);
printf("Minimum je %d\n",m);
return 0;
}
```

```
int min(int a, int b)
{
if(a<b)
return a;
else
return b;
}
```

**6. Formiranje zbira preko funkcije**

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

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

```
int Add (int x, int y)
{
cout << "U Add(), preuzimam " << x << " i " << y << "\n";
return (x+y); }
```

```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Ja sam u main()!
Unesite dva broja: 100
200

Pozivam Add()
U Add(), preuzimam 100 i 200

Nazad u main().
c ima vrednost 300
Izlazim...

Press any key to continue

```

**7. Uraditi meni sa izborom operacija za sabiranje, oduzimanje i množenje dva cela realna broja. U funkciji izvesti opisane računске radnje.**

```

#include<stdio.h>
#include<stdlib.h>
add();
subtract();
multiply();
void main()
{
int choice;
while(1)
{
printf("\n\nMenu:\n");
printf("1- Add\n2- Subtract\n");
printf("3- Multiply\n4- Exit");
printf("\n\nYour choice -> ");
scanf("%d",&choice);
switch(choice)
{ case 1 : add();
break;
case 2 : subtract();
break;
case 3 : multiply();
break;
case 4 : printf("\nProgram Ends. !");
exit(0);
default:
printf("\nInvalid choice");
}
}
}

add()
{
float a,b;
printf("\nEnter a:");
scanf("%f",&a);
printf("\nEnter b:");

```

```
scanf("%f",&b);
printf("a+b=%f",a+b);
return 0;}
```

```
subtract()
{
float a,b;
printf("\nEnter a:");
scanf("%f",&a);
printf("\nEnter b:");
scanf("%f",&b);
printf("a-b=%f",a-b);
return 0;}
```

```
multiply()
{
float a,b;
printf("\nEnter a:");
scanf("%f",&a);
printf("\nEnter b:");
scanf("%f",&b);
printf("a*b=%f",a*b);
return 0;}
```

```

c:\ "E:\TEMP\Debug\Cpp1.e... - □ x
Menu:
1- Add
2- Subtract
3- Multiply
4- Exit

Your choice -> 2

Enter a:100

Enter b:200
a-b=-100.000000

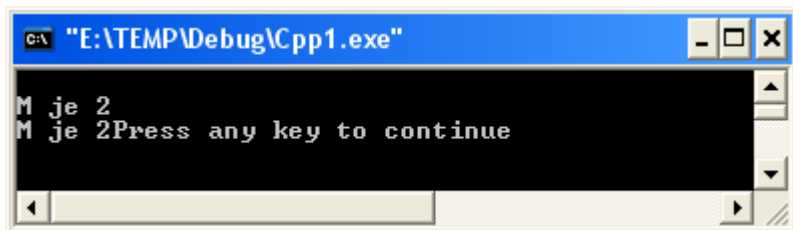
Menu:
1- Add
2- Subtract
3- Multiply
4- Exit

Your choice ->
```

## 8. Pozivanje funkcije po vrednosti

```
/*Pozivanje po vrednosti*/
#include<stdio.h>
void test(int a);
main()
{
int m;
m=2;
printf("\nM je %d",m);
test(m);
printf("\nM je %d",m);
return 0;
}
```

```
void test(int a)
{
a=5;}
```

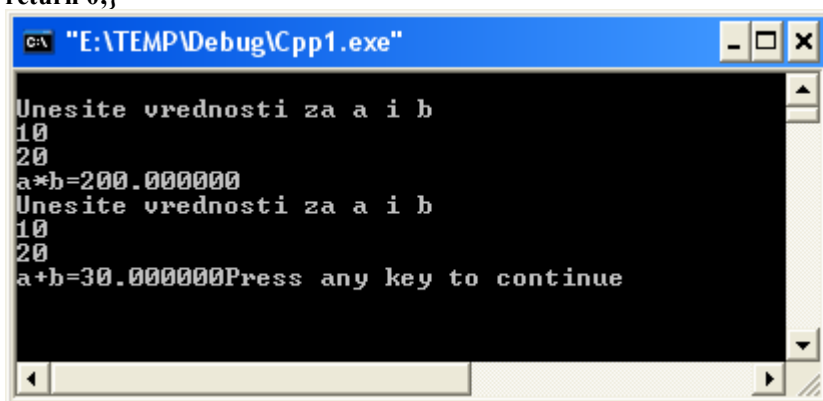


## 9.MNOZENJE I SABIRANJE DVA REALNA BROJA PREKO FUNKCIJE

```
#include<stdio.h>
#include<math.h>
mno();
sab();
void main()
{ mno();
sab();}

mno()
{float a,b;
printf("\nUnesite vrednosti za a i b\n");
scanf("%f%f",&a,&b);
printf("a*b=%f",a*b);
return 0;}

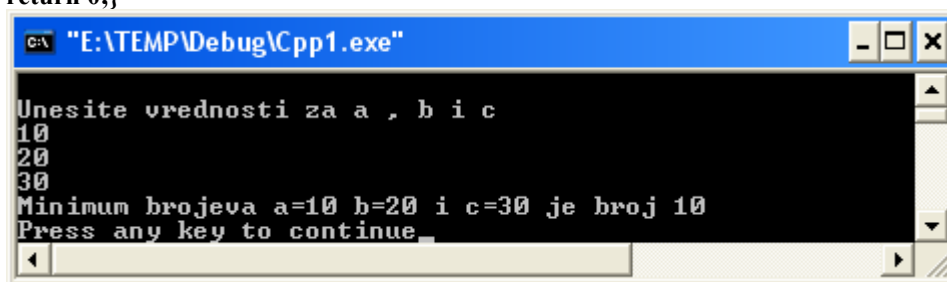
sab()
{float a,b;
printf("\nUnesite vrednosti za a i b\n");
scanf("%f%f",&a,&b);
printf("a+b=%f",a+b);
return 0;}
```



**10. MINIMUM TRI CELA BROJA PREKO FUNKCIJE**

```
#include<stdio.h>
#include<math.h>
min();
void main()
{ min();}

min()
{int a,b,c,m;
printf("\nUnesite vrednosti za a , b i c\n");
scanf("%d%d%d",&a,&b,&c);
m=a;
if(b<m) m=b;
if(c<m) m=c;
printf("Minimum brojeva a=%d b=%d i c=%d je broj %d\n",a,b,c,m);
return 0;}
```

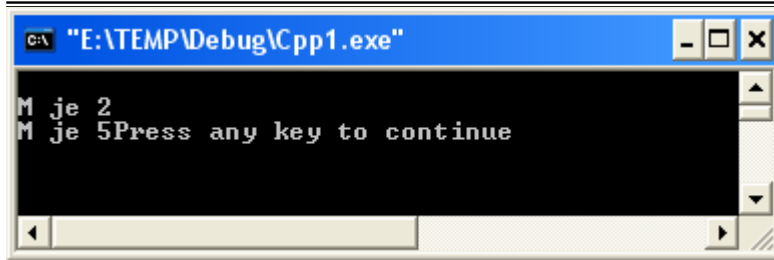


```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite vrednosti za a , b i c
10
20
30
Minimum brojeva a=10 b=20 i c=30 je broj 10
Press any key to continue_
```

**11. Poziv funkcije salje vrednost promenljive m funkciji a ne salje promenljivu samu sebi: m=2, M=2**

```
/*Pozivanje po referenci*/
#include<stdio.h>
void test(int *ptr);
main()
{
int m;
m=2;
printf("\nM je %d",m);
test(&m);
printf("\nM je %d",m);
return 0;
}

void test(int *ptr)
{
*ptr=5;
}
/*Poziv funkcije salje vrednost promenljive m po referenci: m=2, M=5*/
```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
M je 2
M je 5
Press any key to continue

```

## 12. Suma kvadrata celih brojeva od 1 do N preko funkcije

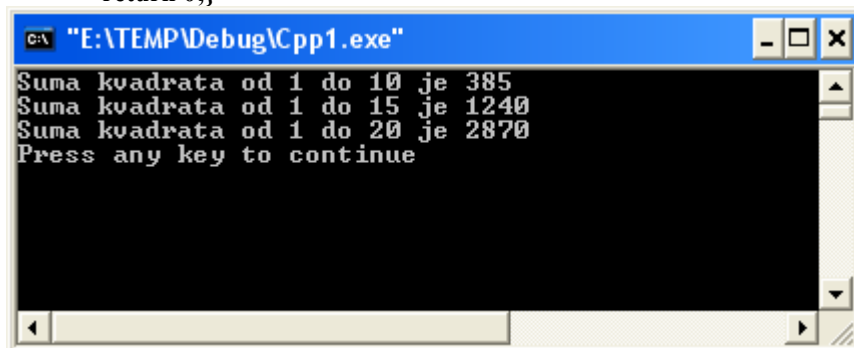
```

/*Program za izracunavanje sume
kvadrata celih brojeva od 1 do n*/
#include <stdio.h>
#include <math.h>
suma_kvadrata(int n);

void main()
{
    suma_kvadrata(10);
    suma_kvadrata(15);
    suma_kvadrata(20);
}

suma_kvadrata(int n) /*f ja za izracunavanje*/
                    /*sume kvadrata*/
{
    int i;
    long suma=0;
    for (i=1; i<=n; suma+=(long)i*i, ++i);
    printf("Suma kvadrata od 1 do %d je %ld\n", n, suma);
    return 0;}

```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Suma kvadrata od 1 do 10 je 385
Suma kvadrata od 1 do 15 je 1240
Suma kvadrata od 1 do 20 je 2870
Press any key to continue

```

## 13. Napisati program za rešavanje kvadratne j-ne $ax^2+bx+c=0$ .

```

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

```

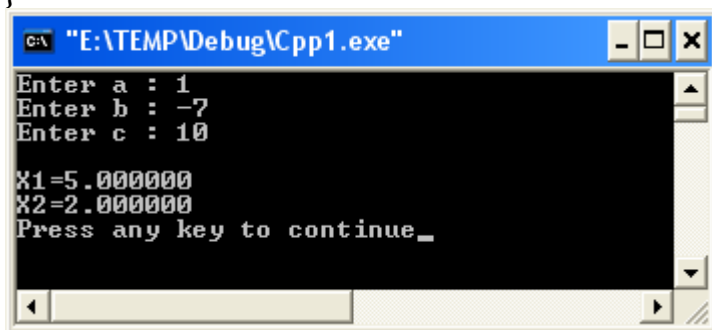
```

void main()
{
float delta,a,b,c,x1,x2;

printf("Enter a : ");
scanf("%f",&a);
printf("Enter b : ");
scanf("%f",&b);
printf("Enter c : ");
scanf("%f",&c);
delta=b*b-(4*a*c);
if(delta<0)
{
printf("Jednacina NEMA resenja !\n");
exit(0); //Potrebna hederska funkcija stdlib
}
if(delta==0)
{
x1=-b/(2*a);
printf("Jednacina IMA dva ista resenja !\n");
printf("x1=x2=%f",x1);
exit(0);
}

x1=(-b+sqrt(delta))/(2*a);
x2=(-b-sqrt(delta))/(2*a);
printf("\nX1=%f",x1);
printf("\nX2=%f\n",x2);
}

```



```

C:\ "E:\TEMP\Debug\Cpp1.exe"
Enter a : 1
Enter b : -7
Enter c : 10

X1=5.000000
X2=2.000000
Press any key to continue_

```

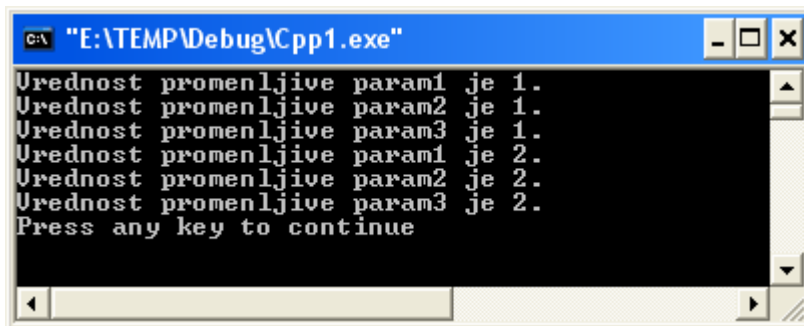
#### 14. F-ja ne vraca nikakvu vrednost

```

#include<stdio.h>
void ispis(int i1, int i2, int i3); /* F-ja ne vraca nikakvu vrednost*/
main()
{
int a,b,c;
a=b=c=1;
ispis(a,b,c);
a=b=c=2;
ispis(a,b,c);
return 0;}

```

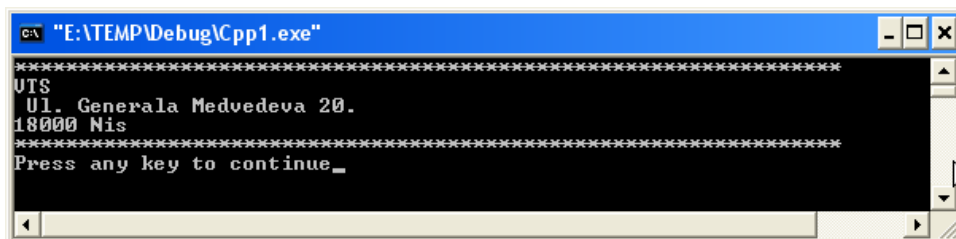
```
void ispis(int i1, int i2, int i3)
{
printf("Vrednost promenljive param1 je %d.\n",i1);
printf("Vrednost promenljive param2 je %d.\n",i2);
printf("Vrednost promenljive param3 je %d.\n",i3);
}
```



```
"E:\TEMP\Debug\Cpp1.exe"
Vrednost promenljive param1 je 1.
Vrednost promenljive param2 je 1.
Vrednost promenljive param3 je 1.
Vrednost promenljive param1 je 2.
Vrednost promenljive param2 je 2.
Vrednost promenljive param3 je 2.
Press any key to continue
```

### 15. Poziv funkcije bez argumenta

```
#include <stdio.h>
#define IME "VTS"
#define ADRESA "Ul. Generala Medvedeva 20."
#define MESTO "18000 Nis"
#define LIMIT 65
void zvezde();
void main()
{ void zvezde(); /*deklaracija funkcije bez argumenata*/
zvezde(); /*poziv korisnicke funkcije */
printf("%s\n", IME); /*poziv funkcije iz standardne biblioteke*/
printf("%s\n", ADRESA);
printf("%s\n", MESTO);
zvezde (); }
/*definicija korisnicke funkcije */
void zvezde() /*funkcija nema argumenata*/
{ int brojac;
for(brojac=1;brojac<=LIMIT;brojac++)
putchar ( '*');
putchar ( '\n'); }
```



```
"E:\TEMP\Debug\Cpp1.exe"
*****
VTS
Ul. Generala Medvedeva 20.
18000 Nis
*****
Press any key to continue_
```

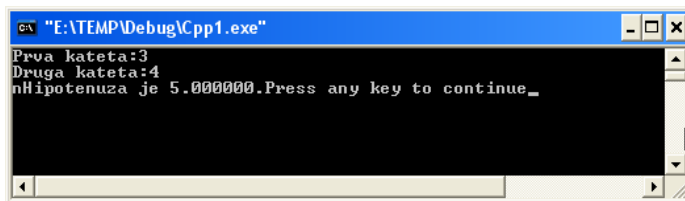


**16. Primer kada program poziva dve funkcije**

```
#include<stdio.h>
#include<math.h>
void Hipotenuza(void);// funkcijski prototipovi
void Ucitaj(void); //funkcijski prototipovi
double a,b,c;/* deklaracija globalne promenljive, pre pocetka programa*/
void main()
{ Ucitaj();
  Hipotenuza();
  printf("\nHipotenuza je %f.",c);}

void Hipotenuza(void)
{c=sqrt(a*a+b*b);}

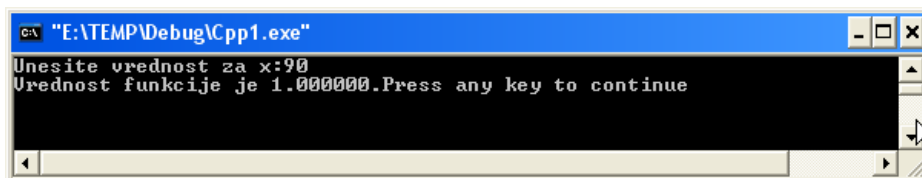
void Ucitaj(void)
{printf("Prva kateta:");
scanf("%lf",&a);
printf("Druga kateta:");
scanf("%lf",&b);}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Prva kateta:3
Druga kateta:4
nHipotenuza je 5.000000.Press any key to continue_
```

**17. Primer za izracunavanje  $y=\sin^2(x)+\cos^2(x)$** 

```
#include<stdio.h>
#include<math.h>
double Funkcija(double x);
void main()
{double x,y;/* deklaracija lokalnih promenljivih, posle pocetka programa*/
printf("Unesite vrednost za x:");
scanf("%lf",&x);
y=Funkcija(x);
printf("Vrednost funkcije je %f.",y);}
double Funkcija(double x)
{double y;
y=sin(x)*sin(x)+cos(x)*cos(x);
return y;}
```



```
C:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite vrednost za x:90
Vrednost funkcije je 1.000000.Press any key to continue
```

**18. Izracunavanje površine trougla sa dve funkcije**

```
#include<stdio.h>
#include<math.h>
void Povrsina(void);
void Ucitaj(void);
double a,b,c,p,pt;/* deklaracija globalne promenljive, pre pocetka programa*/
void main()
{Ucitaj();
Povrsina();
printf("/nPovrsina trougla je %f.",pt);}
void Povrsina(void)
{p=(a+b+c)/2;
pt=sqrt(p*(p-a)*(p-b)*(p-c));}
void Ucitaj(void)
{printf("Prva kateta:");
scanf("%lf",&a);
printf("Druga kateta:");
scanf("%lf",&b);
printf("Trecia kateta:");
scanf("%lf",&c);}
```

**19. Neka je data kvadratna matrica u formatu:**

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Uneti članove matrice u program kao celobrojne veličine, a kao izlaz iz programa predvideti štampanje takve matrice u pokazanom formatu kao i štampanje izračunate vrednosti njene dijagonale  $a_{11} * a_{22} * a_{33} * a_{44}$  urađene preko funkcije.

```
#include<stdio.h>
#include<math.h>
#include<iomanip.h>
main()
{static int mat[4][4];
int i,j;
float dijagonala;
printf("Unesite proizvoljne elemente pravougle matrice:\n");
for(i=0;i<4;++i)
for(j=0;j<4;++j)
{printf("a[%d][%d]=",i,j);
scanf("%d",&mat[i][j]);printf("\n");}
printf("\nPolazna matrica je\n");}
```

```

for(i=0;i<4;++i)
{for(j=0;j<4;++j)
printf("%3d ",mat[i][j]);printf("\n");}
dijagonala=mat[0][0]*mat[1][1]*mat[2][2]*mat[3][3];
printf("Vrednost dijagonale je :%f", dijagonala);
return 0;}

```

```

E:\TEMP\Debug\Cpp1.exe
a[1][3]=8
a[2][0]=9
a[2][1]=10
a[2][2]=11
a[2][3]=12
a[3][0]=13
a[3][1]=14
a[3][2]=15
a[3][3]=16

Polazna matrica je
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 16

Vrednost dijagonale je :1056.000000Press any key to continue.

```

## 20. Četiri računске radnje preko funkcije choice i switch naredbi.

```

#include<stdio.h>
#include<stdlib.h>
void add(void);
void subtract(void);
void multiply(void);
void divide(void);
float a,b,rezultat;
void main()
{int choice;

while(1)
{
    printf("\n\nMenu:\n");
    printf("1- Add\n2- Subtract\n");
    printf("3- Multiply\n4- Divide\n5-Exit");
    printf("\n\nYour choice -> ");
    scanf("%d",&choice);
    switch(choice)
    { case 1 : add();printf("Rezultat operacije je:%f\n",rezultat);
      break;
      case 2 : subtract();printf("Rezultat operacije je:%f\n",rezultat);
      break;
      case 3 : multiply();printf("Rezultat operacije je:%f\n",rezultat);
      break;
      case 4 : divide();printf("Rezultat operacije je:%f\n",rezultat);
      break;
      case 5 : printf("\nProgram Ends. !");
      exit(0);
      default:
      printf("\nInvalid choice");
    }
}
}

```

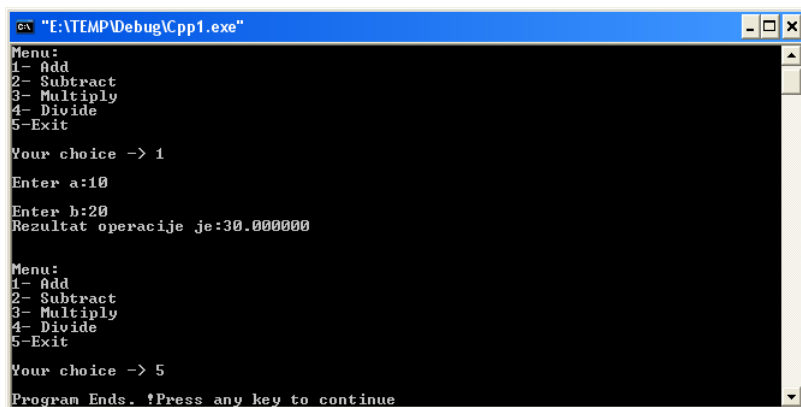
```

void add(void)
{printf("\nEnter a:");
scanf("%f",&a);
printf("\nEnter b:");
scanf("%f",&b);
rezultat=(a+b);}
void subtract(void)
{printf("\nEnter a:");
scanf("%f",&a);
printf("\nEnter b:");
scanf("%f",&b);
rezultat=(a-b);}

void multiply(void)
{printf("\nEnter a:");
scanf("%f",&a);
printf("\nEnter b:");
scanf("%f",&b);
rezultat=(a*b);}

void divide(void)
{printf("\nEnter a:");
scanf("%f",&a);
printf("\nEnter b:");
scanf("%f",&b);
rezultat=(a/b);}

```



```

C:\TEMP\Debug\Cpp1.exe
Menu:
1- Add
2- Subtract
3- Multiply
4- Divide
5-Exit
Your choice -> 1
Enter a:10
Enter b:20
Rezultat operacije je:30.000000

Menu:
1- Add
2- Subtract
3- Multiply
4- Divide
5-Exit
Your choice -> 5
Program Ends. !Press any key to continue

```

21. Treba napisati program koja uzima dva trodimenzionalna vektora i ispituje njihovu NORMALNOST. Matematički pojmovi vektora i matrica implementiraju se u C-u pomoću polja. Polje je niz indeksiranih promenljivih istog tipa, smešenih u memoriji na uzastopnim lokacijama. Indeks polja uvek kreće od nule.

Da bismo ispitili normalnost vektora koristićemo formulu za kosinus ugla između dva vektora:

$$\cos \phi = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|},$$

gde je  $\vec{a} \cdot \vec{b} = a_1b_1 + a_2b_2 + a_3b_3$  skalarni produkt vektora, a

$$\|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

norma vektora. Vektori su normalni ukoliko je kosinus ugla među njima jednak nuli. Budući da se u računu kosinusa pojavljuju greške zaokruživanja, uzimaćemo da su vektori normalni čim je kosinus njihovog kuta manji od nekog malog broja epsilon, u našem primeru  $10^{-10}$ .

```
#include <stdio.h>
#include <math.h>
double epsilon=1.0E-10;
double kosinus_ugla(double x[], double y[]);
int main(void)
{
    double a[3], b[3];
    double cos_phi;
    int i;
    printf("Unesite vektor a.\n");
    for(i=0;i<3;++i){
        printf("a[%d]= ",i+1);
        scanf(" %lf",&a[i]);
    }
    printf("Unesite vektor b.\n");
    for(i=0;i<3;++i){
        printf("b[%d]= ",i+1);
        scanf(" %lf",&b[i]);
    }
    cos_phi= kosinus_ugla(a, b);
    if(fabs(cos_phi) < epsilon){
        printf("Vektori su normalni.\n");
        printf("Kosinus ugla = %f\n", cos_phi);
    }
    else{
        printf("Vektori nisu normalni.\n");
        printf("Kosinus ugla = %f\n", cos_phi);
    }
    return 0;
}
double norma(double x[]) {
    int i;
    double suma;
    suma=0.0;
    for(i=0;i<3;++i) suma = suma + x[i]*x[i];
    return sqrt(suma);
}
double produkt(double x[], double y[]) {
    int i;
    double suma;
    suma=0.0;
    for(i=0;i<3;++i) suma = suma + x[i]*y[i];
    return suma;
}
double kosinus_ugla(double x[], double y[]) {
    double cos_phi;
    cos_phi=produkt(x,y);
    cos_phi=cos_phi/(norma(x)*norma(y));
    return cos_phi; }
```

```

c:\ "E:\TEMP\Debug\Cpp1.exe"
Unesite vektor a.
a[1]= 1
a[2]= 2
a[3]= 3
Unesite vektor b.
b[1]= -1
b[2]= -2
b[3]= -3
Vektori nisu normalni.
Kosinus ugla = -1.000000
Press any key to continue_

```

### RAD SA MAKROIMA

Na sledećem primeru sagledati upotrebu makroa:

```
#include<stdio.h>
```

```

#define STRING1                "Definicija makroa\n"
#define STRING2                "Sve mora biti u novoj liniji\n"
#define IZRAZ1                  1 + 2 + 3 + 4
#define IZRAZ2                  IZRAZ1 + 10
#define ABS(x)                  ((x)<0 ? -(x):(x))
#define MAX(a,b)                (a<b ? (b):(a))
#define BIGGEST(a,b,c) (MAX(a,b)<c ? (c):MAX(a,b))

```

```
int main()
```

```

{
    printf(STRING1);
    printf(STRING2);
    printf("%d\n", IZRAZ1);
    printf("%d\n", IZRAZ2);
    printf("%d\n", ABS(-5));
    printf("Najveci broj izmedju brojeva 1 2 i 3 je:%d\n", BIGGEST(1,2,3));
    return 0;
}

```

```

c:\ "C:\temp\Debug\Cpp1.exe"
Definicija makroa
Sve mora biti u novoj liniji
10
20
5
Najveci broj izmedju brojeva 1 2 i 3 je:3
Press any key to continue_

```

**LITERATURA:**

1. " The C Programming Language ", Bjarne Stroustrup, Addison-Wesley, 1986 (first edition).
2. " Strukturne metode razvoja softvera - Programski jezik C ", Saša Bošnjak, Grafoprodukt Subotica, 1998. god.
3. " C jezik ", Vladan Vujičić, RO Institut za nuklearne nauke Boris Kidrič - Vinča, 1988. god.
4. " Programming in C ", learnem.com, 2004. god.
5. " Programming in C: A Tutorial ", Brian Kernighan, Bell Laboratories, Murray Hill, N.J. 2004.god.
6. " Programski jezici ", Milena Stanković, Elektronski fakultet Niš, 2000. god.
7. " Algorithms in C - Third Edition, Parts 1-2 ", Robert Sadgewick, Pearson Education Inc. 1998. god.
8. " Algorithms in C - Third Edition, Parts 3-4 ", Robert Sadgewick, Pearson Education Inc. 1998. god.
9. " The Annotated C Reference Manual ", Bjarne Stroustrup and Margaret Ellis, Addison-Wesley, 1990, pp. 20-21.

---

---

**SADRŽAJ:**

<b>1. UVOD</b>	<b>1</b>
<b>1.1 UVOD U PROGRAMSKE JEZIKE</b>	<b>1</b>
1.2. Definicija programskih jezika	2
1.3. Klasifikacija programskih jezika po stepenu zavisnosti od računara	4
1.4. Hronologija razvoja viših programskih jezika	7
1.5. Karakteristike programskih jezika	14
1.6. Mašinski jezici	17
1.7. Simbolički jezici	18
1.8. Viši programski jezici	19
1.9. Podela programskih jezika prema oblasti primene	21
1.9.1. Jezici za naučne aplikacije:	21
1.9.2. Jezici za poslovne aplikacije	22
1.9.3. Jezici veštačke inteligencije	22
1.9.4. Jezici za razvoj sistemskog softvera	22
1.9.5. Jezici za računarske komunikacije	23
1.9.6. Jezici specijalne namene	23
1.10. Kriterijumi ocene jezika	23
1.10.1. Čitljivost	23
1.10.2. Pouzdanost jezika	25
1.10.3. Efikasnost jezika	26
1.11. Karakteristike programskih jezika	27
<b>2. ELEMENTI PROGRAMSKIH JEZIKA</b>	<b>28</b>
2.1. Uvod u elemente programskih jezika:	28
2.1.1. Azbuka jezika	29
2.1.2. Rezervisane reči	30
2.1.3. Konstante	31
2.1.4. Promenljive	32
2.1.5. Komentari	34
2.2. TIPOVI PODATAKA	34
2.2.1. Koncept slabih tipova	35
2.2.2. Koncept jakih tipova podataka	36



---

2.2.3. Ekvivalentnost tipova	37
2.2.4. Elementarni tipovi podataka	38
<b>3. PROGRAMIRANJE</b>	<b>40</b>
<b>3.1. PROCES PROGRAMIRANJA</b>	<b>40</b>
3.1.1. Greške u programiranju.	40
3.1.2. Brzina izvršavanja programa	40
3.1.3. Ekonomično korišćenje memorijskog prostora	41
3.1.4. Brzina programiranja	41
3.1.5. Sastavljanje algoritama i programa	41
3.1.6. Sistemski dijagrami	41
3.1.7. Organizacija programa	42
3.1.8. Modularnost programa	44
3.1.9. Cikličnost	45
3.1.10. Indeksiranje	46
3.1.11. Testiranje programa (debugging)	46
3.1.12. Dokumentacija	48
<b>3.2. ALGORITMI I OPERACIJE</b>	<b>49</b>
3.2.1. Algoritmi	49
3.2.2. Osobine algoritma	49
3.2.3. Formiranje i zapis algoritma	49
3.2.4. Metode linearnog i nelinearnog programiranja.	50
3.2.5. Algoritamske strukture	51
<b>4. OSNOVE C JEZIKA</b>	<b>53</b>
<b>4.1 Skup karaktera C jezika</b>	<b>56</b>
<b>4.2 Ključne reči</b>	<b>56</b>
<b>4.3 Osnovni elementi C jezika</b>	<b>57</b>
<b>4.4 Elementarni ulaz i izlaz i predprocesori</b>	<b>61</b>
4.4.1. Preprocesori	63
<b>4.5. OSNOVNI TIPOVI PODATAKA I ARITMETIČKI IZRAZI</b>	<b>66</b>
4.5.1. Osnovni tipovi podataka	66
4.5.2. Tip int	66
4.5.3. Tip float i double	68
4.5.4. Tip char	70
<b>4.6. Aritmetički izrazi i operatori</b>	<b>72</b>
4.6.1. Operatori dodeljivanja vrednosti	74
4.6.2. Konverzija tipova podataka	76

---

<b>4.7. KONTROLA TOKA PROGRAMA</b>	<b>76</b>
4.7.1. Relacioni operatori i izrazi	77
4.7.2. Logički operatori i izrazi	77
4.7.3. Operator inkrementiranja i dekrementiranja	79
4.7.4. Složeni i bezefektni iskazi	81
4.7.5. While iskaz	81
4.7.6. For iskaz	84
4.7.7. Do-while iskaz	85
4.7.8. If iskaz	86
4.7.9. If-else iskaz	88
4.7.10. Else-if iskaz	89
4.7.11. switch iskaz	90
4.7.12. Iskazi bezuslovnog grananja	92
<b>4.8. Operator uslovnog izraza</b>	<b>94</b>
<b>4.9. Vektori</b>	<b>95</b>
4.9.1. Jednodimenzionalni vektori	95
4.9.2. Višedimenzionalni vektori	98
<b>4.10. Strukture, unije i enumerisani tipovi</b>	<b>101</b>
4.10.1. Strukture	101
4.10.2. Strukture i vektori	103
4.10.3. Unija	104
4.10.4. Enumerisani tip podataka	107
4.10.5. Typedef iskaz	108
<b>4.11. Funkcije, domeni i memorijske klase</b>	<b>110</b>
4.11.1. Definicija funkcije	110
4.11.2. Argumenti funkcija	111
4.11.3. Rezultati funkcija	113
4.11.4. Vektori kao argumenti funkcija	115
4.11.5. Strukture kao argumenti funkcija	117
4.11.6. Rekurzivne funkcije	119
4.11.7. Domeni identifikatora	121
4.11.8. Memorijske klase identifikatora	122
<b>4.12. Ukazatelji</b>	<b>124</b>
4.12.1. Pojam ukazatelja	125
4.12.2. Ukazatelji i strukture	127
4.12.3. Dinamičke strukture podataka	129
4.12.4. Ukazatelji i vektori	137
<b>4.13. Nizovi karaktera</b>	<b>140</b>
4.13.1. Pojam nizova karaktera	140
<b>4.14. Operacije na bitovima</b>	<b>144</b>
4.14.1. Pojam bita	144

---

4.14.2. Operacije na bitovima	145
4.14.3. Bitsko polje	149
<b>4.15. Ulaz i izlaz</b>	<b>151</b>
4.15.1. Ulaz/izlaz karaktera- getch() i putch()	151
4.15.2. Formatizovani ulaz/izlaz	151
4.15.3. Upravljanje datotekama	156
<b>PRAKTIKUM ZA LABORATORIJSKE VEŽBE</b>	<b>169</b>
<b>ZBIRKA REŠENIH ZADATAKA:</b>	<b>209</b>
<b>LITERATURA</b>	<b>283</b>
<b>SADRŽAJ</b>	<b>284</b>