Osnovi programiranja

Programski jezik C

— Zadaci sa vežbi —

Milena Vujošević - Janičić 2005/2006

Sadržaj

Predgovor

Ovo je prateći materijal za vežbe koje držim iz predmenta Osnovi programiranja. On ne može zameniti pohađanje vežbi niti korišćenje druge preporučene literature.

Veliki deo materijala čine zadaci i rešenja mr Filipa Marića (raspoloživi na www.matf.bg.ac.yu/~filip/pp/0405/index.pl). Takođe korišćen je i materijal sa sajta koleginice Jelene Grmuše www.matf.bg.ac.yu/~jelenagr i kolege Miroslava Marića www.matf.bg.ac.yu/~maricm. Tekstovi i objašnjenja su uglavnom zasnovani na knjizi *Programski jezik C*, autora *Kerninghan & Ritchie*

Zahvaljujem svojim studentima na aktivnom učešću u nastavi čime su mi pomogli u uobličavanju ovog materijala.

Svi komentari i sugestije vezane za ovaj materijal biće veoma dobrodošli.

Milena Vujošević-Janičić
www.matf.bg.ac.yu/~milena

Glava 1

HTML

http://www.matf.bg.ac.yu/nastavno/dvitas/nastava/op/html/op-html.html

Glava 2

Konverzije

2.1 Pozicioni brojni sistemi

Brojevni sistemi[1]

naziv	vrednost osnove	cifre
dekadni	10	0,1,2,3,4,5,6,7,8,9
binarni	2	0,1
oktalni	8	0,1,2,3,4,5,6,7
heksadekadni	16	0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F

Zapis u osnovi (bazi) b

$$(c_n c_{n-1} ... c_1 c_0)_b$$

gde su

$$c_n, c_{n-1}, ... c_1, c_0$$

cifre u sistemu sa osnovom b ima vrednost

$$c_0 * b^0 + c_1 * b^1 + ... + c_{n-1} * b^{n-1} + c_n * b^n$$

Primer 2.1.1 *1.* $(1101)_2 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = (13)_{10}$

2. $(1101)_{16} = 1 * 16^0 + 0 * 16^1 + 1 * 16^2 + 1 * 16^3 = 1 + 256 + 4096 = (4353)_{10}$

3. $(1101)_7 = 1 * 7^0 + 0 * 7^1 + 1 * 7^2 + 1 * 7^3 = (399)_{10}$

4. $(F9A)_{16} = A * 16^0 + 9 * 16^1 + F * 16^2 == = 10 * 16^0 + 9 * 16^1 + 15 * 16^2 = 10 + 144 + 3840 = (3994)_{10}$

Zadatak 1

$$(110111100)_2 = (?)_{10}$$

$$(77)_8 = (?)_{10}$$

$$(FFFF)_{16} = (?)_{10}$$

[1] Deo teksta i primera preuzet sa sajta www.matf.bg.ac.yu/~jelenagr

2.2 Prevođenje iz dekadnog sistema u binarni

Primer 2.2.1

$$(26)_{10} = (?)_2$$

kolicnik ostatak
26 / 2 = 13 0
13 / 2 = 6 1
6 / 2 = 3 0
3 / 2 = 1 1
1 / 2 = 0 1 kraj postupka konverzije

$$(26)_{10} = (11010)_2$$

Zadatak 2

$$(54)_{10} = (?)_2$$
$$(126)_{10} = (?)_2$$
$$(332)_{10} = (?)_2$$

2.3 Prevođenje iz dekadnog sistema u oktalni

Primer 2.3.1

$$(181)_{10} = (?)_8$$

kolicnik ostatak
181 / 8 = 22 5
22 / 8 = 2 6
2 / 8 = 0 2 kraj postupka konverzije

$$(181)_{10} = (265)_8$$

Zadatak 3

$$(67)_{10} = (?)_8$$
$$(336)_{10} = (?)_8$$
$$(442)_{10} = (?)_8$$

2.4 Prevođenje iz dekadnog sistema u heksadekadni

Primer 2.4.1

$$(181)_{10} = (?)_{16}$$

kolicnik ostatak
181 / 16 = 11 5
11 / 16 = 0 11 (heksadekadna cifra B) kraj postupka konverzije

$$(181)_{10} = (B5)_{16}$$

Zadatak 4

$$(48)_{10} = (?)_{16}$$
$$(1336)_{10} = (?)_{16}$$
$$(332)_{10} = (?)_{16}$$

2.5 Prevođenje iz dekadnog sistema u sistem sa osnovom n

Primer 2.5.1

$$(181)_{10} = (?)_n$$

kolicnik ostatak
181 / n = k_1 o_1
k_1 */ n =* k_2 o_2
...
k_l */ n = 0* o_{l+1} *kraj postupka konverzije*

$$(181)_{10} = (o_{l+1}o_l...o_1)_n$$

Zadatak 5

$$(48)_{10} = (?)_{16}$$
$$(1336)_{10} = (?)_{16}$$
$$(332)_{10} = (?)_{16}$$

2.6 Prevođenje iz binarnog sistema u heksadekadni

Prevođenje iz binarnog sistema u heksadekadni može se uraditi bez međukonverzije u osnovu 10. Za kodiranje heksadecimalnih cifara dovoljne su binarne reči dužine cetiri.

heksadekadna cifra	kod
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

primer:
4 0100 $(0 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3)$
F 1111 $(1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3)$

Odavde je jasno kako zadovoljiti zahtev da se konverzija odvija direktno bez posrednih konverzija. Binarne cifre se grupišu podgrupe od po 4 cifre, pocev

od bitova najmanje težine. Ako ukupan broj bitova nije deljiv sa cetiri, onda se dopisuje potreban broj vodecih nula (one su bez uticaja na promenu vrednosti originalnog zapisa).

Primer 2.6.1

$$(111101110000110101000 0)_2 =$$
$$= (0011110111000011010100 00)_2 =$$
$$= (3DC350)_{16}$$

Zadatak 6

$$(11111101110011000110101001 00)_2 = (?)_{16}$$

2.7 Prevođenje iz binarnog sistema u oktalni

Prevođenje iz binarnog sistema u oktalni može se uraditi bez međukonverzije u osnovu 10. Za kodiranje oktalnih cifara dovoljne su binarne reci dužine tri.

oktalna cifra	kod
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

4 100 $(0 * 2^0 + 0 * 2^1 + 1 * 2^2)$

Odavde je jasno kako zadovoljiti zahtev da se konverzija odvija direktno bez posrednih konverzija. Binarne cifre se grupišu podgrupe od po 3 cifre, pocev od bitova najmanje težine. Ako ukupan broj bitova nije deljiv sa tri, onda se dopisuje potreban broj vodecih nula.

Primer 2.7.1

$$(11111010001010)_2 =$$
$$= (011111010001010)_2 =$$
$$= (37212)_8$$

Zadatak 7

$$(1111110111010100100)_2 = (?)_8$$

Glava 3

Algoritmi

3.1 Rešavanje problema uz pomoć računara

Rešavanje problema uz pomoć računara može se podeliti na sledeće korake:

1. Razumevanje problema
2. Izgradnja modela
3. Formiranje algoritma
4. Provera ispravnosti algoritma
5. Realizacija (implementacija) algoritma — pisanje programa
6. Testiranje programa
7. Sastavljanje dokumentacije.

3.2 Formiranje algoritma

Formirati algoritam znači dati skup preciznih uputstava kako doći do rešenja zadatog problema.

Algoritmi se mogu opisivati: prirodnim jezikom, pseudo-jezikom, blok-shemom (dijagramom toka).

- OPIS ALGORITAMA PRIRODNIM JEZIKOM
  Opisati prirodnim jezikom detaljno, precizno i nedvosmisleno korake pri resavanju problema, vodeći računa o redosledu operacija koji se izvršavaju.
- OPIS ALGORITAMA PSEUDO JEZIKOM
  Pseudo jezik je neformalna kombinacija prirodnog jezika i nekog programskoj jezika. Pri upotrebi pseudo jezika mora se voditi računa da se jezičke konstrukcije koriste uvek na isti način i da budu praćene objašnjenjima (ako je potebno).
- OPISIVANJE ALGORITAMA BLOK SEMOM
  Za ovaj oblik opisa koriste se grafički simboli ciji je opis propisan ISO standardom. Tekst koji opisuje obradu se zapisuje unutar grafičkih simbola.

Tok rada algoritma se opisuju linijama koje povezuju grafičke simbole koji reprezenuju obradu.

3.3 Algebarski algoritmi

Primer 3.3.1 *Algoritam zdravo svete*

```
Algoritam Ispis
ulaz: nema podataka na ulazu
izlaz: poruka Zdravo, svete!!!
{
output " Zdravo, svete!!!"
}
```

Primer 3.3.2 *Algoritam za sabiranje dva broja.*

```
Algoritam Sabiranje
ulaz: x,y
izlaz: zbir brojeva x,y
/* algoritam je formiran tako da funkcionise
za bilo koji realan broj sa ulaza. Zbog toga se
u algoritmu koristi promenljiva - ime koje
oznacava promenljivu vrednost.*/
{
input x,y;
zbir = x + y;
output zbir;
}
```

Primer 3.3.3 *Algoritam za maksimum dva broja.*

```
Algoritam Maksimum
ulaz: x, y;
izlaz: veci od brojeva x i y
{
input x,y;
if (x>y) max=x
       else max=y;
output max;
}
```

Primer 3.3.4 *Algoritam za izracunavanje sume brojeva koja se unosi pre unosa nule.*

```
Algoritam Zbir
ulaz: brojevi sve dok se ne unese nula
izlaz: zbir brojeva
{
zbir=0;     /*Inicijalizujemo zbir na nulu*/
input x;    /*Unosimo prvi u nizu brojeva*/
while x!=0    /*Proveravani da li je broj
```

```
               x razlicit od nule i ako jeste
                onda ga dodajemo na zbir*/
        {
        zbir=zbir+x;
        input x;    /*unosimo novi ulazni broj*/
        }
output zbir;    /*Na izlaz saljemo izracunati zbir*/
}
```

Zadatak 8 *Algoritam za maksimum brojeva koji se unose, zavrsetak unosa je nula.*

```
Algoritam Maksimum
ulaz: brojevi sve dok se ne unese nula
izlaz: maksimum unetih brojeva
{
input x;
max = x;
while (x!=0)   /*Proveravani da li je broj
                x razlicit od nule*/
        {
        input x;    /*unosimo novi ulazni broj*/
        if (x>max) max=x; /*proveravamo da li je uneti
                        broj veci od tekuceg
                        maksimuma i ako jeste
                        onda tekucem maksimumu
                        dodeljujemo njegovu
                        vrednost*/
        }
output max;    /*Na izlaz saljemo izracunati
                 maximum*/
}
```

Zadatak 9 *Algoritam za n!.*

```
Algoritam n!
ulaz: prirodan broj n;
izlaz: faktorijel prirodnog broja n;
{
input n;
f=1;
while (n>1)
        {
        f=f*n;
        n=n-1;
        }
output f;
}
```

Zadatak 10 *Algoritam za razmenu dva broja.*

```
Algoritam razmena
ulaz: dva broja
izlaz: razmena dva broja
{
input x,y;
pom=x;
x=y;
y=pom;
output x,y;
}
```

Zadatak 11 *Celobrojni kolicnik i ostatak pri deljenju*

$$x = qy + r, 0 \leq r < y, 0 \leq q$$

.

```
Algoritam kolicnik
ulaz: prirodni brojevi x i y;
izlaz: kolicnik i ostatak pri deljenju
{
input x,y;
kolicnik=0;
ostatak=x;
while (ostatak>=y)
        {
        ostatak=ostatak-y;
        kolicnik=kolicnik+1;
        }
output kolicnik, ostatak;
}
```

Zadatak 12 *Algoritam koji resava kvadratnu jednacinu.*

$$a * x^2 + b * x + c$$

```
Algoritam kvadratna jednacina
ulaz: koeficijenti a, b,c
izlaz: resenje kvadratne jednacine
{
input a,b,c;
if (a=0)
    if (b=0)
        if (c=0)
            output "C";
        else output "prazan skup";
    else x=-c/b
        output x;
else d=b*b - 4ac
    if (d>0)
        {
        x1=(-b+sqrt(d))/2a;
```

```
        x2=(-b-sqrt(d))/2a;
        output x1, x2;
        }
    else if (d=0)
            {
            x=-b/2a;
            output x;
            }
            else
            {
            Re=-b/2a;
            Im=sqrt(-d)/2a;
            output Re+i*Im, Re-i*Im;
            }
}
```

Zadatak 13 *Fibonacijevi brojevi.*

```
Algoritam fibonaci
ulaz n;
izlaz n-ti fibonacijev broj;
{
input n;
x0=0;
x1=1;
if n=0 rezultat=x0;
    else {
          while (n>1)
          {
            novi=x0+x1;
            x0=x1;
            x1=novi;
            n=n-1;
          }
    rezultat=x1;
output rezultat;
    }
}
```

3.3.1 NZD

Zadatak 14 *Naći najveći zajednički delitelj za dva broja.*

```
Algoritam NZD1
ulaz prirodni brojevi a, b;
izlaz nzd(a,b);
{
input a,b;
nzd = 1;
br=2;
while (br<=a && br<=b)
```

```
    {
    if (a%br)==0 && (b%br)==0 nzd=br;
    br=br+1;
    }
output nzd;
}
```

Zadatak 15 *Naći najveći zajednički delitelj za dva broja.*

```
Algoritam NZD2
ulaz prirodni brojevi a, b;
izlaz nzd(a,b);
{
input a,b;
nzd = 1;
if (a<b) nzd=a;
    else nzd=b;
indikator=1;
while (indikator)
    {
    if (a%nzd)==0 && (b%nzd)==0 indikator=0;
        else    nzd=nzd-1;
    }
output nzd;
}
```

Zadatak 16 *Euklidov algoritam 1.*

```
Algoritam NZD3
ulaz prirodni brojevi a, b;
izlaz nzd(a,b);
{
input a,b;
while (a!=b)
    {
    if (a>b) a=a-b;
        else b=b-a;
    }
output a;
}
```

Zadatak 17 *Euklidov Algoritam 2.*

```
Algoritam NZD4
ulaz prirodni brojevi a, b;
izlaz nzd(a,b);
{
input a,b;
if (a<b) {
        pom=a;
        a=b;
        b=pom;
```

```
        }
while (b!=0)
    {
    pom = b;
    b = a % b;
    a = pom;
    }
output a;
}
```

3.3.2 Koren prirodnog broja

Zadatak 18 *Naći ceo deo korena prirodnog broja.*

```
Algoritam Koren1
ulaz prirodan broj n;
izlaz ceo deo korena prirodnog broja;
{
input n;
koren=1;
while (koren*koren<=n)
            koren=koren+1;
koren=koren-1;
output koren;
}
```

Zadatak 19 *Naći ceo deo korena prirodnog broja.*

```
Algoritam Koren2
ulaz prirodan broj n;
izlaz ceo deo korena prirodnog broja;
{
input n;
suma=0;
broj=0;
while (suma<=n)
            {
            suma=suma+broj+broj+1;
            broj=broj+1;
            }
broj=broj-1;
output broj;
}
```

Glava 4

Programski jezik C

4.1 Zdravo svete!

Primer 4.1.1 *Program štampa poruku "hello, world".*

```
#include <stdio.h>

main()
/*iskazi f-je main su zatvoreni u zagrade */
{
/*poziv f-je printf da odstampa poruku*/
printf("hello, world\n");
}
```

Primer 4.1.2 *Program štampa poruku "hello, world"*

```
#include <stdio.h>

main()
{
printf("hello, ");
printf("world");
printf("\n");
}
```

Specijalni znaci:

```
\n novi red
\t tabulator
\\ kosa crta
\" navodnici
\a zvuk
\' jednstruki navodnik
```

4.2 Imena promenljivih

Postoje ograničenja: u imenu se mogu pojaviti slova i cifre, potcrta "_" se smatra slovom.

Velika i mala slova se razlikuju.
`int x, X; /*To su dve razlicite promenljive!!!*/`
Ključne reči kao što su if, else, for, while, se ne mogu koristiti za imena promenljivih.

4.3 Deklaracije

Da bi se promenljiva mogla upotrebljavati ona se mora na početku programa deklarisati. Prilikom deklaracije može se izvršiti i početna inicijalizacija.

```
int broj; /*Deklaracija celog broja*/
int vrednost=5; /*Deklaracija i inicijalizacija celog broja*/
```

Kvalifikator const može biti dodeljen deklaraciji bilo koje promenljive da bi označio da se ona neće menjati

```
const double e=2.71828182845905
```

4.4 Tipovi i veličina podataka

Osnovni tipovi podataka:

```
int     ceo broj
char    znak, jedan bajt
float   realan broj
double  realan broj dvostruke tacnosti

char    jedan bajt, sadrzi jedan znak
int     celobrojna vrednost,2 ili 4 bajta
float   realan broj, jednostruka tacnost
double dvostruka tacnost
```

Postoje kvalifikatori koje pridružujemo osnovnim tipovima short(16) i long(32):

```
short int kratak_broj;
long int dugacak_broj;
short kratak;
long dugacak;
```

Važi

$$broj\_bajtova(short) <= broj\_bajtova(int) <= broj\_bajtova(long)$$

Postoje kvalifikatori signed i unsigned koji se odnose na označene i neoznačene cele brojeve. Npr.
signed char: -128 do 127
dok je
unsigned char: od 0 do 255.
Float, double i long double.

Primer 4.4.1 *Uvođenje promenljivih u program.*

```
#include <stdio.h>

main()
 {
 /*deklaracija vise promenljivih
 istog tipa */
 int rez,pom1,pom2;
 pom1=20;
 pom2=15;
 rez=pom1-pom2;

 /*ispisivanje rezultata*/
 printf("Rezultat je %d-%d=%d\n",pom1,pom2,rez);
 }
```

Izlaz iz programa:
Rezultat je 20-15=5

Iskaz dodele:
pom1=20;
pom2=15;
Individualni iskazi se zavrsavaju sa ;

4.5 Funkcije printf i scanf

```
printf("%d\t%d\n", broj1, broj2);
uvek je prvi argument izmedju " "
%d ceo broj
\t tab izmedju
\n novi red
Svaka % konstrukcija je u paru sa argumentom koji sledi.
```

Primer 4.5.1

```
#include <stdio.h>
main()
  {
  printf("Slova:\n%3c\n%5c\n", 'z' , 'Z');
  }
```

Izlaz iz programa:

```
Slova:
  z
    Z
```

`%c` *je za stampanje karaktera*
`%3c` *je za stampanje karaktera na tri pozicije*
Isto tako smo mogli i `%3d` *za stampanje broja na tri pozicije ili* `%6d` *za stampanje broja na 6 pozicija.*

Pravila:
`%d` stampaj kao ceo broj

`%6d` stampaj kao ceo broj sirok najvise 6 znakova
`%f` stampaj kao realan broj
`%6f` stampaj kao realan broj sirok najvise 6 znakova
`%.2f` stampaj kao realan broj sa dve decimale
`%6.2f` stampaj kao realan broj sirok najvise 6 znakova a od toga 2 iza decimalne tacke
`%c` karakter
`%s` string
`%x` heksadecimalni broj
`%%` je procenat

Primer 4.5.2 *Prikazuje unos celog broja koristeci* `scanf("%d", &x)`

```
#include <stdio.h>

main()
{
    int x;
    printf("Unesi ceo broj : ");

    /* Obratiti paznju na znak &
       (operator uzimanja adrese)
       pre imena promenljive u funkciji
       scanf */
    scanf("%d",&x);

    /* U funkciji printf nije
    potrebno stavljati &  */
    printf("Uneli ste broj %d\n", x);
}
```

Primer 4.5.3 *Program sabira dva uneta cela broja*

```
#include <stdio.h>

main()
{
    int a, b, c;
    printf("Unesi prvi broj : ");
    scanf("%d", &a);
    printf("Unesi drugi broj : ");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
}
Ulaz:
Unesi prvi broj : 2 <enter>
Unesi drugi broj : 3 <enter>
Izlaz:
2 + 3 = 5
```

4.6 Aritmetički operatori

```
+ - * /
% (samo za celobrojne vrednosti)
unarno + i -
```

Asocijativnost sleva na desno, prioritet kao u matematici.

Primer 4.6.1 *Program ilustruje neke od aritmetičkih operacija.*

```
#include <stdio.h>
main()
{
int a, b;
printf("Unesi prvi broj : ");
scanf("%d",&a);

printf("Unesi drugi broj : ");
scanf("%d",&b);

/* Kada se saberu dva cela broja, rezultat je ceo broj*/
printf("Zbir a+b je : %d\n",a+b);
/* Kada se oduzmu dva cela broja, rezultat je ceo broj*/
printf("Razlika a-b je : %d\n",a-b);
/* Kada se pomnoze dva cela broja, rezultat je ceo broj*/
printf("Proizvod a*b je : %d\n",a*b);
/* Kada se podele dva cela broja, rezultat je ceo broj!!!*/
printf("Celobrojni kolicnik a/b je : %d\n", a/b);
/* Rezultat je ceo broj, bez obzira sto ga ispisujemo kao realan*/
printf("Pogresan pokusaj racunanja realnog kolicnika a/b je : %f\n", a/b);
/* Eksplicitna konverzija, a i b pretvaramo u relane brojeve kako
   bi deljenje bilo realno*/
printf("Realni kolicnik a/b je : %f\n", (float)a/(float)b);
/* Ostatak pri deljenju se moze izvrsiti samo nad celim brojevima*/
printf("Ostatak pri deljenju a/b je : %d\n", a%b);
}

Ulaz:
Unesi prvi broj : 2 <enter>
Unesi drugi broj : 3 <enter>
Izlaz:
Zbir a+b je : 5
Razlika a-b je : -1
Proizvod a*b je : 6
Celobrojni kolicnik a/b je : 0
Progresan pokusaj racunanja realnog kolicnika a/b je : 0.000000
Realni kolicnik a/b je : 0.666667
Ostatak pri deljenju a/b je : 2
```

Primer 4.6.2 *Program ilustruje celobrojno i realno deljenje.*

```
#include <stdio.h>

main()
{
int a = 5;
int b = 2;
int d = 5/2;    /* Celobrojno deljenje - rezultat je 2 */
float c = a/b;  /* Iako je c float, vrsi se celobrojno
                        deljenje jer su i a i b celi */

/* Neocekivani rezultat 2.000000 */
printf("c = %f\n",c);
printf("Uzrok problema : 5/2 = %f\n", 5/2);
printf("Popravljeno : 5.0/2.0 = %f\n", 5.0/2.0);
printf("Moze i : 5/2.0 = %f i 5.0/2 = %f \n", 5/2.0, 5.0/2);
printf("Za promenljive mora kastovanje : %f\n", (float)a/(float)b);

}

Izlaz iz programa:
c = 2.000000
Uzrok problema : 5/2 = 2.000000
Popravljeno : 5.0/2.0 = 2.500000
Moze i : 5/2.0 = 2.500000 i 5.0/2 = 2.500000
Za promenljive mora kastovanje : 2.500000
```

Zadatak 20 *Šta će biti ispisano nakon izvršavanja sledećeg programa?*

```
#include <stdio.h>
main()
 {
     int x=506, y=3, z=21, t=2;
     printf("x=%d y=%d\n",x,y);
     printf("z - t=%d\n", z-t);
     printf("z / t =%d\n",z / t);
     printf("-x=%d\n",- x);
     printf("x %% y=%d\n", x%y);
}
```

4.7 Operatori i izrazi dodeljivanja vrednosti

```
i = i + 2;
ekvivalento je sa
i+=2;

Moze i za:
+ - * / % << >> ^ |
izraz1 op = izraz2
je ekvivalnetno sa
izraz1 = (izraz1) op (izraz2)
```

```
x*= y+1 je ekvivalento sa x = x * (y+1)
```

Takvo pisanje je krace i efikasnije.

4.8 Inkrementacija i dekrementacija

Operatori ++ i --

```
x=++n; se razlikuje od x=n++;

y=(x++)*(++z);
```

Primer 4.8.1 *Ilustracija prefiksnog i postfiksnog operatora ++*

```
#include <stdio.h>
main()
{
int x, y;
int a = 0, b = 0;

printf("Na pocetku : \na = %d\nb = %d\n", a, b);

/* Ukoliko se vrednost izraza ne koristi, prefiksni i
    postfiksni operator se ne razlikuju */
a++;
++b;
printf("Posle : a++; ++b; \na = %d\nb = %d\n", a, b);

/* Prefiksni operator uvecava promenljivu, i rezultat
    je uvecana vrednost */
x = ++a;

/* Postfiksni operator uvecava promenljivu, i rezultat je
stara (neuvecana) vrednost */
y = b++;

printf("Posle : x = ++a; \na = %d\nx = %d\n", a, x);
printf("Posle : y = b++; \nb = %d\ny = %d\n", b, y);
}

Izlaz iz programa:
Na pocetku:
a = 0
b = 0
Posle : a++; ++b;
a = 1
b = 1
Posle : x = ++a;
a = 2
```

```
x = 2
Posle : y = b++;
b = 2
y = 1
```

4.9 Relacioni i logički operatori

Relacioni operatori:

```
>   >=      <   <= isti prioritet
== !=               nizi prioritet

(3<5)
(a<=10)
 a < 5 != 1  <=> (a < 5)!=1
```

Logicki operatori:

```
! unarna negacija (najvisi prioritet)
&& logicko i (visi prioritet od ili)
|| logicko ili izracunavaju se sleva na desno!

5 && 4 vrednost je tacno
10 || 0 vrednost je tacno
0 && 5 vrednost je 0
!1 vrednost je 0
!9 vrednost je 0
!0 vrednost je 1
!(2>3) je 1
a>b && b>c || b>d je isto sto i ((a>b) && (b>c)) || (b>d)
koja je vrednost ako je a=10, b=5, c=1, d=15?
```

Primer 4.9.1 *Ilustracija logičkih i relacijskih operatora.*

```
#include <stdio.h>

main()
{
    int a = 3>5,  /* manje */
    b = 5>3,  /* vece */
    c = 3==5, /* jednako */
    d = 3!=5; /* razlicito */

    printf("3>5 - %d\n5>3 - %d\n3==5 - %d\n3!=5 - %d\n", a, b, c, d);

    /*Lenjo izracunavanje: kako 3 nije vece od 5 to se vrednost
    drugog poredjenja nece racunati jer je netacno u konjunkciji
    sa proizvoljnim izrazom sigurno netacno. */
    printf("Konjunkcija : 3>5 && 5>3 - %d\n", a && b);

    /*Lenjo izravunavanje: tacno u disjunkciji sa proizvoljnim
```

```
    izrazom daje tacno tako da se vrednost izraza 3>5 nece
    izracunavati*/
    printf("Disjunkcija : 5>3 || 3>5 - %d\n", b || a);
    printf("Negacija : !(3>5) - %d\n", !a);

}

Izlaz iz programa:
3>5 - 0
5>3 - 1
3==5 - 0
3!=5 - 1
Konjunkcija : 3>5 && 5>3 - 0
Disjunkcija : 3>5 || 5>3 - 1
Negacija : !(3>5) - 1
```

Primer 4.9.2 *Ilustracija lenjog izračunavanja logičkih operatora.*

Prilikom izracunavanja izraza - A && B, ukoliko je A netačno, izraz B se ne izračunava. Prilikom izračunavanja izraza - A || B, ukoliko je A tačno, izraz B se ne izračunava.

```
#include <stdio.h>

/* Globalna promenljiva, vidljiva i iz funkcije main() i
   iz funkcije izracunaj*/
int b = 0;

/* Funkcija ispisuje da je pozvana i uvecava promenjivu b.
   Funkcija uvek vraca vrednost 1 (tacno)
*/
int izracunaj()
{
    printf("Pozvano izracunaj()\n");
    b++;
    return 1;
}

main()
{
/* Funkcija izracunaj() ce biti pozvana
   samo za parne vrednosti a */
int a;
for (a = 0; a < 10; a++)
    if (a%2 == 0 && izracunaj())
        printf("Uslov ispunjen : a = %d, b = %d\n", a, b);
    else
        printf("Uslov nije ispunjen : a = %d, b = %d\n", a, b);

printf("----------------------------\n");

/* Funkcija izracunaj() ce se pozivati samo
```

```
   za neparne vrednosti a */
b = 0;
for (a = 0; a < 10; a++)
    if (a%2 == 0 || izracunaj())
        printf("Uslov ispunjen : a = %d, b = %d\n", a, b);
    else
        printf("Uslov nije ispunjen : a = %d, b = %d\n", a, b);
}
```

4.10 Kontrola toka — if, while, do - while, for

4.10.1 if

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

Primer 4.10.1 *Program ilustruje if i ispisuje ukoliko je uneti ceo broj negativan*

```
#include <stdio.h>

main()
{
    int b;
    printf("Unesi ceo broj:");
    scanf("%d", &b);
    if (b < 0)
        printf("Broj je negativan\n");
}
```

Else se odnosi na prvi neuparen if, voditi o tome računa, ako želimo drugačije moramo da navedemo vitičaste zagrade.

```
if (izraz)
    if (izraz1) iskaz 1
else iskaz
```

ovo else se odnosi na drugo if a ne na prvo if!

```
if (izraz)
    {
    if (izraz1) iskaz 1
    }
else iskaz
```

tek sada se else odnosi na prvo if!!!

4.10.2 Else-if

```
if (izraz1)
    iskaz1
```

```
else if (izraz2)
    iskaz2
else if (izraz3)
    iskaz3
else if (izraz4)
    iskaz4
else iskaz

npr if (a<5)
    printf("A je manje od 5\n");
else if (a=5)
    printf("A je jednako 5\n");
else if (a>10)
    printf("A je vece od 10\n");
else if (a=10)
    printf("A je jednako 10\n");
else printf("A je vece od pet i manje od 10\n");
```

Primer 4.10.2 *Program ilustruje if-else konstrukciju i ispituje znak broja.*

```
#include <stdio.h>

main()
{
    int b;
    printf("Unesi ceo broj : ");
    scanf("%d", &b);
    if (b < 0)
        printf("Broj je negativan\n");
    else if (b == 0)
        printf("Broj je nula\n");
    else
        printf("Broj je pozitivan\n");
}

Ulaz:
Unesi ceo broj:-5
Izlaz:
Broj je negativan

Ulaz:
Unesi ceo broj:5
Izlaz:
Broj je pozitivan
```

Primer 4.10.3 *Pogresan program sa dodelom = umesto poredjenja ==.*

```
#include <stdio.h>

main()
```

```
{
    int b;
    printf("Unesi ceo broj : ");
    scanf("%d", &b);

    /* Obratiti paznju na = umesto == Analizirati rad programa*/
    if (b = 0)
        printf("Broj je nula\n");
    else if (b < 0)
        printf("Broj je negativan\n");
    else
        printf("Broj je pozitivan\n");
}

Ulaz:
Unesi ceo broj:-5
Izlaz:
Broj je pozitivan
```

4.10.3 while

`while(uslov) { ... }`
Uslov u zagradi se testira i ako je ispunjen telo petlje se izvrsava. Zatim se uslov ponovo testira i ako je ispunjen ponovo se izvrsava telo petlje. I tako sve dok uslov ne bude ispunjen. Tada se izlazi iz petlje i nastavlja sa prvom sledecom naredbom u programu.

Ukoliko iza while sledi samo jedna naredba nema potrebe za zagradama.

```
while (i<j)
    i=2*i;
```

4.10.4 do-while

Ovo je slično paskalskom repeat-until izrazu.

```
do iskaz while (izraz)
```

Primer 4.10.4 *Program ilustruje petlju do-while.*

```
#include <stdio.h>

main()
{
    int x;

    x = 1;
    do
    {
        printf("x = %d\n",x);
        x++; /* x++ je isto kao i x=x+1 */
```

```
    } while (x<=10);
}
```

4.10.5 for

Primer 4.10.5 *Program ilustruje petlju - for.*

```
#include <stdio.h>

main()
{
    int x;

    /* Inicijalizacija; uslov; inkrementacija*/
    for (x = 1; x < 5; x++)
        printf("x = %d\n",x);

}
Izlaz:
1
2
3
4
```

4.11 Switch

```
switch (iskaz) {
    case konstantan_izraz1: iskazi1
    case konstantan_izraz2: iskazi2
    ...
    default: iskazi
}
```

Primer 4.11.1 *Voditi računa o upotrebi break-a.*

```
#include <stdio.h> /*
 Upotreba switch-a
*/

main() {
    char x;
    scanf("%c",&x);

    switch (x)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': printf(" x je samoglasnik");
```

```
                break;
        case 'r': printf(" x je r");
                break;
        default:  printf(" x je suglasnik");
    }
}
```

Primer 4.11.2 *Ilustracija switch konstrukcije.*

```
#include<stdio.h>
main()
{
int n;
printf("Unesi paran broj manji od 10\n");
scanf("%d",&n);
switch(n) {
case 0:
      printf("Uneli ste nulu\n");
      break;
case 2:
      printf("Uneli ste dvojku\n");
      break;
case 4:
      printf("Uneli ste cetvorku\n");
      break;
case 6:
      printf("Uneli ste sesticu\n");
      break;
case 8:
      printf("Uneli ste osmicu\n");
      break;
defalut:
      printf("Uneli ste nesto sto nije paran broj\n");
}
}
Ulaz:
Unesi paran broj manji od 10
2
Izlaz:
Uneli ste dvojku
```

4.12 Uslovni izraz

Slično kao `if`.

```
izraz1 ? izraz2 : izraz3

z = (a<b)? a : b; /*z=min(a,b)*/
max = (a>b)? a : b;
```

4.13 Simboličke konstante

Primer 4.13.1 *Konverzija centimetara u inče - while petlja.*

```
#include <stdio.h>

/* Definicija simbolickih konstanti preko #define direktiva */
/* U fazi pretprocesiranja se vrsi doslovna zamena konstanti
   njihovim vrednostima */

#define POCETAK 0
#define KRAJ 20
#define KORAK 10

main()
{
    int a;
    a = POCETAK;
    while (a <= KRAJ)
    {
        printf("%d cm = %f in\n", a, a/2.54);
        a += KORAK; /* isto sto i a = a + KORAK; */
    }
}

Izlaz:
0 cm = 0.000000 in
10 cm = 3.937008 in
20 cm = 7.874016 in
```

Primer 4.13.2 *Konverzija centimetara u inče - for petlja.*

```
#include <stdio.h>
#define POCETAK 0
#define KRAJ 20
#define KORAK 10

main()
{
    int a;
    for  (a = POCETAK; a <= KRAJ; a += KORAK)
        printf("%d cm = %f in\n", a, a/2.54);
}

Izlaz:
0 cm = 0.000000 in
10 cm = 3.937008 in
20 cm = 7.874016 in
```

Zadatak 21 *Šta će biti ispisano nakon izvršavanja sledećeg programa?*

```
#include <stdio.h>
#define EURO 85.90
main()
{
   printf("4 eura ima vrednost %f dinara\n", 4*EURO);
   printf("1 euro ima vrednost %.0f dinara\n",EURO);
}
```

4.14 Enumeracija

Izvesna alternativa za define

```
enum boolean {NO, YES};
enum meseci {JAN = 1, FEB, MAR, APR, MAJ, JUN,
                JUL, AVG, SEP, OKT, NOV, DEC}
enum boje {CRVENA, ZELENA=5, PLAVA,
            LJUBICASTA=10, ZUTA, CRNA}

koriscenje:

int x=0;
boje b;

x=CRVENA+3; /*x ce biti jednako tri*/

b=ZELENA;
x=b+CRNA; /* 5 + 12=17*/

b=0; /*Greska, ovako ne moze!!!*/
```

4.15 Funkcije

Primer 4.15.1 *sum - najjednostavnija funkcija koja sabira dva broja*

```
/* Definicija funkcije */
int sum(int a, int b)
{
    int c;
    c = a + b;
    return c;
    /* Ovo je krace moglo da bude napisano
       kao return a+b; */
}

main()
{

    int c;
```

```
    /* Poziv funkcije */
    c = sum(3,5);
    printf("%d\n", c);

    /* Ovo smo krace mogli da napisemo kao
    printf("%d\n", sum(3,5)); */
}
```

Primer 4.15.2 *Deklaracija funkcije moze da stoji nezavisno od definicije funkcije. Deklaracija je neophodna u situacijama kada se definicija funkcije navodi nakon upotrebe date funkcije u kodu.*

```
/* Deklaracija funkcije*/
int zbir(int, int);

main()
{
    /* Poziv funkcije */
    printf("%d\n", zbir(3,5));
}

/* Definicija funkcije */
int zbir(int a, int b)
{
    return a+b;
}
```

Primer 4.15.3 *power - funkcija koja stepenuje realan broj na celobrojni izlozilac*

```
 #include <stdio.h>

/* stepenuje x^k tako sto k puta pomnozi x */
float power(float x, int k)
{
    int i;
    float rezultat = 1;
    for (i = 0; i<k; i++)
        rezultat*=x;

    return rezultat;
}
```

Primer 4.15.4 *Verzija koja radi i za negativne izlozioce*

```
float power(float x, int k)
{
    int i;
    float s = 1;
    int negativan = (k<0);
    float rezultat;
```

```
    if (negativan)
        k = -k;

    for (i = 0; i<k; i++)
        s*=x;

    rezultat = negativan ? 1.0/s : s;
    return rezultat;
}

main()
{
    /* Poziv funkcije */
    float s = power(2.0,8);
    printf("%f\n", s);
}
```

4.16 Nizovi

Deklaracija niza:

```
int niz[5]; /* niz od 5 elemenata tipa int*/
```

Pristupanje elementima niza:

```
niz[0] = 4;
niz[1] = 2 * niz[0];        /*niz[1] = 8*/
niz[2] = niz[0] * niz[1];   /*niz[2] = 32*/
niz[3] = 5;
niz[4] = 7;
```

Unos vrednosti elemenata niza sa tastature:

```
for(i=0; i<5; i++)
    scanf("%d", &a[i]);
```

Stampanje elemenata niza

```
for(i=0; i<5; i++)
    printf("%d ", a[i]);
```

Brojanje elemenata niza je od nule!
Pristupanje elementu niza, indeks može da bude proizvoljan izraz celobrojne vrednosti: `niz[i*2]=5`.

Primer 4.16.1 *Program ilustruje korišćenje nizova. Ispisuje 10 unetih brojeva unazad.*

```
#include <stdio.h>

main()
{
    int a[10];
```

```
    int i;
    for (i = 0; i<10; i++)
    {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
    }

    printf("Unazad : \n");

    for (i = 9; i>=0; i--)
        printf("a[%d]=%d\n",i,a[i]);
}
```

Primer 4.16.2 *Program pronalazi maksimum brojeva sa ulaza - verzija sa nizom.*

```
#include <stdio.h>
#define BR_ELEM 5
main()
{
    int a[BR_ELEM];
    int i;
    int max;

    /* Ucitavamo niz brojeva */
    for (i = 0; i < BR_ELEM; i++)
        scanf("%d",&a[i]);

    /* Pronalazimo maksimum */
    max = a[0];
    for (i = 1; i < BR_ELEM; i++)
        if (a[i]>max)
            max = a[i];

    /* Ispisujemo maksimum */
    printf("Max = %d\n",max);
}
```

Primer 4.16.3 *Program pronalazi maksimum brojeva sa ulaza - verzija bez niza.*

```
#include <stdio.h>
#define BR_ELEM 5

main()
{
    int a, max, i;
    scanf("%d",&a);
    max = a;
    for (i = 1; i < BR_ELEM; i++)
    {
```

```
        scanf("%d",&a);
        if (a>max)
            max = a;
    }

    printf("Max : %d\n", max);
}
```

4.17 Konstante

Koji su tipovi konstanti?

Celobrojna konstanta 1234 je tipa int.
Da bi konstanta bila long navodi se iza nje slovo L ili l, npr 123456789L.
Ako želimo da nam je konstanta unsigned onda na kraju pišemo U ili u.
Može i 1234567ul.

Konstante realnih brojeva sadrže decimalnu tačku(123.4) ili eksponent(1e-2) ili i jedno i drugo. Njihov tip je double osim ako nemaj sufiks f ili F kada je u pitanju float. L ili l označavaju long double.

Oktalna konstanta počinje sa 0, a heksadecimalna sa 0x. Npr broj 31 ili 037 - oktalno ili 0x1f - heksadecimalno. I one mogu da imaju U i L na kraju.

Znakovna konstanta je celobrojna vrednost napisana između jednostrukih navodnika. Vrednost date konstante je numerička vrednost datog znaka u računarskom setu znakova. Npr možemo da pišemo '0' umesto 48.
!!!Razlikovati znakovne konstante i niske koje se navode između dvostrukih navodnika!

Posebni znaci su znak za kraj reda, tab i slično.

Znakovna konstanta `'\0'` predstavlja znak čija je vrednost nula, treba ga razlikovati od '0' koja je znak čija je vrednost 48.

Konstantna niska: "Ja sam niska"
ili
"" /*prazna niska*/
Navodnici nisu deo niske već se koriste da bi je ograničili. Ako ih želimo unutar niske, oni se navode sa `\"`.

Konstantna niska je polje znakova. Da bi se znalo gde je kraj niske, fizičko memorisanje liste zahteva da postoji jedan znak više koji označava kraj, to je `'\0'`. Da bi se odredila dužina niske mora se proći kroz celu nisku.

!!!Važno:

Koja je razlika između "x" i 'x'?

Primer 4.17.1 *Primer funkcije koja izračunava dužinu niske znakova.*

```
#include <stdio.h>

int strlen(char s[])
{
    int i=0;

    while (s[i] != '\0')
```

```
        ++i;
    return i;
}

int main()
{
printf("Duzina ove niske
        je: %d \n",strlen("Duzina ove niske je:"));
return 0;
}
```

4.18 Konverzija

4.18.1 Automatska konverzija

Ako je jedan od operanada različličit vrši se konverzija, uvek u smeru manjeg ka većem tipu

Naredba dodele:

```
int i=5;
float f=2.3;
f=i; /* f ce imati vrednost 5.0*/

obrnuto:

int i=5;
float f=2.3;
i=f; /* i ce imati vrednost 2*/
```

4.18.2 Eksplicitna konverzija

```
(tip)<izraz>

float x;
x=2.3+4.2;          /* x ce imati vrednost 6.5 */
x=(int)2.3+(int)4.2; /* x ce imati vrednost 6 */
x=(int)2.3*4.5;     /* x ce imati vrednost 9.0 jer zbog prioriteta
                    operatora konverzije prvo ce biti izvrsena
                    konverzija broja 2.3 u 2 pa tek onda izvrseno
                    mnozenje. */
x=(int)(2.3*4.5)    /* x ce imati vrednost 10.0 */
```

Primer 4.18.1 *Kako izbeći celobrojno deljenje*

```
int a,b;
float c;
a = 5;
b = 2;
c = a/b; /* Celobrojno deljenje, c=2*/
c = (1.0*a)/b;  /* Implicitna konverzija: 1.0*a je realan
```

```
                broj pa priliko deljenja sa b dobija se
                realan rezultat c=2.5*/
c = (0.0+a)/b;  /* Implicitna konverzija: (0.0+a) je realan
                broj pa priliko deljenja sa b dobija se
                realan rezultat c=2.5*/
c = (float)a/(float)b; /* Eksplicitna konverzija*/
```

4.18.3 Funkcije koje vrše konverziju

Primer 4.18.2

```
#include <stdio.h>
main()
{
int vrednost;
vrednost='A';
printf("Veliko slovo\n karakter=%3c\nvrednost=%3d\n",vrednost,vrednost);
vrednost='a';
printf("Malo\n karakter=%3c\nvrednost=%3d\n",vrednost,vrednost);
}

Izlaz (u slucaju ASCII):
Veliko slovo
karakter= A
vrednost= 65
Malo
karakter= a
vrednost= 97
```

Primer 4.18.3 *Funkcija koja konvertuje velika slova u mala slova.*

```
#include<stdio.h>

/* Konvertuje karakter iz velikog u malo slovo */
char lower(char c)
{
if (c >= 'A' && c <= 'Z')
    return c - 'A' + 'a' ;
else
    return c;
}

main()
{
char c;
printf("Unesi neko veliko slovo:\n");
scanf("%c", &c);
printf("Odgovarajuce malo slovo je %c\n", lower(c));
}
```

Primer 4.18.4 *Konvertovanje niske cifara u ceo broj.*

```
#include<stdio.h>

/* atoi:  konvertuje s u ceo broj */
int atoi(char s[])
{
int i, n;
n = 0;
for (i = 0; (s[i] >= '0') && (s[i] <= '9'); ++i)
    n = 10 * n + (s[i] - '0');
return n;
}

main()
{
int n;
n = atoi("234");
printf("\nN je : %d\n",n);
}
```

4.19 Operator `sizeof()`

Primer 4.19.1 *Demonstracija sizeof operatora.* ***`Sizeof`*** *operator izračunava veličinu tipa odnosno promenjive.*

```
#include<stdio.h>
main()
{
int i;
float f;
int n[10];

printf("sizeof(int)=%d\n", sizeof(int));
printf("sizeof(long)=%d\n", sizeof(long));
printf("sizeof(short)=%d\n", sizeof(short));
printf("sizeof(signed)=%d\n", sizeof(signed));
printf("sizeof(unsigned)=%d\n", sizeof(unsigned));
printf("sizeof(char)=%d\n", sizeof(char));
printf("sizeof(float)=%d\n", sizeof(float));
printf("sizeof(double)=%d\n", sizeof(double));

printf("sizeof(i)=%d\n", sizeof(i));
printf("sizeof(f)=%d\n", sizeof(f));
printf("sizeof(n)=%d\n", sizeof(n));
printf("Broj elemenata niza n : %d\n", sizeof(n)/sizeof(int));

}

Izlaz iz programa(u konkretnom slucaju):
sizeof(int)=4
sizeof(long)=4
```

```
sizeof(short)=2
sizeof(signed)=4
sizeof(unsigned)=4
sizeof(char)=1
sizeof(float)=4
sizeof(double)=8
sizeof(i)=4
sizeof(f)=4
sizeof(n)=40
Broj elemenata niza n : 10
```

4.20 Znakovni ulaz i izlaz

Funkcija za čitanje jednog znaka sa ulaza
`c = getchar()`
promenljiva c sadrži jedan znak sa ulaza.

Funkcija za štampanje jednog znaka na izlaz
`putchar(c)`
štampa sadržaj promenljive c obično na ekranu.

Konstanta `EOF` je celobrojna vrednost definisana u biblioteci `<stdio.h>`. Ovu vrednost vrati funkcija `getchar()` kada nema više ulaza. Nazvana je `EOF` kao `End Of File`, kraj datoteke. Ova vrednost mora da se razlikuje od svake vrednosti koja može da bude karakter. Zato za `c` za koje je `c=getchar()` treba da koristimo tip dovoljno veliki da moze da prihvati sve sto moze da vrati `getchar()`, dakle i `EOF`. Zbog toga se za `c` koristi tip `int`.

Primer 4.20.1 *Program cita jedan karakter i ispisuje ga - demonstracija putchar i getchar.*

```
#include <stdio.h>

main()
{
    int c;         /* Karakter - obratiti paznju na int */
    c = getchar(); /* cita karakter sa standardnog ulaza */
    putchar(c);    /* pise karakter c na standardni izlaz */

    putchar('\n'); /* prelazak u novi red */
    putchar('a');  /* ispisuje malo a */
    putchar(97);   /* ekvivalentno prethodnom */
}
Ulaz:
s
Izlaz iz programa:
s
s
aa
```

Primer 4.20.2 *Program prepisuje standardni ulaz na standardni izlaz. Ilustracija redirekcije standardnog ulaza i izlaza, pokrenuti program sa :*

```
./a.out <primer.c
./a.out >tekst.txt
./a.out <primer.c >kopija.c
```

```
#include <stdio.h>

main()
{
/*Koristi se int a ne char zato sto zelimo
  da razlikujemo kraj ulaza od vazecih znakova*/
int c;

/*Ucitava se prvi znak sa ulaza*/
c = getchar();

/*EOF predstavlja celobrojnu vrednost kraja datoteke.
  To je konstanta definisana u <stdio.h>*/
while (c != EOF) {
    putchar(c);
    c = getchar();
    }
}
```

Bilo koje dodeljivanje vrednosti je izraz koji ima vrednost a to je vrednost leve strane posle dodeljivanja.

Primer 4.20.3 *Program koji kopira ulaz na izlaz, skraćeni kod.*

```
#include <stdio.h>

main()
{
    int c;

    /* Obratiti paznju na raspored zagrada */
    while ((c = getchar()) != EOF)
        putchar(c);
}
```

Primer 4.20.4 *Brojanje znakova na ulazu.*

```
#include <stdio.h>

main()
{
    long nc;

    nc = 0;
    while (getchar() != EOF)
```

```
        ++nc;
    /* %ld odnosi se na tip long. */
    printf("%ld\n", nc);
}
```

Primer 4.20.5 *Brojanje znakova na ulazu korišćenjem for petlje.*

```
#include <stdio.h>

  main()
   {
       double nc;

        /*For petlja mora da ima telo pa makar
        ono bilo przno*/
       for (nc = 0; getchar() != EOF; ++nc)
           ;
       printf("%.0f\n", nc);
   }
```

Primer 4.20.6 *Program broji linije i znakove na ulazu.*

```
#include <stdio.h>
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);
}
```

Primer 4.20.7 *Program broji blankove, horizontalne tabulatore i linije na ulazu.*

```
#include <stdio.h>
main()
{
int znak;       /*prihvata znak sa ulaza */
int Blanks=0;   /*brojac blankova */
int Tabs=0;     /*brojac horizontalnih tabulatora */
int NewLines=0; /*brojac linija */

/*UOCITI: blok naredbi while ciklusa NIJE OGRADJEN
  viticastim zagradama jer postoji samo jedna if naredba! */
while( (znak=getchar())!=EOF )
    if( znak==' ' ) ++Blanks; /* brojimo blanko simbole */
```

```
    else if( znak=='\t' ) ++Tabs; /* brojimo tab-ove */
        else if( znak=='\n' ) ++NewLines; /* brojimo redove */

/*izdavanje rezultata na standardni izlaz*/
printf("Blankova: %d. Tabulatora: %d. Prelazaka u novi red: %d\n",
                                                Blanks, Tabs, NewLines);

}
```

Primer 4.20.8 *Program prepisuje ulaz na izlaz pri čemu više blanko znakova zamenjuje jednim.*

```
#include <stdio.h>

main()
{
   int znak; /*tekuci znak sa ulaza*/
   int preth; /*znak koji prethodi tekucem */
   preth='a'; /* inicijalizujemo vrednost prethodnog
                 da bi prvi prolazak kroz petlju bio ispravan*/
     while ( (znak=getchar() ) !=EOF)
       {
         if (znak !=' ' || preth != ' ') putchar(znak);
            preth=znak;
        }
}
```

Primer 4.20.9 *Program vrši prebrojavanje cifara unetih na ulazu.*

```
#include <stdio.h>

main()
{
    int c;
    int br_cifara = 0;
    while ((c = getchar()) != EOF)
        if ('0'<=c && c<='9')
            br_cifara++;

    printf("Broj cifara je : %d\n", br_cifara);
}
```

Primer 4.20.10 *Program vrši brojanje pojavljivanja karaktera 0, 1 i 2 (ilustruje switch).*

```
#include <stdio.h>

main()
{
    int c;
    int br_0=0, br_1=0, br_2=0;
```

```
    while ((c = getchar()) != EOF)
    {
        switch(c)
        {
            /* Obratiti paznju da nije
                case 0: */
            case '0':
                br_0++;
                break;  /* Isprobati veziju bez break */
            case '1':
                br_1++;
                break;
            case '2':
                br_2++;
                break;
            default:
        }
    }
    printf("Br 0 : %d\nBr 1 : %d\nBr 2 : %d\n",br_0, br_1, br_2);
}
```

4.21 Nizovi

Primer 4.21.1 *Program ilustruje inicijalizaciju nizova.*

```
#include <stdio.h>

main()
{
    /* Niz inicijalizujemo tako sto mu navodimo vrednosti
       u viticasnim zagradama. Dimenzija niza se odredjuje
       na osnovu broja inicijalizatora */
    int a[] = {1, 2, 3, 4, 5, 6};

    /* Isto vazi i za niske karaktera */
    char s[] = {'a', 'b', 'c'};

    /* Ekvivalentno prethodnom bi bilo
    char s[] = {97, 98, 99};
    */

    /* Broj elemenata niza */
    int a_br_elem = sizeof(a)/sizeof(int);
    int s_br_elem = sizeof(s)/sizeof(char);

    /* Ispisujemo nizove */

    int i;
    for (i = 0; i < a_br_elem; i++)
        printf("a[%d]=%d\n",i, a[i]);
```

```
    for (i = 0; i < s_br_elem; i++)
        printf("s[%d]=%c\n",i, s[i]);

}
```

Primer 4.21.2 *Program uvodi niske karaktera terminisane nulom.*

```
#include <stdio.h>

main()
{
    /* Poslednji bajt niske karaktera se postavlja na '\0' tj. 0 */
    char s[] = {'a', 'b', 'c', '\0' };

    /* Kraci nacin da se postigne prethodno */
    char t[] = "abc";

    /* Ispis niske s karakter po karakter*/
    int i;
    for (i = 0; s[i] != '\0'; i++)
        putchar(s[i]);
    putchar('\n');

    /* Ispis niske s koristeci funkciju printf */
    printf("%s\n", s);

    /* Ispis niske t karakter po karakter*/
    for (i = 0; t[i] != '\0'; i++)
        putchar(t[i]);
    putchar('\n');

    /* Ispis niske t koristeci funkciju printf */
    printf("%s\n", t);
}
```

Primer 4.21.3 *Brojanje pojavljivanja svake od cifara. Koriscenje niza brojača.*

```
#include <stdio.h>

/* zbog funkcije isdigit */
#include <ctype.h>

main()
{
    /* Niz brojaca za svaku od cifara */
    int br_cifara[10];
    int i, c;

    /* Resetovanje brojaca */
```

```
    for (i = 0; i < 10; i++)
        br_cifara[i] = 0;

    /* Citamo sa ulaza i povecavamo odgovarajuce brojace */
    while ((c = getchar()) != EOF)
        if (isdigit(c))
            br_cifara[c-'0']++;

    /* Ispis rezultata */
    for (i = 0; i < 10; i++)
        printf("Cifra %d se pojavila %d put%s\n",
            i, br_cifara[i], br_cifara[i] == 1 ? "" : "a");
}
```

Primer 4.21.4 *Brojanje reči*

```
#include <stdio.h>

#define IN   1  /* inside a word */
#define OUT  0  /* outside a
word */

/* count lines, words, and characters in input */

main()
{
int c, nl, nw, nc, state;

state = OUT;

/*Postavljaju se sve tri promenljive na nulu*/
/*Isto kao da smo napisali
nl = (nw = (nc = 0));*/

nl = nw = nc = 0;
while ((c = getchar()) != EOF) {
    ++nc;
    if (c == '\n')
        ++nl;
    /*Operator || znaci OR*/
    if (c == ' ' || c == '\n' || c == '\t')
        state = OUT;
    else if (state == OUT) {
        state = IN;
        ++nw;
                }
}
printf("%d %d %d\n", nl, nw, nc);
}
```

4.22 Dvostruka `for` petlja

Primer 4.22.1 *Dvostruka for petlja*

```
#include<stdio.h>
int main()
{
int i,j;

for(i=1; i<=10; i++)
    {
    for(j=1; j<=10; j++)
        printf("%d * %d = %d\t", i, j, i*j);
    printf("\n");
    }
}
```

Primer 4.22.2 *Trostruka for petlja*

```
#include<stdio.h>
int main()
{
int i,j;

for(i=1; i<=10; i++)
    for(j=1; j<=10; j++)
        {
        for(k=1; k<=10; k++)
            printf("%d * %d * %d = %d\t", i, j, k, i*j*k);
        printf("\n");
        }
}
```

Koliko puta se izvrsi naredba štampanja
`printf("%d * %d * %d = %d\t", i, j, k, i*j*k);`

4.23 Formiranje HTML dokumenta

Primer 4.23.1 *Prilikm pokretanja programa koristiti redirekciju:*
`a.out >primer.html`
kako bi se rezultat rada programa upisao u datoteku `primer.html`.

```
/*Ovaj program formira html dokument*/
#include <stdio.h>
main()
{
printf("<html><head><title>Ova stranica
    je napravljena u c-u</title></head>");
printf("<body><h3 align=center>
            Rezultat </h3></body></html>");
}
```

Primer 4.23.2 *Napisati program koji generiše html dokument sa engleskim alfabetom.*

```
#include <stdio.h>
main()
{
int i;
printf("<HTML><head><title>Engleski alfabet</title><head>\n");
printf("<body><ul>");
for(i=0;i<=25;i++)
    printf("<li> %c %c \n",'A'+i,'a'+i);
printf("</ul></body></HTML>\n"); }
```

Primer 4.23.3 *Napisati program koji generise html dokument koji prikazuje tablicu mnozenja za brojeve od 1 do 10.*

```
#include<stdio.h>
main()
{
int i,j;
printf("<html><head><title>Mnozenje</title></head>");
printf("<body><h3 align=center> Rezultat </h3>");
printf("<table border=1>\n");

/* Prva vrsta sadrzi brojeve od 1 do 10*/
printf("<tr>");
printf("<th></th>");
for(i=1; i<=10; i++)
    printf("<th> %d </th>\n", i);
printf("</tr>");

for(i=1; i<=10; i++)
    {
    printf("<tr>");

    /* Na pocetku svake vrste stampamo broj
    odgovarajuce vrste*/
    printf("<th>%d</th>", i);

    for(j=1; j<=10; j++)
        printf("<td>%d\t</td>\n", i*j);

    printf("</tr>");
}
printf("</table>");
printf("</body></html>");
}
```

4.24 Funkcije — prenos parametara po vrednosti

Primer 4.24.1 *Funkcija koja proverava da li je broj prost i program koji ispisuje sve proste brojeve manje od 100.*

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

int prost(int p)
{
    int i, koren, ind;
    koren=sqrt(p);
    ind=(p%2) || (p==2);
    i=3;
    while (ind && i<=koren)
        {
        ind=p%i;
        i+=2;
        }
    return ind;
}

main()
{
int k;
for(k=2;k<=100;k++)
    if (prost(k)) printf("%d ",k);
}
```

Primer 4.24.2

```
#include <stdio.h> /* Ilustruje vidljivost imena*/

int i=10;

void main() {
    {
        int i=3;
        {
            int i=1;
            printf("%d\n", i);
        }
        printf("%d\n",i);
    }
     printf("%d\n",i);
}
```

Primer 4.24.3 *Demonstracija prenosa parametara po vrednosti - preneti parametri se ne mogu menjati*

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

```
void f(int x)
{
x++;
}

main()
{
int x=3;
f(x);
printf("%d\n", x);
}
```

Primer 4.24.4 *Demonstrira prenos nizova u funkciju - preneti niz se moze menjati.*

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

/* Funkcija ucitava rec sa standardnog ulaza i smesta je u niz karaktera s.
   Ovo uspeva zbog toga sto se po vrednosti prenosi adresa pocetka niza,
   a ne ceo niz */
void get_word(char s[])
{
    int c, i = 0;
    while (!isspace(c=getchar()))
        s[i++] = c;
    s[i] = '\0';
}

main()
{
    /* Obavezno je alocirati memoriju za niz karaktera */
    char s[100];

    get_word(s);
    printf("%s\n", s);
}
```

Primer 4.24.5 *Funkcija za ispis niza brojeva - demonstrira prenos nizova brojeva u funkciju.*

```
#include <stdio.h>

/* Nizovi se prenose tako sto se prenese adresa njihovog pocetka.
   Uglaste zagrade ostaju prazne!

   Nizove je neophodno prenositi zajedno sa dimenzijom niza
   (osim za niske karaktera, jer tamo vazi konvencija da
   se kraj niza obelezava znakom '\0')
*/
void print_array(int a[], int n)
```

```
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ",a[i]);
    putchar('\n');

    /* Obratite paznju na ovo : */
    printf("sizeof(a) - u okviru fje : %d\n", sizeof(a));
}

main()
{
    int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

    printf("sizeof(a) - u okviru main : %d\n", sizeof(a));
    print_array(a, sizeof(a)/sizeof(int));
}
```

Primer 4.24.6 *Funkcija za ispis niske karaktera - demonstrira prenos niske karaktera u funkciju.*

```
#include <stdio.h>

/* Uz nisku karaktera nije potrebno prenositi dimenziju
   ukoliko se postuje dogovor
   da se svaka niska zavrsava karakterom '\0'.*/
void print_string(char s[])
{
    int i;
    for (i = 0; s[i]!='\0'; i++)
        putchar(s[i]);
}


main()
{
    print_string("Zdravo\n");
}
```

4.25 Break i continue

Primer 4.25.1 *Funkcija uklanja beline, tabulatore ili znak za kraj reda sa kraja stringa.*

```
int trim(char s[])
{
int n;
for (n = strlen(s)-1; n >= 0; n--)
    if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
                break;
```

```
s[n+1] = '\0';
return n;
}
```

`Continue` se ređe koristi, on prouzrokuje da se pređe na sledeću iteraciju u petlji.

Primer 4.25.2

```
for(i=0; i<n; i++)
{
if (a[i]==0) continue;
... /* obradi pozitivne elemente nekako*/
}
```

4.26 Rad sa niskama karaktera

Primer 4.26.1 *string_reverse - obrće nisku karaktera.*

```
#include <stdio.h>

/* Ova funkcija racuna duzinu date niske karaktera.
   Umesto nje, moguce je koristiti standardnu funkciju strlen
   za cije je koriscenje potrebno ukljuciti zaglavlje
   <string.h>
 */
int string_length(char s[])
{
    int i;
    for (i = 0; s[i]!='\0'; i++)
        ;

    return i;
}

/* Funkcija obrce nisku karaktera */
void string_reverse(char s[])
{
    int i, j;
    for (i = 0, j = string_length(s)-1; i<j; i++, j--)
    {
        int tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
    }

    /* Napomena : razlikovati prethodnu petlju od dve ugnjezdjene petlje:
    for ( i = 0; ....)
        for ( j = duzina(s)-1; ...
     */
```

```
}

main()
{
    char s[] = "Zdravo svima";
    string_reverse(s);
    printf("%s\n", s);
}
/*
Izlaz:
amivs ovardZ
*/
```

Primer 4.26.2 `strlen`, `strcpy`, `strcat`, `strcmp`, `strchr`, `strstr` - *manipulacija niskama karaktera. Vezbe radi, implementirane su funkcije biblioteke* `string.h`

```
#include <stdio.h>

/* Izracunava duzinu stringa */
int string_length(char s[])
{
    int i;
    /* Uslov s[i] je ekvivalentan uslovu
       s[i]!='\0' ili uslovu s[i]! = 0*/
    for (i = 0; s[i]; i++)
        ;
    return i;
}


/* Kopira string src u string dest.
  Pretpostavlja da u dest ima dovoljno prostora. */
void string_copy(char dest[], char src[])
{
    /* Kopira karakter po karakter, sve dok nije iskopiran karakter '\0' */
    int i;
    for (i = 0; (dest[i]=src[i]) != '\0'; i++)
        ;

    /* Uslov != '\0' se, naravno, moze izostaviti :

    for (i = 0; dest[i]=src[i]; i++)
        ;
    */

}

/* Nadovezuje string t na kraj stringa s.
```

```
   Pretpostavlja da u s ima dovoljno prostora. */
void string_concatenate(char s[], char t[])
{
    int i, j;
    /* Pronalazimo kraj stringa s */
    for (i = 0; s[i]; i++)
        ;

    /* Vrsi se kopiranje, slicno funkciji string_copy */
    for (j = 0; s[i] = t[j]; j++, i++)
        ;
}

/* Vrsi leksikografsko poredjenje dva stringa.
    Vraca :
        0 - ukoliko su stringovi jednaki
        <0 - ukoliko je s leksikografski ispred t
        >0 - ukoliko je s leksikografski iza t
*/
int string_compare(char s[], char t[])
{
    /* Petlja tece sve dok ne naidjemo na prvi razliciti karakter */
    int i;
    for (i = 0; s[i]==t[i]; i++)
        if (s[i] == '\0') /* Naisli smo na kraj oba stringa,
                             a nismo nasli razliku */
            return 0;

    /* s[i] i t[i] su prvi karakteri u kojima se niske razlikuju.
       Na osnovu njihovog odnosa, odredjuje se odnos stringova */
    return s[i] - t[i];
}

/* Pronalazi prvu poziciju karaktera c u stringu s, odnosno -1
   ukoliko s ne sadrzi c */
int string_char(char s[], char c)
{
    int i;
    for (i = 0; s[i]; i++)
        if (s[i] == c)
            return i;
        /* nikako
        else
            return -1;
        */
    /* Nije nadjeno */
    return -1;
}

/* Pronalazi poslednju poziciju karaktera c u stringu s, odnosno -1
```

```
   ukoliko s ne sadrzi c */
int string_last_char(char s[], char c)
{
    /* Pronalazimo kraj stringa s */
    int i;
    for (i = 0; s[i]; i++)
        ;

    /* Krecemo od kraja i trazimo c unazad */
    for (i--; i>=0; i--)
        if (s[i] == c)
            return i;

    /* Nije nadjeno */
    return -1;

    /*
    Koristeci string_length :

    for (i = string_length(s) - 1; i>0; i--)
        if (s[i] == c)
            return i;

    return -1;
    */
}

/* Proverava da li string str sadrzi string sub.
   Vraca poziciju na kojoj sub pocinje, odnosno -1 ukoliko ga nema
*/
int string_string(char str[], char sub[])
{
    int i, j;
    /* Proveravamo da li sub pocinje na svakoj poziciji i */
    for (i = 0; str[i]; i++)
        /* Poredimo sub sa str pocevsi od poziciji i
           sve dok ne naidjemo na razliku */
        for (j = 0; str[i+j] == sub[j]; j++)
            /* Nismo naisli na razliku a ispitali smo
               sve karaktere niske sub */
            if (sub[j+1]=='\0')
                return i;
    /* Nije nadjeno */
    return -1;
}

main()
{
    char s[100];
    char t[] = "Zdravo";
```

```
    char u[] = " svima";

    string_copy(s, t);
    printf("%s\n", s);

    string_concatenate(s, u);
    printf("%s\n", s);

    printf("%d\n",string_char("racunari", 'n'));
    printf("%d\n",string_last_char("racunari", 'a'));

    printf("%d\n",string_string("racunari", "rac"));
    printf("%d\n",string_string("racunari", "ari"));
    printf("%d\n",string_string("racunari", "cun"));
    printf("%d\n",string_string("racunari", "cna"));
}

/*
Izlaz:
Zdravo
Zdravo svima
4
5
0
5
2
-1*/
```

4.27 Makroi

```
#define ime tekst_zamene
```

Zamene se vrše samo na simbolima, a ne obavljaju se u niskama nutar navodnika.

Moguće je definisati makroe sa argumentima tako da tekst zamene bude različit za različita pojavljivanja makroa.

Primer 4.27.1 *Demonstracija pretprocesorske direktive* `#define`

```
#include<stdio.h>

/* Racuna sumu dva broja */
#define sum(a,b) ((a)+(b))

/* Racuna kvadrat broja - pogresna verzija */
#define square_w(a) a*a

/* Racuna kvadrat broja */
#define square(a) ((a)*(a))

/* Racuna minimum tri broja */
```

```
#define min(a, b, c) (a)<(b)? ((a)<(c)? (a) : (c)) : ((b)<(c)? (b) : (c))

main()
{
    printf("sum(3,5) = %d\n", sum(3,5));
    printf("square_w(5) = %d\n", square_w(5));
    printf("square_w(3+2) = %d\n", square_w(3+2));
    printf("square(3+2) = %d\n", square(3+2));
    printf("min(1,2,3) = %d\n", min(1,2,3));
    printf("min(1,3,2) = %d\n", min(1,3,2));
    printf("min(2,1,3) = %d\n", min(2,1,3));
    printf("min(2,3,1) = %d\n", min(2,3,1));
    printf("min(3,1,2) = %d\n", min(3,1,2));
    printf("min(3,2,1) = %d\n", min(3,2,1));
}

Izlaz iz programa:
sum(3,5) = 8
square_w(5) = 25
square_w(3+2) = 11
square(3+2) = 25
min(1,2,3) = 1
min(1,3,2) = 1
min(2,1,3) = 1
min(2,3,1) = 1
min(3,1,2) = 1
min(3,2,1) = 1
```

Primer 4.27.2

```
#define max(A, B) ((A)>(B) ? (A) : (B))
```

na osnovu ovoga će linija

```
x=max(p+q, r+s)
```

biti zamenjena linijom

```
x=((p+q) > (r+s) ? (p+q) : (r+s));
```

Treba voditi računa o sporednim efektima. Sledeća linija koda pruzrokovaće uvećanje vrednosti `i` ili `j` za dva.

```
max(i++, j++)
```

Takođe treba voditi računa o zagradama. Sledeći makro prouzrokovaće neočekivane rezultate za upotrebu `square(a+1)`

```
#define square(x) x*x
```

Primer 4.27.3 *Ilustacija beskonačne petlje:*

```
#define forever for(;;);
```

Primer 4.27.4

```
#include  <stdio.h>
#define max1(x,y) (x>y?x:y)
#define max2(x,y) ((x)>(y)?(x):(y))
#define swapint(x,y) { int z; z=x;  x=y; y=z; }
#define swap(t,x,y) { \
    t z; \
    z=x;  \
    x=y; \
    y=z; }

main()
{

int x=2,y=3;

printf( "max1(x,y) = %d\n", max1(x,y)  );
/* max1(x,y) = 3 */

/* Zamena makroom se ne vrsi
unutar niski pod navodnicima*/
printf( "max1(x=5,y) = %d\n", max1(x,y)  );
/* max1(x=5,y) = 3 */

printf( "max1(x++,y++) = %d\n", max1(x++,y++)  );
/* max1(x++,y++) = 4 */

printf( "x = %d, y = %d\n", x, y  );
/* x = 3, y = 5 */

swapint(x,y);

printf( "x = %d, y = %d\n", x, y  );
/* x = 5, y = 3 */

swap(int,x,y);
printf( "x = %d, y = %d\n", x, y  );
/* x = 3, y = 5 */
}

Izlaz:
max1(x,y) = 3
max1(x=5,y) = 3
max1(x++,y++) = 4
x = 3, y = 5
x = 5, y = 3
x = 3, y = 5
```

4.28 Bitski operatori

!!!Ne mešati sa logičkim operatorima!!!

```
& bitsko AND
| bitsko OR
^ bitsko ekskluzivno OR
<< levo pomeranje
>> desno pomeranje
~ jedinicni komplement
```

Primer 4.28.1 *Demonstracija bitskih operatora*

```
#include <stdio.h>

main()
{  printf("%o %o\n",255,15);
   printf( "255 & 15 = %d\n", 255 & 15 );
   printf( "255 | 15 = %d\n", 255 | 15 );
   printf( "255 ^ 15 = %d\n", 255 ^ 15 );
   printf( "2 << 2   = %d\n", 2 << 2 );
   printf( "16 >> 2  = %d\n", 16 >> 2 );
}

Izlaz iz programa je:
377 17
255 & 15 = 15
255 | 15 = 255
255 ^ 15 = 240
2 << 2   = 8
16 >> 2  = 4
```

Primer 4.28.2 *print_bits - stampa bitove u zapisu datog celog broja x.*

```
#include <stdio.h>

/* Funkcija stampa bitove datog celog broja x.
   Vrednost bita na poziciji i je 0 ako i
   samo ako se pri konjunkciji broja x sa maskom
   000..010....000 - sve 0 osim 1 na poziciji i,
   dobija 0.
   Funkcija krece od pozicije najvece tezine
   kreirajuci masku pomeranjem jedinice u levo
   za duzina(x) - 1 mesto, i zatim pomerajuci
   ovu masku za jedno mesto u levo u svakoj
   sledecoj iteraciji sve dok maska ne postane 0.
*/

void print_bits(int x)
{
    /* Broj bitova tipa unsigned */
```

```
    int wl = sizeof(int)*8;

    unsigned mask;
    for (mask = 1<<wl-1; mask; mask >>= 1)
        putchar(x&mask ? '1' : '0');

    putchar('\n');
}

main()
{
    print_bits(127);
    print_bits(128);
    print_bits(0x00FF00FF);
    print_bits(0xFFFFFFFF);
}

Izlaz iz programa:
00000000000000000000000001111111
00000000000000000000000010000000
00000000111111110000000011111111
11111111111111111111111111111111
```

Primer 4.28.3 *Program proverava da li se na k-tom mestu nalazi 1.*

```
#include <stdio.h>

/* Pozicije brojimo kao u sledecem primeru:
   poz: ... 10 9 8 7 6 5 4 3 2 1 0
   bit: ...  0 0 0 1 1 1 0 0 1 1 0  */

main()
{
   int n,k;
   printf("Unesite broj i poziciju tog broja koju zelite da proverite:\n");
   scanf("%d%d",&n,&k);
   if ((n & (1 << k))!=0)
     printf("Bit je 1\n");
   else
     printf("Bit je 0\n");
}
```

Primer 4.28.4 *Program postavlja na k-to mesto 1*

```
#include <stdio.h>

void print_bits(int x)
{
    /* Broj bitova tipa unsigned */
    int wl = sizeof(int)*8;
```

```
    unsigned mask;
    for (mask = 1<<wl-1; mask; mask >>= 1)
        putchar(x&mask ? '1' : '0');

    putchar('\n');
}

main(){
  int n,k;
  printf("Unesite broj i poziciju tog broja koju zelite da promenite:\n");
  scanf("%d%d",&n,&k);
  printf("Binarno, une\v seni broj je\n");
  print_bits(n);
  printf("Novi broj je %d\n",(n |(1<<k)));
  printf("Binarno, novi broj je\n");
  print_bits((n |(1<<k)));
}
```

Izrazom `a>>b` vrši se pomeranje sadržaja operanda `a` predstavljenog u binarnom obliku za `b` mesta u desno. Popunjavanje upražnjenih mesta na levoj strani zavisi od tipa podataka i vrste računara. Ako se pomeranje primenjuje nad operandom tipa `unsigned` popunjavanje je nulama. Ako se radi o označenom operandu popunjavanje je jedinicama kada je u krajnjem levom bitu jedinica, a nulama kada je u krajnjem levom bitu nula.

Primer 4.28.5 *sum_of_bits - izračunava sumu bitova datog neoznačenog broja.*

```
#include <stdio.h>

/* Pomocna funkcija - stampa bitove neoznacenog broja */
void print_bits(unsigned x)
{
    int wl = sizeof(unsigned)*8;

    unsigned mask;
    for (mask = 1<<wl-1; mask; mask >>= 1)
        putchar(x&mask ? '1' : '0');

    putchar('\n');
}

int sum_of_bits(unsigned x)
{
    int br;
    for (br = 0; x; x>>=1)
        if (x&1)
            br++;

    return br;
}
```

```
main()
{
    printf("Binarni zapis broja 127 je\n");
    print_bits(127);
    printf("Suma bitova broja 127 je %d\n",sum_of_bits(127));
    printf("Binarni zapis broja 128 je\n");
    print_bits(128);
    printf("Suma bitova broja 128 je %d\n",sum_of_bits(128));
    printf("Binarni zapis broja 0x00FF00FF je\n");
    print_bits(0x00FF00FF);
    printf("Suma bitova broja 0x00FF00FF je %d\n",sum_of_bits(0x00FF00FF));
    printf("Binarni zapis broja 0xFFFFFFFF je\n");
    print_bits(0xFFFFFFFF);
    printf("Suma bitova broja 0xFFFFFFFF je %d\n",sum_of_bits(0xFFFFFFFF));
}
```

```
Izlaz iz programa:
Binarni zapis broja 127 je
00000000000000000000000001111111
Suma bitova broja 127 je 7
Binarni zapis broja 128 je
00000000000000000000000010000000
Suma bitova broja 128 je 1
Binarni zapis broja 0x00FF00FF je
00000000111111110000000011111111
Suma bitova broja 0x00FF00FF je 16
Binarni zapis broja 0xFFFFFFFF je
11111111111111111111111111111111
Suma bitova broja 0xFFFFFFFF je 32
```

Primer 4.28.6 *get_bits, set_bits, invert_bits - izdvajanje, postavljanje i invertovanje pojedinacnih bitova*

```
#include <stdio.h>

/* Pomocna funkcija - stampa bitove neoznacenog broja */
void print_bits(unsigned x)
{
    int wl = sizeof(unsigned)*8;

    unsigned mask;
    for (mask = 1<<wl-1; mask; mask >>= 1)
        putchar(x&mask ? '1' : '0');

    putchar('\n');
}

/* Funkcija vraca n bitova broja x koji pocinju na poziciji p */
unsigned get_bits(unsigned x, int p, int n)
{
    /* Gradimo masku koja ima poslednjih n jedinica
```

```
        0000000...00011111
       tako sto sve jedinice ~0 pomerimo u levo za n mesta
            1111111...1100000
       a zatim komplementiramo
    */
    unsigned last_n_1 = ~(~0 << n);

    /* x pomerimo u desno za odgovarajuci broj mesta, a zatim
       konjunkcijom sa konstruisanom maskom obrisemo pocetne cifre */

    return (x >> p+1-n) & last_n_1;
}

/* Funkcija vraca modifikovano x tako sto mu je izmenjeno n bitova
   pocevsi od pozicije p i na ta mesta je upisano poslednjih n bitova
   broja y */
unsigned set_bits(unsigned x, int p, int n, unsigned y)
{
    /* Maska 000000...000111111  - poslednjih n jedinica */
    unsigned last_n_1 = ~(~0 << n);

    /* Maska 1111100..000111111 - n nula pocevsi od pozicije p */
    unsigned middle_n_0 = ~(last_n_1 << p+1-n);

    /* Brisemo n bitova pocevsi od pozicije p */
    x = x & middle_n_0;

    /* Izdvajamo poslednjih n bitova broja y i pomeramo ih
       na poziciju p */
    y = (y & last_n_1) << p+1-n;

    /* Upisujemo bitove broja y u broj x i vracamo rezultat */
    return x | y;
}

/* Invertuje n bitova broja x pocevsi od pozicije p */
unsigned invert_bits(unsigned x, int p, int n)
{
    /* Maska 000000111...1100000 - n jedinica pocevsi od pozicije p */
    unsigned middle_n_1 = ~(~0 << n) << p+1-n;

    /* Invertujemo koristeci ekskluzivnu disjunkciju */
    return x ^ middle_n_1;
}

main()
{
    unsigned x = 0x0AA0AFA0;
    print_bits(x);
```

```
    print_bits(get_bits(x, 15, 8));
    print_bits(set_bits(x, 15, 8, 0xFF));
    print_bits(invert_bits(x, 15, 8));
}

Izlaz iz programa:
00001010101000001010111110100000
00000000000000000000000010101111
00001010101000001111111110100000
00001010101000000101000010100000
```

Primer 4.28.7 *right_rotate_bits, mirror_bits - rotiranje i simetrija bitova.*

```
#include <stdio.h>

/* Pomocna funkcija - stampa bitove neoznacenog broja */
void print_bits(unsigned x)
{
int wl = sizeof(unsigned)*8;

unsigned mask;
for (mask = 1<<wl-1; mask; mask >>= 1)
    putchar(x&mask ? '1' : '0');

putchar('\n');
}

/* Funkcija vrsi rotaciju neoznacenog broja x za n pozicija u desno */
unsigned right_rotate(unsigned x, int n)
{
int i;
int wl = sizeof(unsigned)*8;

/* Postupak se ponavlja n puta */
for (i = 0; i < n; i++)
{
    /* Poslednji bit broja x */
    unsigned last_bit = x & 1;

    /* x pomeramo za jedno mesto u desno */
    x >>= 1;

    /* Zapamceni poslednji bit stavljamo na pocetak broja x*/

    x |= last_bit<<wl-1;
}

return x;
}

/* Funkcija obrce binarni zapis neoznacenog broja x tako sto bitove
```

```
   cita unatrag */
unsigned mirror(unsigned x)
{
int i;
int wl = sizeof(unsigned)*8;

/* Rezultat inicijalizujemo na poslednji bit broja x */
unsigned y = x & 1;

/* Postupak se ponavlja wl-1 puta */
for (i = 1; i<wl; i++)
{
    /* x se pomera u desno za jedno mesto */
    x >>= 1;
    /* rezultat se pomera u levo za jedno mesto */
    y <<= 1;

    /* Poslednji bit broja x upisujemo na poslednje
       mesto rezultata */
    y |= x & 1;
}
return y;
}

main()
{
    unsigned x = 0xFAF0FAF0;
    print_bits(x);
    print_bits(mirror(x));
    print_bits(right_rotate(x, 2));
}

Izlaz iz programa:
11111010111100001111101011110000
00001111010111110000111101011111
00111110101111000011111010111100
```

4.29 Linearna i binarna pretraga

Primer 4.29.1 *Linearna pretraga*

```
#include <stdio.h>

/* Funkcija proverava da li se dati element x nalazi
   u datom nizu celih brojeva.
   Funkcija vraca poziciju u nizu na
   kojoj je x pronadjen
   odnosno -1 ukoliko elementa nema.
*/
int linearna_pretraga(int niz[], int br_elem, int x)
```

```
{
    int i;
    for (i = 0; i<br_elem; i++)
        if (niz[i] == x)
            return i;
        /* nikako else */

    return -1;
}

main()
{
    /* Inicijalizacija niza moguca je
    i na ovaj nacin*/
    int a[] = {4, 3, 2, 6, 7, 9, 11};

    /* Da bi smo odredili koliko clanova
     ima niz mozemo koristiti operator
     sizeof*/
    int br_elem = sizeof(a)/sizeof(int);
    int x;
    int i;
    printf("Unesite broj koji trazimo : ");
    scanf("%d",&x);

    i = linearna_pretraga(a, br_elem, x);
    if (i == -1)
        printf("Element %d nije nadjen\n",x);
    else
        printf("Element %d je nadjen na
            poziciji %d\n",x, i);
}
```

Primer 4.29.2 *Binarna pretraga niza*

```
/* Binarna pretraga niza celih brojeva - iterativna verzija*/
#include <stdio.h>

/* Funkcija proverava da li se element x javlja unutar niza
   celih brojeva a.
   Funkcija vraca poziciju na kojoj je element nadjen odnosno
   -1 ako ga nema.

   !!!!! VAZNO !!!!!
   Pretpostavka je da je niz a uredjen po velicini
*/

int binarna_pretraga(int a[], int n, int x)
{
    /* Pretrazujemo interval [l, d] */
    int l = 0;
```

```
    int d = n-1;

    /* Sve dok interval [l, d] nije prazan */
    while (l <= d)
    {
        /* Srednja pozicija intervala [l, d] */
        int s = (l+d)/2;

        /* Ispitujemo odnos x i a[srednjeg elementa] */
        if (x == a[s])
            /* Element je pronadjen */
            return s;
        else if (x < a[s])
        {
            /* Pretrazujemo interval [l, s-1] */
            d = s-1;
        }
        else
        {
            /* Pretrazujemo interval [s+1, d] */
            l = s+1;
        }
    }

    /* Element je nadjen */
    return -1;
}

main()
{
    int a[] = {3, 5, 7, 9, 11, 13, 15};
    int x;
    int i;

    printf("Unesi element kojega trazimo : ");
    scanf("%d",&x);
    i = binarna_pretraga(a, sizeof(a)/sizeof(int), x);

    if (i==-1)
        printf("Elementa %d nema\n", x);
    else
        printf("Pronadjen na poziciji %d\n", i);
}
```

4.30 Razni zadaci

Primer 4.30.1 *Obrtanje stringa i pretvaranje broja u string*

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

```
/* reverse:  obrce string, npr string "1234" postaje "4321" */
void reverse(char s[])
{
    int c, i, j;

    for (i = 0, j = strlen(s)-1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

/* itoa:  konvertuje broj n u niz karaktera s */
void itoa(int n, char s[])
{
    int i, sign;

    if ((sign = n) < 0)  /* sacuvaj znak */
        n = -n;          /* napravi da je n pozitivno */
    i = 0;
    do {
    /* generisanje cifara u obrnutom smeru */
    s[i++] = n % 10 + '0';  /* izracunaj sledecu cifru */
    } while ((n /= 10) > 0);    /* izbaci cifru iz zapisa */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

main()
{
int n=-44;
char nst[100];
itoa(n,nst);
printf("%s\n",nst);
}
```

Primer 4.30.2 *btoi - konverzija iz datog brojnog sistema u dekadni.*

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

/* Pomocna funkcija koja izracunava vrednost
    koju predstavlja karakter u datoj osnovi
   Funkcija vraca -1 ukoliko cifra nije validna.

   Npr.
   cifra 'B' u osnovi 16 ima vrednost 11
```

```
   cifra '8' nije validna u osnovi 6
*/

int digit_value(char c, int base)
{
/* Proveravamo obicne cifre */
if (isdigit(c) && c < '0'+base)
    return c-'0';

/* Proveravamo slovne cifre za mala slova */
if ('a'<=c && c < 'a'+base-10)
    return c-'a'+10;

/* Proveravamo slovne cifre za velika slova */
if ('A'<=c && c < 'A'+base-10)
    return c-'A'+10;

return -1;
}

/* Funkcija izracunava vrednost celog broja koji je zapisan u datom
   nizu karaktera u datoj osnovi. Za izracunavanje se koristi
   Hornerova shema.
*/
int btoi(char s[], int base)
{
int sum = 0;

/* Obradjuju se karakteri sve dok su to validne cifre */
int i, vr;
for (i = 0; (vr = digit_value(s[i], base)) != -1; i++)
    sum = base*sum + vr;

return sum;
}

main()
{
char bin[] = "11110000";
char hex[] = "FF";

printf("Dekadna vrednost binarnog broja %s je %d\n", bin, btoi(bin, 2));
printf("Dekadna vrednost heksadekadnog
                                broja %s je %d\n", hex, btoi(hex, 16));
}
```

Primer 4.30.3 *Učitava linije sa ulaza i pamti najdužu liniju.*

```
#include <stdio.h>
#define MAXLINE 1000 /* maximum input line length */
```

```
int getline(char line[], int maxline);
void copy(char to[], char from[]);

/* print the longest input line */
main()
{
    int len;   /* current line length */
    int max;   /* maximum length seen so far */
    char line[MAXLINE]; /* current input line */
    char longest[MAXLINE]; /* longest line saved here */

    max = 0;
    while ((len = getline(line, MAXLINE)) > 0)
        if (len > max) {
            max = len;
            copy(longest, line);
        }
    if (max > 0)  /* there was a line */
        printf("%s", longest);
}

/* getline:  read a line into s, return length  */
int getline(char s[],int lim)
{
    int c, i;

    for (i=0; i < lim-1
            && (c=getchar())!=EOF && c!='\n';++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return i;
}

/* copy:  copy 'from' into 'to';
assume to is big enough */
void copy(char to[], char from[])
{
    int i;

    i = 0;
    while ((to[i] = from[i]) != '\0')
        ++i;
}
```

Primer 4.30.4 *Konvertovanje stringa u broj u pokretnom zarezu.*

```
#include <ctype.h>
#include <stdio.h>
#define MAXLINE 100

/* getline:  get line into s, return length */
int getline(char s[], int lim)
{
int c, i;

i = 0;
while (--lim > 0 && (c=getchar()) != EOF && c != '\n')
    s[i++] = c;
if (c == '\n')
    s[i++] = c;
s[i] = '\0';
return i;
}

/* atof:  convert string s to double */
double atof(char s[])
{
double val, power;
int i, sign;

/* skip white space */
for (i = 0; isspace(s[i]); i++)
    ;
/* Postavlja se znak broja*/
sign = (s[i] == '-') ? -1 : 1;

/* Preskace se jedno mesto ukoliko je bio upisan znak u broj*/
if (s[i] == '+' || s[i] == '-')
    i++;

/* Racuna se vrednost broja dok se ne naidje na tacku */
for (val = 0.0; isdigit(s[i]); i++)
    val = 10.0 * val + (s[i] - '0');

if (s[i] == '.')
    i++;

/* Racuna se vrednost broja iza tacke*/
for (power = 1.0; isdigit(s[i]); i++)
    {
    val = 10.0 * val + (s[i] - '0');
    power *= 10;
    }
return sign * val / power;
}
```

```
/* Program sabira brojeve u pokretnom zarezu koji se unose sa ulaza*/
main()
{
double sum;
char line[MAXLINE];

sum = 0;
while (getline(line, MAXLINE) > 0)
    printf("\t%g\n", sum += atof(line));
}
```

Primer 4.30.5 *Funkcija koja uklanja znak* `c` *kad god se pojavi u stringu* `s`.

```
#include <stdio.h>

void squeeze(char s[], char c)
{
int i,j;
for(i=j=0; s[i]!='\0';i++)
    if(s[i]!=c) s[j++]=s[i];
s[j]='\0';
}

main()
{
    char niz[20];
    char c;

    printf("Unesi karakter\n\n");
    scanf("%c", &c);

    scanf("%s", niz);
    squeeze(niz, c);
    printf("%s\n", niz);
}
```

4.31 Sortiranje

Primer 4.31.1 *U prvom prolazu se razmenjuju vrednosti a[0] sa onim članovima ostatka niza koji su veći od njega. Na taj način će se posle prvog prolaza kroz niz a[0] postaviti na najveći element niza.*

```
#include<stdio.h>
#define MAXDUZ 100

int main()
{
   /* Niz od maksimalno MAXDUZ elemenata*/
   int a[MAXDUZ];
```

```
   /* Dimenzija niza, pomocna i brojacke promenljive */
   int n,pom,i,j;

   printf("Unesite dimenziju niza\n");
   scanf("%d",&n);

   if (n>MAXDUZ)
   {
        printf("Nedozvoljena vrednost za n\n");
        exit(1);
   }

   /* Unos clanova niza */
   for(i=0; i<n; i++)
   {
      printf("Unesite %d. clan niza\n",i+1);
      scanf("%d",&a[i]);
   }

   /*Sortiranje*/
   for(i=0; i<n-1; i++)
      for(j=i+1; j<n; j++)
         if(a[i]<a[j])
         {
            pom=a[i];
            a[i]=a[j];
            a[j]=pom;
         }

   /* Ispis niza */
   printf("Sortirani niz:\n");
   for(i=0; i<n; i++)
      printf("%d\t",a[i]);

   putchar('\n');

   return 0;

}
```

Primer 4.31.2 *sort2*
Modifikacija prethodnog rešenja radi dobijanja na efikasnosti. Ne vrše se zamene svaki put već samo jednom, kada se pronađe odgovarajući element u nizu sa kojim treba izvršiti zamenu tako da u nizu bude postavljen trenutno najveći element na odgovarajuće mesto.

```
#include<stdio.h>
#define MAXDUZ 100

int main()
{
```

```
/* Niz od maksimalno MAXDUZ elemenata*/
int a[MAXDUZ];

  /* Dimenzija niza, indeks najveceg elementa
u i-tom prolazu,pomocna i brojacke promenljive */
int n,ind,pom,i,j;

printf("Unesite dimenziju niza\n");
scanf("%d",&n);

if (n>MAXDUZ)
{
     printf("Nedozvoljena vrednost za n\n");
     exit(1);
}

/* Unos clanova niza */
for(i=0; i<n; i++)
{
   printf("Unesite %d. clan niza\n",i+1);
   scanf("%d",&a[i]);
}

/*Sortiranje - bez stalnih zamena vec se
pronalazi indeks trenutno najveceg clana niza*/
for(i=0; i<n-1; i++)
{
    for(ind=i,j=i+1; j<n; j++)
       if(a[ind]<a[j])
          ind=j;

    /* Vrsi se zamena onda kada na i-tom mestu
     nije najveci element. Tada se na i-to mesto
     postavlja najveci element koji se nalazio na
     mestu ind. */
    if(i != ind)
    {
       pom=a[ind];
       a[ind]=a[i];
       a[i]=pom;
    }
}
/* Ispis niza */
printf("Sortirani niz:\n");
for(i=0; i<n; i++)
   printf("%d\t",a[i]);

return 0;
```

```
}
```

Primer 4.31.3 *bbsort1*
Algoritam sortiranja buble sort poredi dva susedna elementa niza i ako su pogrešno raspoređeni zamenjuje im mesta. Posle poređenja svih susednih parova najmanji od njih će isplivati na kraj niza. Zbog toga se ovaj metod naziva metod mehurića. Da bi se najmanji broj nesortiranog dela niza doveo na svoje mesto treba ponoviti postupak.

```
#include<stdio.h>
#define MAXDUZ 100

int main()
{
   /* Dimenzija niza, pomocna promenljiva
   i brojacke promenljive */
   int n,pom,i,j;

   /* Niz od maksimalno MAXDUZ elemenata*/
   int a[MAXDUZ];

   printf("Unesite dimenziju niza\n");
   scanf("%d",&n);

   if (n>MAXDUZ)
   {
        printf("Nedozvoljena vrednost za n\n");
        exit(1);
   }

   /* Unos clanova niza */
   for(i=0; i<n; i++)
   {
      printf("Unesite %d. clan niza\n",i+1);
      scanf("%d",&a[i]);
   }

   /*Sortiranje */
   for(i=n-1; i>0; i--)
      for(j=0; j<i; j++)
         if(a[j]<a[j+1])
         {
            pom=a[j];
            a[j]=a[j+1];
            a[j+1]=pom;
         }

   /* Ispis niza */
   printf("Sortirani niz:\n");
   for(i=0; i<n; i++)
      printf("%d\t",a[i]);
```

```
   /* Stampa prazan red */
   putchar('\n');

   /*Regularan zavrsetak rada programa */
   return 0;

}
```

Primer 4.31.4 *bbsort2*
Unapredjujemo prethodni algoritam kako bismo obezbedli da se ne vrše provere onda kada je niz već sortiran nego da se u tom slučaju prekine rad.

```
#include<stdio.h>
#define MAXDUZ 100

int main()
{
   /* Dimenzija niza, pomocna promenljiva
    i brojacke promenljive */
   int n,pom,i,j;

   /* Niz od maksimalno MAXDUZ elemenata*/
   int a[MAXDUZ];

   /* Promenljiva koja govori da li je izvrsena
   zamena u i-tom prolazu kroz niz pa ako nije
   sortiranje je zavrseno jer su svaka dva
   susedna elementa niza u odgovarajucem poretku */
   int zam;

   printf("Unesite dimenziju niza\n");
   scanf("%d",&n);

   if (n>MAXDUZ)
   {
        printf("Nedozvoljena vrednost za n\n");
        exit(1);
   }

   /* Unos clanova niza */
   for(i=0; i<n; i++)
   {
      printf("Unesite %d. clan niza\n",i+1);
      scanf("%d",&a[i]);
   }

   /*Sortiranje */
   for(zam=1,i=n-1; zam && i>0; i--)
      for(zam=0,j=0; j<i; j++)
         if(a[j]<a[j+1])
```

```
         {
            /* Zamena odgovarajucih clanova niza */
            pom=a[j];
            a[j]=a[j+1];
            a[j+1]=pom;

            /* Posto je u i-tom prolazu
            izvrsena bar ova zamena zam
            se postavlja na 1 sto
            nastavlja sortiranje */
            zam=1;
         }

   /* Ispis niza */
   printf("Sortirani niz:\n");
   for(i=0; i<n; i++)
      printf("%d\t",a[i]);

   return 0;

}
```

Primer 4.31.5 *isort*
Insert sort, u svakom trenutku je početak niza sortiran a sortiranje se vrši tako što se jedan po jedan element niza sa kraja ubacuje na odgovarajuće mesto.

```
#include<stdio.h>
#define MAXDUZ 100

int main()
{
   /* Dimenzija niza, pomocna
   i brojacke promenljive */
   int n,pom,i,j;

   /* Niz od maksimalno MAXDUZ elemenata*/
   int a[MAXDUZ];

   printf("Unesite dimenziju niza\n");
   scanf("%d",&n);

   if (n>MAXDUZ)
   {
        printf("Nedozvoljena vrednost za n!\n");
        exit(1);
   }

   /* Unos clanova niza */
   for(i=0; i<n; i++)
   {
      printf("Unesite %d. clan niza\n",i+1);
```

```
      scanf("%d",&a[i]);
   }

   /*Sortiranje*/
   for(i=1; i<n; i++)
      for(j=i; (j>0) && (a[j]>a[j-1]); j--)
      {
         pom=a[j];
         a[j]=a[j-1];
         a[j-1]=pom;
      }

   /* Ispis niza */
   printf("Sortirani niz:\n");
   for(i=0; i<n; i++)
      printf("%d\t",a[i]);

   putchar('\n');
   return 0;
}
```

Primer 4.31.6 *Binarno pretrazivanje*

```
#include<stdio.h>
#define MAXDUZ 100

int main()
{
   /* Dimenzija niza,pomocna i brojacke
      promenljive */
   int n,pom,i,j;

   /* Niz od maksimalno MAXDUZ elemenata*/
   int a[MAXDUZ];

   /* Elemet koji se trazi i pozicija
   na kojoj se nalazi- ukoliko je u nizu*/
   int x,pozicija;

   /* Pomocne promenljive za pretragu */
   int donji, gornji, srednji;

   printf("Unesite dimenziju niza\n");
   scanf("%d",&n);

   /* Unos clanova niza */
   for(i=0; i<n; i++)
   {
      printf("Unesite %d. clan niza\n",i+1);
      scanf("%d",&a[i]);
   }
```

```
   /*Sortiranje*/
   for(i=0; i<n-1; i++)
      for(j=i+1; j<n; j++)
         if(a[i]>a[j])
         {
            pom=a[i];
            a[i]=a[j];
            a[j]=pom;
         }
   /* Unos elementa binarne pretrage */
   printf("Unesite element koji se trazi\n");
   scanf("%d",&x);

   donji = 0;
   gornji = n-1;
   pozicija = -1;

   while(donji<=gornji)
   {
      srednji = (donji + gornji)/2;
      if(a[srednji] == x)
      {
         pozicija = srednji;
         break;
      }
      else
         if(a[srednji] < x)
            donji = srednji + 1;
         else
            gornji = srednji -1;
   }


   /* Ispis rezultata */
   if(pozicija == -1)
      printf("Trazeni broj se ne nalazi u nizu!\n");
   else
      printf("Broj %d se nalazi na %d poziciji
       sortiranog niza! \n",x,pozicija+1);

   putchar('\n');

   return 0;
}
```

Primer 4.31.7 *Sabiranje dva velika broja, njihovo poređenje, unos i ispis, množenje velikog broja cifrom.*

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

```
#define MAXDUZ 1000

int unos_broja(int cifre[], int maxduz)
   {
     int brcifara=0;
     char c;

     c=getchar();
     while ( brcifara < maxduz && c >= '0' && c <= '9')
     {
        cifre[brcifara++]=c-'0';
        c=getchar();
     }

     return brcifara;
   }

void obrni(int cifre[],int brcifara)
{
   int i,pom;

   for (i=0; i<brcifara/2; i++)
   {
      pom=cifre[i];
      cifre[i]=cifre[brcifara-i-1];
      cifre[brcifara-i-1]=pom;
   }
}

void ispisi(int cifre[],int brcifara)
{  int i;
   putchar('\n');
   for (i=brcifara-1; i>=0; i--)
      printf("%d",cifre[i]);
      /* ili
      putchar(cifre[i]+'0');
      */
    putchar('\n');
}

int jednaki(int cifre1[],int cifre2[],
            int brcifara1, int brcifara2)
{
   int i;
   if (brcifara1 != brcifara2) return 0;

   for (i=0; i<brcifara1; i++)
     if (cifre1[i] != cifre2[i]) return 0;
   return 1;
}
```

```
int veci(int cifre1[], int brcifara1,
        int cifre2[], int brcifara2)
{
   int i;
   if (brcifara1>brcifara2) return 1;
   if (brcifara1<brcifara2) return 0;

   for (i=brcifara1-1; i>=0; i--)
   {
      if (cifre1[i]<cifre2[i]) return 0;
      if (cifre1[i]>cifre2[i]) return 1;
   }

   return 0;
}


int saberi(int cifre1[], int brcifara1,
           int cifre2[], int brcifara2,
           int cifre[])
{
    int brcifara=0;
    int i,pom,pamtim=0;

    for(i=0; i<brcifara1 || i<brcifara2; i++)
    {
        pom =((i < brcifara1)? cifre1[i] : 0 )
            +((i < brcifara2)? cifre2[i] : 0)
            + pamtim;

        cifre[i] = pom%10;
        pamtim = pom/10;
    }
    if (pamtim)
    {
       cifre[i]=pamtim;
       brcifara=i+1;
    }
    else brcifara=i;

    return brcifara;
}

int pomnozic(int c,int cifre[],
             int brcifara, int pcifre[])
{
   int pbrcifara=0;
   int i,pamtim=0;
```

```
   for (i=0; i<brcifara; i++)
   {
      pcifre[i]=(cifre[i]*c+pamtim)%10;
      pamtim=(cifre[i]*c+pamtim)/10;
   }
   pbrcifara=brcifara;
   if (pamtim)
   {
   pcifre[pbrcifara]=pamtim;
   pbrcifara++;
   }

   return pbrcifara;
}


int main()
{
int d1,d2,d;
int broj1[MAXDUZ], broj2[MAXDUZ], zbir[MAXDUZ];
d1=unos_broja(broj1,MAXDUZ);
d2=unos_broja(broj2,MAXDUZ);

obrni(broj1,d1);
obrni(broj2,d2);
d=saberi(broj1,d1,broj2,d2,zbir);
ispisi(zbir,d);
return 0;
}
```

4.32 Rekurzija

C funkcije se mogu rekurzivno koristiti, što znači da funkcija može pozvati samu sebe direktno ili indirektno.

Primer 4.32.1 *Štampanje celog broja.*

```
#include<stdio.h>
void printb(long int n)
{
   if(n<0)
   {
      putchar('-');
      n=-n;
   }
   if(n/10)
      printb(n/10);
   putchar(n % 10 + '0');
}
```

```
int main()
{
long int b=-1234;
printb(b);
putchar('\n');
return 0;
}
```

Kad funkcija rekurzivno pozove sebe, svakim pozivom pojavljuje se novi skup svih automatskih promenljivih, koji je nezavisan od prethodonog skupa. Prva funkcija printb kao argument dobija broj -12345, ona prenosi 1234 u drugu printb funkciju, koja dalje prenosi 123 u treću, i tako redom do poslednje koja prima 1 kao argument. Ta funkcija štampa 1 i završava sa radom tako da se vraća na prethodni nivo, na kome se štampa dva i tako redom.

Primer 4.32.2 *Računanje sume prvih n prirodnih brojeva.*

```
#include<stdio.h>
int suma(int n)
{
   if(n!=0)
      return( n + suma(n-1) );
   else return n;
}

main()
{
int S,n;
printf("Unesite n\n");
scanf("%d",&n);
S=suma(n);
printf("S=%d",S);
putchar('\n');
}
```

Primer 4.32.3 *Računanje faktorijela prirodnog broja.*

```
#include<stdio.h>
unsigned long fakt(int n)
{
   if(n!=0)
      return( n*fakt(n-1) );
   else return 1;
}

main()
{
int n;
unsigned long f;
printf("Unesite n\n");
scanf("%d",&n);
```

```
f=fakt(n);
printf("f=%d",f);
putchar('\n');
}
```

Primer 4.32.4 *Fibonačijevi brojevi.*

```
#include<stdio.h>
int fibr(int n)
{
   if((n==1)||(n==2))
      return 1;
   else return(fibr(n-1)+fibr(n-2));
}

int main()
{
int Fn,n;
printf("Unesite n\n");
scanf("%d",&n);
Fn=fibr(n);
printf("F[%d]=%d",n,Fn);
putchar('\n');
return 0;
}
```

Primer 4.32.5 *Iterativna i rekurzivna varijanta računanja sume niza.*

```
int suma_niza_iterativno(int a[], int n)
{
    int suma = 0;
    int i;
    for (i = 0; i<n;  i++)
        suma+=a[i];
    return suma;
}

int suma_niza(int a[], int n)
{
    if (n == 1)
        return a[0];
    else
        return suma_niza(a, n-1)+a[n-1];
}
```

Primer 4.32.6 *Stepenovanje prirodnog broja*

```
int stepenuj (int n, int k)
{
    if (k == 0)
        return 1;
    else
```

```
        return n*stepenuj(n, k-1);
}
```

4.33 Životni vek i oblast važenja promenjivih, statičke promenljive

Primer 4.33.1 *Demonstracija životnog veka i oblasti važenja promenjivih (scope).*

```
#include <stdio.h>

/* Globalna promenjiva */
int a = 0;

/* Uvecava se globalna promenjiva a */
void increase()
{
    a++;
    printf("increase::a = %d\n", a);
}

/* Umanjuje se lokalna promenjiva a.
Globalna promenjiva zadrzava svoju vrednost. */
void decrease()
{
    /* Ovo a je nezavisna promenjiva u odnosu na globalno a */
    int a = 0;
    a--;
    printf("decrease::a = %d\n", a);

}

void nonstatic_var()
{
    /* Nestaticke promenjive ne cuvaju vrednosti kroz pozive funkcije */
    int s=0;
    s++;
    printf("nonstatic::s=%d\n",s);
}

void static_var()
{
    /* Staticke promenjive cuvaju vrednosti kroz pozive funkcije.
       Inicijalizacija se odvija samo u okviru prvog poziva. */
    static int s=0;
    s++;
    printf("static::s=%d\n",s);
}

main()
```

```
{
    /* Promenjive lokalne za funkciju main */
    int i;
    int x = 3;

    printf("main::x = %d\n", x);


    for (i = 0; i<3; i++)
    {
        /* Promenjiva u okviru bloka je nezavisna od spoljne promenjive.
           Ovde se koristi promenjiva x lokalna za blok petlje koja ima
           vrednost 5, dok originalno x i dalje ima vrednost 3*/
        int x = 5;
        printf("for::x = %d\n", x);
    }

    /* U ovom bloku x ima vrednost 3 */
    printf("main::x = %d\n", x);

    increase();
    decrease();

    /* Globalna promenjiva a */
    printf("main::a = %d\n", a);

    /* Demonstracija nestatickih promenjivih */
    for (i = 0; i<3; i++)
        nonstatic_var();

    /* Demonstracija statickih promenjivih */
    for (i = 0; i<3; i++)
        static_var();
}

Izlaz iz programa:
main::x = 3
for::x = 5
for::x = 5
for::x = 5
main::x = 3
increase::a = 1
decrease::a = -1
main::a = 1
nonstatic::s=1
nonstatic::s=1
nonstatic::s=1
static::s=1
static::s=2
static::s=3
```

Primer 4.33.2 *Ilustracija statičkih promenljivih.*

```
#include <stdio.h>

void f()
{
    static int a;
    a++;
    printf("%d\n",a);
}

main()
{
int i;
   for (i=0; i<=10; i++)
          f();
}

/* Izlaz iz programa 1 2 3 4 5 6 7 8 9 10 11*/
```

4.34 Pokazivači

Pokazivač je promenljiva koja sadrži adresu promenljive.

```
int x=1, y=1, z[10];
int *ip;    /* ip je pokazivac na int,
               odnosno *ip je tipa int*/

ip = &x;    /* ip sada pokazuje na x */
y=*ip;      /* y je sada 1 */
*ip = 0;    /* x je sada 0 */

*ip+=10;    /* x je sada 10*/
++*ip;      /* x je sada 11*/
(*ip)++;    /* x je sada 12,
            zagrada neophodna zbog prioriteta
            operatora*/

ip = &z[0]; /* ip sada pokazuje na z[0]*/
```

Primer 4.34.1 *Ilustracija rada sa pokazivačkim promenljivim.*

```
#include <stdio.h>
main()
{
int x = 3;

/* Adresu promenjive x zapamticemo u novoj promeljivoj.
    Nova promenljiva je tipa pokazivaca na int (int*) */
int* px;
```

```
printf("Adresa promenljive x je : %p\n", &x);
printf("Vrednost promenljive x je : %d\n", x);

px = &x;
printf("Vrednost promenljive px je (tj. px) : %p\n", px);
printf("Vrednost promenljive na koju ukazuje px (tj. *px) je : %d\n", *px);

/* Menjamo vrednost promenljive na koju ukazuje px */
*px = 6;
printf("Vrednost promenljive na koju ukazuje px (tj. *px) je : %d\n", *px);

/* Posto px sadrzi adresu promenljive x, ona ukazuje na x tako da je
    posredno promenjena i vrednost promenljive x */
printf("Vrednost promenljive x je : %d\n", x);

}

Izlaz (u konkretnom slucaju):
Adresa promenljive x je : 0012FF88
Vrednost promenljive x je : 3
Vrednost promenljive px je (tj. px) : 0012FF88
Vrednost promenljive na koju ukazuje px (tj. *px) je : 3
Vrednost promenljive na koju ukazuje px (tj. *px) je : 6
Vrednost promenljive x je : 6
```

Pored pokazivača na osnovne tipove, postoji i pokazivač na prazan tip (void).

```
void *pp;
```

Njemu može da se dodeli da pokazuje na int, ili na char ili na proizvoljan tip ali je to neophodno eksplicitno naglasiti svaki put kada želimo da koristimo ono na šta on pokazuje.

Primer 4.34.2 *Upotreba pokazivača na prazan tip.*

```
#include<stdio.h>

main()
{
void *pp;
int x=2;
char c='a';

pp = &x;
*(int *)pp = 17;    /* x postaje 17*/
printf("\n adresa od x je %p", &x);
printf("\n%d i %p",*(int*)pp,(int * )pp);

pp = &c;
printf("\n adresa od c je %p", &c);
```

```
printf("\n%c i %p",*(char*)pp,(char * )pp);

}

/*
 adresa od x je 0012FF78
17 i 0012FF78
 adresa od c je 0012FF74
a i 0012FF74
*/
```

Posebna konstanta koja se koristi da se označi da pokazivač ne pokazuje na neko mesto u memoriji je NULL.

4.35 Pokazivači i argumenti funkcija

C prosleđuje argumente u funkcije pomoću vrednosti. To znači da sledeća funkcija neće uraditi ono što želimo:

```
void swap (int x, int y) /* POGRESNO!!!!!!!!*/
{
int temp;
temp = x;
x=y;
y=temp;
}
```

Zbog prenosa parametara preko vrednosti swap ne može da utiče na argumente a i b u funkciji koja je pozvala swap. Ova swap funkcija samo zamenjuje kopije od a i b.

Da bi se dobio željeni efekat, potrebno je da se proslede pokazivači:

```
/* Zameni *px i *py */
void swap (int *px, int *py)
{
int temp;
temp =*px;
*px = *py;
*py = temp;
}
```

a poziv funkcije swap izlgeda sada ovako

```
swap(&a, &b);
```

Primer 4.35.1 *Demonstracija više povratnih vrednosti funkcije koristeći prenos preko pokazivača.*

```
/* Funkcija istovremeno vraca dve vrednosti - kolicnik i ostatak
dva data broja. Ovo se postize tako sto se funkciji predaju
vrednosti dva broja (x i y) koji se dele
```

```
i adrese dve promenljive na koje ce se smestiti rezultati */

void div_and_mod(int x, int y, int* pdiv, int* pmod)
{
    printf("Kolicnik postavljam na adresu : %p\n", pdiv);
    printf("Ostatak postavljam na adresu : %p\n", pmod);
    *pdiv = x / y;
    *pmod = x % y;
}

main()
{
    int div, mod;
    printf("Adresa promenljive div je %p\n", &div);
    printf("Adresa promenljive mod je %p\n", &mod);

    /* Pozivamo funkciju tako sto joj saljemo
       vrednosti dva broja (5 i 2)
       i adrese promenljvih div i mod na koje
       ce se postaviti rezultati */
    div_and_mod(5, 2, &div, &mod);

    printf("Vrednost promenljive div je %d\n", div);
    printf("Vrednost promenljive mod je %d\n", mod);

}

Izlaz u konkretnom slucaju:
Adresa promenljive div je 0012FF88
Adresa promenljive mod je 0012FF84
Kolicnik postavljam na adresu : 0012FF88
Ostatak postavljam na adresu : 0012FF84
Vrednost promenljive div je 2
Vrednost promenljive mod je 1
```

4.36 Pokazivači i nizovi (polja)

U C-u postoji čvrsta veza između pokazivača i nizova. Bilo koja operacija koja se može ostvariti dopisivanjem indeksa niza može se uraditi i sa pokazivačima.

Deklaracija

`int a[10];`

definiše niz `a` veličine 10 koji predstavlja blok od 10 uzastopnih objekata nazvanih `a[0]`, `a[1]`, ..., `a[9]`. Notacija `a[i]` odgovara i-tom elementu niza.

Ako je `pa` pokazivač na ceo broj

`int *pa;`

tada iskaz `pa = &a[0];`

podešava da pa pokaže na nulti element niza `a`, odnosno `pa` sadrži adresu od `a[0]`.

Ako `pa` pokazuje na određeni element polja, onda po definiciji `pa+1` pokazuje na sledeći element, `pa+i` pokazuje na i-ti element posle `pa`. Stoga, ako `pa` pokazuje na `a[0]` tada

`*(pa+1)`

se odnosi na sadržaj od `a[1]`.

`pa+i` je adresa od `a[i]`, a

`*(pa+i)`

je sadržaj od `a[i]`. Dozvoljeno je upotrebljavati i sintaksu kao kod nizova:

`*(pa+i) <==> pa[i]`

Ovo sve važi bez obzira na tip ili veličinu elemenata u polju `a`.

Iskaz `pa=&a[0]` se može napisati kao **pa=a jer je ime niza sinonim za lokaciju početnog elementa.**

Primer 4.36.1 *Veza između pokazivača i nizova.*

```
#include <stdio.h>

void print_array(int* pa, int n);

main()
{
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int num_of_elements = sizeof(a)/sizeof(int);
int* pa;

/* Niz je isto sto i adresa prvog elementa */
printf("Niz a : %p\n", a);
printf("Adresa prvog elementa niza a (&a[0]) : %p\n", &a[0]);
/* Niz a : 0012FF5C
Adresa prvog elementa niza a (&a[0]) : 0012FF5C */

/* Moguce je dodeliti niz pokazivacu odgovarajuceg tipa */
pa = a;

printf("Pokazivac pa ukazuje na adresu : %p\n", pa);
/* Pokazivac pa ukazuje na adresu : 0012FF5C */

/* Nizu nije moguce dodeliti pokazivacku promenljivu
   (nizove mozemo smatrati KONSTANTNIM pokazivacima na prvi element) */
/* a = pa; */

/* Pokazivace je dalje moguce indeksirati kao nizove */
printf("pa[0] = %d\n", pa[0]);
printf("pa[5] = %d\n", pa[5]);
/* pa[0] = 1
   pa[5] = 6 */

/* Medjutim, sizeof(pa) je samo velicina pokazivaca, a ne niza */
printf("sizeof(a) = %d\n", sizeof(a));
printf("sizeof(pa) = %d\n", sizeof(pa));
```

```
/* sizeof(a) = 40
   sizeof(pa) = 4 */

/* Pozivamo funkciju za stampanje niza i saljemo joj niz */
print_array(a, num_of_elements);
/* 1 2 3 4 5 6 7 8 9 10 */

/* Pozivamo funkciju za stampanje niza
  i saljemo joj pokazivac na pocetak niza */
print_array(pa, num_of_elements);
/* 1 2 3 4 5 6 7 8 9 10 */
}

/* Prosledjivanje niza u funkciju
   void print_array(int pa[], int n);
   je ekvivalentno prosledjivanju pokazivaca u funkciju
   void print_array(int* pa, int n);
   Izmedju ovih konstrukcija nema nikakve razlike!
*/
void print_array(int* pa, int n)
{
    int i;
    for (i = 0; i<n; i++)
        printf("%d ", pa[i]);
    putchar('\n');
}
```

Prilikom deklaracije treba praviti razliku između niza znakova i pokazivača:

```
char poruka[]="danas je lep dan!";
char *pporuka = "danas je lep dan!";
```

`poruka` je niz znakova koji sadrži dati tekst. Pojedine znake moguće je promeniti ali se poruka uvek odnosi na isto mesto u memoriji.
`pporuka` je pokazivač, koji je inicijalizovan da pokazuje na konstantnu nisku, on može biti preusmeren da pokazuje na nešto drugo, ali rezultat neće biti definisan ako pokušate da modifikujete sadržaj niske (jer je to konstantna niska).

Ako deklarišemo

```
char *pporuka1 = "danas je lep dan!";
char *pporuka2 = "danas je lep dan!";
char *pporuka3 = "danas pada kisa";
```

tada će pokazivači pporuka1 i pporuka2 pokazivati na isto mesto u memoriji, a pporuka3 na neko drugo mesto u memoriji.

Ako uporedimo (pporuka1==pporuka3) uporediće se vrednosti pokazivača. Ako uporedimo (pporuka1 < pporuka2) uporediće se vrednosti pokazivača. Ako dodelimo pporuka1=pporuka3 tada će pporuka1 dobiti vrednost pokazivača pporuka3 i pokazivaće na isto mesto u memoriji. Neće se izvršiti kopiranje sadržaja memorije!!!

Primer 4.36.2 *Vežba pokazivačke aritmetike.*

```
#include <stdio.h>

/* Funkcija pronalazi x u nizu niz date dimenzije,
   bez koriscenja indeksiranja. Funkcija vraca pokazivac na
   poziciju pronadjenog elementa. */

int* nadjiint(int* niz, int n, int x)
{
    while (--n >= 0 && *niz!=x)
                niz++;

      return (n>=0)? niz: NULL;
}

main()
{
    int a[]={1,2,3,4,5,6,7,8};
    int* poz=nadjiint(a,sizeof(a)/sizeof(int),4);

    if (poz!=NULL)
        printf("Element pronadjen na poziciji %d\n",poz-a);
}
```

Primer 4.36.3

```
int strlen(char *s)
{
    int n;
    for(n=0; *s != '\0'; s++) n++;
    return n;
}
```

Primer 4.36.4

```
/* Funkcija kopira string t u string s */
void copy(char* s, char* t)
{
while (*s++=*t++)
              ;
}

/* Ovo je bio skraceni zapis za sledeci kod
    while(*t != '\0')
        {
        *s=*t;
        s++;
        t++;
        }
    *s = '\0';


*/
```

Primer 4.36.5

```
/* Vrsi leksikografsko poredjenje dva stringa.
    Vraca :
        0 - ukoliko su stringovi jednaki
        <0 - ukoliko je s leksikografski ispred t
        >0 - ukoliko je s leksikografski iza t
*/
int string_compare1(char *s, char *t)
{
    /* Petlja tece sve dok ne naidjemo
    na prvi razliciti karakter */
    for (; *s == *t; s++, t++)
        if (*s == '\0') /* Naisli smo na kraj
                           oba stringa, a nismo nasli razliku */
            return 0;


    /* *s i *t su prvi karakteri u kojima
       se niske razlikuju.
       Na osnovu njihovog odnosa,
       odredjuje se odnos stringova */

    return *s - *t;
}

/* Mozemo koristiti i sintaksu kao kod nizova */
int string_compare2(char *s, char *t) {
    int i;
    for (i = 0; s[i] == t[i]; i++)
        if (s[i] == '\0')
            return 0;
    return s[i] - t[i];
}
```

Primer 4.36.6 *Pronalazi prvu poziciju karaktera c u stringu s, i vraća pokazivač na nju, odnosno NULL ukoliko s ne sadrži c.*

```
char* string_char(char *s, char c)
{
    int i;
    for (; *s; s++)
        if (*s == c)
            return s;

    /* Nije nadjeno */
    return NULL;
}
```

Primer 4.36.7 *Pronalazi poslednju poziciju karaktera c u stringu s, i vraća pokazivač na nju, odnosno NULL ukoliko s ne sadrži c.*

```
char* string_last_char(char *s, char c)
{
    char *t = s;
    /* Pronalazimo kraj stringa s */
    while (*t++)
        ;

    /* Krecemo od kraja i trazimo c unazad */
    for (t--; t >= s; t--)
        if (*t == c)
            return t;

    /* Nije nadjeno */
    return NULL;
}
```

Primer 4.36.8

```
#include <stdio.h>

/* proverava da li se niska t nalazi unutar niske s*/
int sadrzi_string(char s[], char t[])
{
    int i;
    for (i = 0; s[i]; i++)
    {
        int j;
        for (j=0, k=0; s[i+j]==t[j]; j++)
            if (t[j+1]=='\0')
                return i;
    }
    return -1;
}

/* Verzija funkcije strstr implementirane bez
   koriscenja indeksiranja */
/* proverava da li se niska t nalazi unutar niske s*/
char* sadrzi_string_pok(char* s, char* t)
{
    while(*s)
    {
        char *i, *j;
        for (i = s, j = t; *i == *j; i++,j++)
            if (*(j+1)=='\0')
                return s;
        s++;
    }
    return NULL;
}

/* Cita liniju sa stadnardnog ulaza i
```

```
   vraca njenu duzinu */
int getline(char* line, int max)
{
    char *s=line;
    int c;
    while ( max-->0 && (c=getchar())!='\n' && c!=EOF)
        *s++ = c;

    if (c=='\n')
        *s++ = c;

    *s = '\0';
    return s - line;
}


main()
{
    char rec[]="zdravo";
    char linija[100];
    while (getline(linija, 100))
        if (sadrzi_string_pok(linija, rec))
            printf("%s",linija);
}
```

4.37 Alokacija memorije

`void* malloc(size_t n)` vraća pokazivač na n bajtova neinicijalizovane memorije ili `NULL` ukoliko zahtev ne može da se ispuni.

Za njeno korišćenje neophodno je uključiti zaglavlje `stdlib.h`. Oslobađanje memorije - funkcija `free`.

Ne sme se koristiti nešto što je već oslobođeno, ne sme se dva puta oslobađati ista memorija.

Primer 4.37.1

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

main()
{
int n;
int i;
int *a;

printf("Unesi broj clanova niza : ");
scanf("%d", &n);

/* Kao da ste mogli da uradite
```

```
    int a[n];
*/
a = (int*)malloc(n*sizeof(int));

/* Kad god se vrsi alokacija memorije mora se proveriti da li je ona
   uspesno izvrsena!!! */
if (a == NULL)
{
    printf("Nema slobodne memorije\n");
    exit(1);
}

/* Od ovog trenutka a koristim kao obican niz */
for (i = 0; i<n; i++)
    scanf("%d",&a[i]);

/* Stampamo niz u obrnutom redosledu */
for(i = n-1; i>=0; i--)
    printf("%d",a[i]);

/* Oslobadjamo memoriju*/
free(a);
}
```

Primer 4.37.2 *Demonstracija funkcije* `calloc` *- funkcija inicijalizuje sadrzaj memorije na 0.*

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

main()
{
int *m, *c, i, n;

printf("Unesi broj clanova niza : ");
scanf("%d", &n);

/* Niz m NE MORA garantovano da ima sve nule */
m = malloc(n*sizeof(int));
if (m == NULL) {
                printf("Greska prilikom alokacije memorije!\n");
                exit(1);
               }

/* Niz c MORA garantovano da ima sve nule */
c = calloc(n, sizeof(int));
if (c == NULL) {
                printf("Greska prilikom alokacije memorije!\n");
                free(m);
                exit(1);
                }
```

```
for (i = 0; i<n; i++)
    printf("m[%d] = %d\n", i, m[i]);

for (i = 0; i<n; i++)
    printf("c[%d] = %d\n", i, c[i]);

free(m);
free(c);
}
```

4.38 Niz pokazivača

Primer 4.38.1

```
#include <stdio.h>
#include <stdlib.h>
main()
{
/* Niz od tri elemenata tipa int*/
int nizi[3];

/* Niz od tri elemenata tipa int*, dakle
  niz od tri pokazivaca na int*/
int* nizip[3];

/* Alociramo memoriju za prvi element niza*/
nizip[0] = (int*) malloc(sizeof(int));
if (nizip[0] == NULL)
{
    printf("Nema slobodne memorije\n");
    exit(1);
}
/* Upisujemo u prvi element niza broj 5*/
*nizip[0] = 5;
printf("%d", *nizip[0]);

/* Alociramo memoriju za drugi element niza.
   Drugi element niza pokazuje na niz od dva
   elementa*/
nizip[1] = (int*) malloc(2*sizeof(int));
if (nizip[1] == NULL) {
    printf("Nema slobodne memorije\n");
    free(nizip[0]);
    exit(1);
}

/* Pristupamo prvom elementu na koji pokazuje
   pokazivac nizip[1]*/
*(nizip[1]) = 1;
```

```
/* Pristupamo sledecem elementu u nizu na koji pokazuje
   nizip[1].
*/
*(nizip[1] + 1) = 2;

printf("%d", nizip[1][1]);

/* Alociramo memoriju za treci element niza nizip. */
nizip[2] = (int*) malloc(sizeof(int));
if (nizip[2] == NULL) {
    printf("Nema slobodne memorije\n");
    free(nizip[0]);
    free(nizip[1]);
    exit(1);
}

*(nizip[2]) = 2;

printf("%d", *(nizip[2]));

free(nizip[0]);
free(nizip[1]);
free(nizip[2]);
}
```

Primer 4.38.2

```
#include <stdio.h>
#include <stdlib.h>
main()
{
/* Niz karaktera*/
char nizc[5];

/* Niz karaktera od cetiri elementa
  ('A', 'n', 'a', '\0')*/
char nizcc[]="Ana";
printf("%s", nizcc);

/* Niz od tri pokazivaca. Prvi pokazuje na
   nisku karaktera Kruska, drugi na nisku karaktera
   Sljiva a treci na Ananas. */
char* nizcp[]={"Kruska", "Sljiva", "Ananas"};

printf("%s", nizcp[0]);
printf("%s", nizcp[1]);
printf("%s", nizcp[2]);
}
```

4.39 Pokazivači na funkcije

Primer 4.39.1 *Program demonstrira upotrebu pokazivača na funkcije.*

```
#include <stdio.h>

int kvadrat(int n)
{
return n*n;
}

int kub(int n)
{
return n*n*n;
}

int parni_broj(int n)
{
return 2*n;
}

/* Funkcija izracunava sumu od 1 do n f(i),
   gde je f data funkcija */
int sumiraj(int (*f) (int), int n)
{
    int i, suma=0;
    for (i=1; i<=n; i++)
        suma += (*f)(i);

    return suma;
}

main()
{
printf("Suma kvadrata brojeva od jedan do 3 je %d\n", sumiraj(kvadrat,3));
printf("Suma kubova brojeva od jedan do 3 je %d\n", sumiraj(kub,3));
printf("Suma prvih pet parnih brojeva je %d\n", sumiraj(parni_broj,5));
}
/*Izlaz:
Suma kvadrata brojeva od jedan do 3 je 14
Suma kubova brojeva od jedan do 3 je 36
Suma prvih pet parnih brojeva je 30
*/
```

4.40 Matrice

Primer 4.40.1 *Statička alokacija prostora za matricu.*

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

```
main()
{
int a[3][3] = {{0, 1, 2}, {10, 11, 12}, {20, 21, 22}};
int i, j;

/* Alternativni unos elemenata matrice
for(i=0; i<3; i++)
    for(j=0; j<3; j++)
        {
        printf("a[%d][%d] = ", i, j);
        scanf("%d", &a[i][j]);
        }
*/

a[1][1] = a[0][0] + a[2][2];
/* a[1][1] = 0 + 22 = 22 */

printf("%d\n", a[1][1]);   /* 22 */

/* Stampanje elemenata matrice*/
for(i=0; i<3; i++)
    {
    for(j=0; j<3; j++)
        printf("%d\t", a[i][j]);
    printf("\n");
    }
}
```

Nama je potrebno da imamo veću fleksibilnost, tj da se dimenzije matrice mogu uneti kao parametri našeg programa. Zbog toga je neophodno koristiti dinamicku alokaciju memorije.

Primer 4.40.2 *Implementacija matrice preko niza.*

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

/* Makro pristupa clanu na poziciji i, j matrice koja ima
   m vrsta i n kolona */
#define a(i,j) a[(i)*n+(j)]

main()
{
   /* Dimenzije matrice */
   int m, n;

   /* Matrica */
   int *a;

   int i,j;
```

```
   /* Suma elemenata matrice */
   int s=0;

   /* Unos i alokacija */
   printf("Unesi broj vrsta matrice : ");
   scanf("%d",&m);

   printf("Unesi broj kolona matrice : ");
   scanf("%d",&n);

   a=malloc(m*n*sizeof(int));
   if (a == NULL) {
                   printf("Greska prilikom alokacije memorije!\n");
                   exit(1);
                   }

   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
      {
      printf("Unesi element na poziciji (%d,%d) : ",i,j);
      scanf("%d",&a(i,j));
      }

   /* Racunamo sumu elemenata matrice */
   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
            s+=a(i,j);

   /* Ispis unete matrice */
   printf("Uneli ste matricu : \n");
   for (i=0; i<m; i++)
   {   for (j=0; j<n; j++)
          printf("%d ",a(i,j));
       printf("\n");
   }

   printf("Suma elemenata matrice je %d\n", s);

   /* Oslobadjamo memoriju */
   free(a);
}
```

Primer 4.40.3 *Program ilustruje rad sa kvadratnim matricama i relacijama. Elementi i je u relaciji sa elementom j ako je m[i][j] = 1, a nisu u relaciji ako je m[i][j] = 0.*

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

/* Dinamicka matrica je odredjena adresom
   pocetka niza pokazivaca i dimenzijama tj.
```

```
   int** a;
   int m,n;
*/

/* Alokacija kvadratne matrice nxn */
int** alociraj(int n)
{
    int** m;
    int i;
    m=malloc(n*sizeof(int*));
    if (m == NULL) {
        printf("Greska prilikom alokacije memorije!\n");
        exit(1);
        }

    for (i=0; i<n; i++)
        {
        m[i]=malloc(n*sizeof(int));
        if (m[i] == NULL)
        {
        int k;
        printf("Greska prilikom alokacije memorije!\n");
        for(k=0;k<i;k++)
                free(m[k]);
        free(m);
        exit(1);
        }
        }

    return m;
}

/* Dealokacija matrice dimenzije nxn */
void obrisi(int** m, int n)
{
    int i;
    for (i=0; i<n; i++)
       free(m[i]);
    free(m);
}

/* Ispis matrice /
void ispisi_matricu(int** m, int n)
{
     int i, j;
      for (i=0; i<n; i++)
      {
         for (j=0; j<n; j++)
             printf("%d ",m[i][j]);
         printf("\n");
```

```
      }
}

/* Provera da li je relacija predstavljena matricom refleksivna */
int refleksivna(int** m, int n)
{
    int i;
    for (i=0; i<n; i++)
       if (m[i][i]==0)
           return 0;

    return 1;
}

/* Provera da li je relacija predstavljena matricom simetricna */
int simetricna(int** m, int n)
{
    int i,j;
    for (i=0; i<n; i++)
      for (j=i+1; j<n; j++)
         if (m[i][j]!=m[j][i])
             return 0;
    return 1;
}

/* Provera da li je relacija predstavljena matricom tranzitivna*/
int tranzitivna(int** m, int n)
{
    int i,j,k;

    for (i=0; i<n; i++)
       for (j=0; j<n; j++)
           for (k=0; k<n; k++)
               if ((m[i][j]==1)
                   && (m[j][k]==1)
                   && (m[i][k]!=1))
                     return 0;
    return 1;
}

/* Pronalazi najmanju simetricnu relaciju koja sadrzi relaciju a */
void simetricno_zatvorenje(int** a, int n)
{
   int i,j;
   for (i=0; i<n; i++)
     for (j=0; j<n; j++)
     {
        if (a[i][j]==1 && a[j][i]==0)
                  a[j][i]=1;
        if (a[i][j]==0 && a[j][i]==1)
```

```
                    a[i][j]=1;
     }
}

main()
{
       int **m;
       int n;
       int i,j;

       printf("Unesi dimenziju matrice : ");
       scanf("%d",&n);
       m=alociraj(n);

       for (i=0; i<n; i++)
          for (j=0; j<n; j++)
             scanf("%d",&m[i][j]);

       printf("Uneli ste matricu : \n");

       ispisi_matricu(m,n);

       if (refleksivna(m,n))
            printf("Relacija je refleksivna\n");
       if (simetricna(m,n))
            printf("Relacija je simetricna\n");
       if (tranzitivna(m,n))
            printf("Relacija je tranzitivna\n");

       simetricno_zatvorenje(m,n);

       ispisi_matricu(m,n);

       obrisi(m,n);
}
```

Primer 4.40.4 *Izračunati vrednost determinante matrice preko Laplasovog razvoja.*

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

/* Funkcija alocira matricu dimenzije nxn */
int** allocate(int n)
{
     int **m;
     int i;
     m=(int**)malloc(n*sizeof(int*));
     if (m == NULL) {
        printf("Greska prilikom alokacije memorije!\n");
        exit(1);
        }
```

```
    for (i=0; i<n; i++)
        {
        m[i]=malloc(n*sizeof(int));
        if (m[i] == NULL)
         {
         int k;
         for(k=0;k<i;k++)
                free(m[k]);
         printf("Greska prilikom alokacije memorije!\n");
         free(m);
         exit(1);
         }
        }

     return m;
}

/* Funkcija vrsi dealociranje date matrice dimenzije n */
void deallocate(int** m, int n)
{
     int i;
     for (i=0; i<n; i++)
        free(m[i]);
     free(m);
}

/* Funkcija ucitava datu alociranu matricu sa standardnog ulaza */
void ucitaj_matricu(int** matrica, int n)
{
     int i,j;
     for (i=0; i<n; i++)
        for (j=0; j<n; j++)
           scanf("%d",&matrica[i][j]);
}

/* Rekurzivna funkcija koja vrsi Laplasov razvoj */
int determinanta(int** matrica, int n)
{
int i;
int** podmatrica;
int det=0,znak;

/* Izlaz iz rekurzije je matrica 1x1 */
if (n==1) return matrica[0][0];

/* Podmatrica ce da sadrzi minore polazne matrice */
podmatrica=allocate(n-1);
znak=1;
for (i=0; i<n; i++)
```

```
{
int vrsta,kolona;
for (kolona=0; kolona<i; kolona++)
    for(vrsta=1; vrsta<n; vrsta++)
        podmatrica[vrsta-1][kolona] = matrica[vrsta][kolona];
for (kolona=i+1; kolona<n; kolona++)
    for(vrsta=1; vrsta<n; vrsta++)
        podmatrica[vrsta-1][kolona-1] = matrica[vrsta][kolona];

det+= znak*matrica[0][i]*determinanta(podmatrica,n-1);
znak*=-1;
}
deallocate(podmatrica,n-1);
return det;
}

main()
{
   int **matrica;
   int n;

   scanf("%d", &n);
   matrica = allocate(n);
   ucitaj_matricu(matrica, n);
   printf("Determinanta je : %d\n",determinanta(matrica,n));
   deallocate(matrica, n);
}
```

4.41 Strukture

Informacije kojima se opisuje realni svet retko se predstavljaju u elementarnoj formi u vidu celih, realnih, znakovnih konstanti itd. Mnogo češće imamo posla sa složenim objektima koji se sastoje od elemenata raznih tipova. Na primer jednu osobu karakterišu ime, prezime, datum i mesto rođenja.

Struktura predstavlja skup podataka kojim se opisuju neka bitna svojstva objekta. Komponente koje obrazuju strukturu nazivaju se elementi strukture.

Sintaksa strukture:

```
struct ime_strukture
{
tip ime_elementa1;
tip ime_elementa2;
...
};
```

Primer 4.41.1 *Primer jednostavne strukture.*

```
struct licnost
{
char ime[31];
```

```
char adresa[41];
unsigned starost;
};


struct licnost osoba1, osoba2;
```

Deklaraciju osobe1 i osobe2 mogli smo da zapišemo i na sledeći način

```
struct licnost
{
char ime[31];
char adresa[41];
unsigned starost;
} osoba1, osoba2;
```

Ukoliko nemamo potrebu da se ličnost koristi dalje u programu mogu se napraviti dve osobe bez davanja imena strukturi:

```
struct
{
char ime[31];
char adresa[41];
unsigned starost;
} osoba1, osoba2;
```

Kada imamo promenljivu strukturnog tipa tada elementima date strukture pristupamo uz pomoc operatora '.'.

Primer 4.41.2

```
osoba1.starost=20;
osoba2.starost=21;
...
if (osoba1.starost == osoba2.starost)
     printf(" Osobe su iste starosti");
```

Dozvoljeno je praviti nizove struktura. Npr. niz od 20 elemenata koji sadrži ličnosti:

```
struct licnost nizLicnosti[20];
```

Tada da bi pročitali starost neke ličnosti u nizu pišemo:

```
nizLicnosti[5].starost
```

Može se definisati pokazivač na strukturu.

```
struct licnost *posoba;
```

Tada se pristupanje elementima strukture može vršiti upotrebom operatora '.' na standardni način:

```
(*posoba).ime
(*posoba).adresa
(*posoba).starost
```

ili korišćenjem specijalnog operatora $'->'$ na sledeći način:

```
posoba->ime
posoba->adresa
posoba->starost
```

Primer 4.41.3 *Elementi strukture mogu da budu i druge strukture.*

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

struct licnost
{
char ime[30];
struct datum datumrodjenja;
};
```

Sada se danu, mesecu i godini datuma rodjenja pristupa na sledeći način:

```
osoba.datumrodjenja.dan = 10;
osoba.datumrodjenja.mesec = 5;
osoba.datumrodjenja.godina = 1986;
```

4.41.1 Operator typedef

Operator typedef omogućava nam da definišemo naša imena za neki od osnovih ili izvedenih tipova. Na primer, možemo da uradimo sledeće:

```
typedef double RealanBroj;
```

Nakon ovoga možemo u tekstu deklarista promenljivu x kao RealanBroj, ona će zapravo biti tipa double.

```
RealanBroj x; /* Umesto: double x;*/
```

Ili, ako želimo da skratimo pisanje za neoznačene duge brojeve tj za unsigned long int to možemo da uradimo na sledeći način

```
typedef unsigned long int VelikiBroj;
```

Sada u kodu možemo da koristimo VelikiBroj kao tip.

Operator typedef je naročito pogodan da bi se izbeglo ponavljalnje reči struct pri deklarisanju strukturnih promenljivih.

```
typedef struct _licnost licnost;
```

Sada deklaracija može da bude:

```
licnost osoba1, osoba2;
/* umesto: struct _licnost osoba1, osoba2; */
```

Kao skraćen zapis za

```
struct _tacka {
    float x;
    float y;
}

typedef struct _tacka tacka;
```

može se koristiti:

```
typedef struct _tacka
{
    float x;
    float y;
} tacka;
```

Primer 4.41.4 *Struktura artikal.*

```
typedef struct _artikal
{
    long bar_kod;
    char ime[MAX_IME];
    float pdv;
} artikal;
```

Primer 4.41.5 *Program ilustruje osnovne geometrijske algoritme kao i rad sa strukturama.*

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

typedef struct _tacka
{
    float x;
    float y;
} tacka;

typedef struct _vektor
{
   float x, y;
} vektor;

/* Koordinatni pocetak */
tacka kp={0.0,0.0};

/* Niz tacaka */
tacka niz[100];

/* Pokazivac na strukturu tacke */
tacka *pt;
```

```
void IspisiTacku(tacka A)
{
   printf("(%f,%f)\n",A.x,A.y);
}

void IspisiVektor(vektor v)
{
   printf("(%f,%f)\n",v.x,v.y);
}

float duzina(vektor v)
{
   return sqrt(v.x*v.x+v.y*v.y);
}

vektor NapraviVektor(tacka *pA, tacka *pB)
{
     vektor ab;
     ab.x=pB->x - pA->x;
     ab.y=pB->y - pA->y;
     return ab;
}

float rastojanje(tacka A, tacka B)
{
    float dx=B.x - A.x;
    float dy=B.y - A.y;
    return sqrt(dx*dx+dy*dy);
}

/* Heronov obrazac */
float PovrsinaTrougla(tacka A, tacka B, tacka C)
{
    float a=rastojanje(B,C);
    float b=rastojanje(A,C);
    float c=rastojanje(A,B);

    float s=(a+b+c)/2.0;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}

float PovrsinaKonveksnogPoligona(tacka poligon[], int br_temena)
{
int i;
   float povrsina=0.0;
   for (i=1; i<br_temena-1; i++)
      povrsina+=PovrsinaTrougla(poligon[0], poligon[i], poligon[i+1]);

   return povrsina;
}
```

```
float Obim(tacka poligon[], int br_temena)
{
int i;
float o=0;

for (i=0; i<br_temena-1; i++)
    o+=rastojanje(poligon[i], poligon[i+1]);

o+=rastojanje(poligon[0], poligon[br_temena-1]);
return o;
}

main()
{
    tacka poligon[]={{0.0,0.0},
                     {0.0,1.0},
                     {1.0,1.0},
                     {1.0,0.0}};
    printf("Obim poligona je %f\n",Obim(poligon,4));
    printf("Povrsina poligona je %f\n",
              PovrsinaKonveksnogPoligona(poligon,4));
}
```

Primer 4.41.6 *Program koji učitava niz studenata i sortira ih po njihovim ocenama.*

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

#define MAX_IME  20

typedef struct _student
{
   char ime[MAX_IME];
   char prezime[MAX_IME];
   int ocena;
} student;

/* Funkcija ucitava rec i vraca njenu duzinu ili
   -1 ukoliko smo dosli do znaka EOF*/
int getword(char word[],int max)
{
    int c, i=0;

    while (isspace(c=getchar()))
         ;

    while(!isspace(c) && c!=EOF && i<max-1)
    {
          word[i++]=c;
```

```
          c = getchar();
    }

    word[i]='\0';

    if (c==EOF) return -1;
        else return i;

}

/* Funkcija ucitava niz studenata, vraca duzinu niza koji ucita */
int UcitajPodatke(student studenti[], int max)
{
int i=0;
while(i<max && getword(studenti[i].ime, MAX_IME)>0)
{
if (getword(studenti[i].prezime, MAX_IME) < 0)
        break;
scanf("%d",&studenti[i].ocena);
i++;
}
return i;
}

void IspisiPodatke(student studenti[], int br_studenata)
{
int i;
printf("IME          PREZIME          OCENA\n");
printf("---------------------------------------\n");
for (i=0; i<br_studenata; i++)
    printf("%-20s %-20s %5d\n",studenti[i].
        ime, studenti[i].prezime, studenti[i].ocena);
}

/* Sortiranje studenata po ocenama */
void SelectionSort(student studenti[], int br_studenata)
{
    int i,j;
    for (i=0; i<br_studenata-1; i++)
       for (j=i; j<br_studenata; j++)
           if (studenti[i].ocena<studenti[j].ocena)
           {    student tmp=studenti[i];
                studenti[i]=studenti[j];
                studenti[j]=tmp;
           }
}

main()
{
student studenti[100];
```

```
int br_studenata = UcitajPodatke(studenti,100);

SelectionSort(studenti, br_studenata);

IspisiPodatke(studenti, br_studenata);

}
```

Primer 4.41.7 *Dinamički niz.*

```
/* Program za svaku rec unetu sa standardnog
   ulaza ispisuje broj pojavljivanja.
   Verzija sa dinamickim nizom i realokacijom.
*/

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Rec je opisana imenom i brojem pojavljivanja */
typedef struct _rec
{   char ime[80];
    int br_pojavljivanja;
} rec;

/* Dinamicki niz reci je opisan pokazivacem na
   pocetak, tekucim brojem upisanih elemenata i
   tekucim brojem alociranih elemenata */
rec* niz_reci;
int duzina=0;
int alocirano=0;

/* Realokacija se vrsi sa datim korakom */
#define KORAK 10

/* Funkcija ucitava rec i vraca njenu duzinu ili
   -1 ukoliko smo dosli do znaka EOF*/
int getword(char word[],int max)
{
    int c, i=0;

    while (isspace(c=getchar()))
         ;

    while(!isspace(c) && c!=EOF && i<max-1)
    {
          word[i++]=c;
          c = getchar();
    }
```

```
    word[i]='\0';

    if (c==EOF) return -1;
        else return i;
}

main()
{
char procitana_rec[80];
int i;

while (getword(procitana_rec,80)!=-1)
{
   /* Proveravamo da li rec vec postoji u nizu */
   for (i=0; i<duzina; i++)
        /* Ako bismo uporedili procitana_rec == niz_reci[i].ime
        bili bi uporedjeni pokazivaci a ne odgovarajuci sadrzaji!!!
        Zato koristimo strcmp. */
        if (strcmp(procitana_rec, niz_reci[i].ime)==0)
        {
        niz_reci[i].br_pojavljivanja++;
        break;
        }

    /* Ukoliko rec ne postoji u nizu */
    if (i==duzina)
    {
        rec nova_rec;
        /* Ako bismo dodelili nova_rec.ime = procitana_rec
        izvrsila bi se dodela pokazivaca a ne kopiranje niske
        procitana_rec u nova_rec.ime. Zato koristimo strcpy!!! */
        strcpy(nova_rec.ime,procitana_rec);
        nova_rec.br_pojavljivanja=1;

        /* Ukoliko je niz "kompletno popunjen" vrsimo realokaciju */
        if (duzina==alocirano)
        {
        alocirano+=KORAK;

        /* Sledeca linija zamenjuje blok koji sledi i moze se
           koristiti alternativno. Blok je ostavljen samo da bi
           demonstrirao korisnu tehniku */
        /* niz_reci=realloc(niz_reci, (alocirano)*sizeof(rec)); */
        {
        /* alociramo novi niz, veci nego sto je bio prethodni */
        rec* novi_niz=(rec *)malloc(alocirano*sizeof(rec));

        if (novi_niz == NULL)
        {
            free(niz_reci);
```

```
            printf("Greska prilikom alokacije memorije");
            exit(1);
        }

        /* Kopiramo elemente starog niza u novi */
        for (i=0; i<duzina; i++)
                novi_niz[i]=niz_reci[i];

        /* Uklanjamo stari niz */
        free(niz_reci);

        /* Stari niz postaje novi */
        niz_reci=novi_niz;
        }
    }
    /* Upisujemo rec u niz */
    niz_reci[duzina]=nova_rec;
    duzina++;
    }
}

/* Ispisujemo elemente niza */
for(i=0; i<duzina; i++)
 printf("%s - %d\n",niz_reci[i].ime, niz_reci[i].br_pojavljivanja);

free(niz_reci);
}
```

4.42 qsort

Primer 4.42.1 *Implementacija funkcije qsort.*

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

void printarray(int v[], int left, int right)
{
    int i;
    for (i=left; i<=right; i++)
            printf("%d  ",v[i]);
    putchar('\n');
}

void swap(int v[], int i, int j)
{
    int tmp=v[i];
    v[i]=v[j];
    v[j]=tmp;
}
```

```
/* qsort:  sortira v[left]...v[right] u rastucem poretku */
void qsort(int v[], int left, int right)
{
    int i, last;

    /* ne radi nista ako niz sadrzi  */
    /* manje od dva elementa */
    if (left >= right)
        return;
    /* prebaci element particioniranja */
    /* u v[left] */
    swap(v, left, (left + right)/2);
    last = left;

    /* partition */
    for (i = left + 1; i <= right; i++)
        if (v[i] < v[left])
            swap(v, ++last, i);

    /* restore partition  elem */
    swap(v, left, last);

    /* Sortiraj preostala dva dela niza */
    qsort(v, left, last-1);
    qsort(v, last+1, right);
}

main()
{
    int array[]={8, 3, 2, 6, 5, 7, 4, 9, 1};
    int n=sizeof(array)/sizeof(int);

    printarray(array, 0, n-1);
    qsort(array, 0, n-1);
    printarray(array, 0, n-1);

}
```

4.43 Sortiranje — generička funkcija

Sortiranje niza celih brojeva (jedan od algoritama)

```
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        if(a[i]<a[j])
        {
        int pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
```

Sortiranje iz programa mozemo da izdvojimo u funkciju:

```
void sort_int(int a[], int n)
{
int i, j;
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        if(a[i]<a[j])
        {
        int pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
}
```

Sortiranje niza realnih brojeva:

```
void sort_float(float a[], int n)
{
int i, j;
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        if(a[i]<a[j])
        {
        float pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
}
```

Razlike:

- prvi argument funkcije;
- pomoćna promenljiva;
- poređenje.

Sortiranje studenata po oceni ukoliko je data struktura student:

```
typedef struct _student {
   char ime[MAX_IME];
   char prezime[MAX_IME];
   int ocena;
} student;

void sort_po_oceni(student a[], int n)
{
int i, j;
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        if(a[i].ocena < a[j].ocena)
```

```
        {
        student pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
}
```

Sortiranje studenta po prezimenu:

```
void sort_po_prezimenu(student a[], int n)
{
int i, j;
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        if(strcmp(a[i].prezime, a[j].prezime)<0)
        {
        student pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
}
```

Sortiranje studenta po imenu:

```
void sort_po_imenu(student a[], int n)
{
int i, j;
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        if(strcmp(a[i].ime, a[j].ime)<0)
        {
        student pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
}
```

Kako da napravimo jednu funkciju koja sortira studente bez obzira na kriterijum?

Prvo moramo da izdvojimo funkciju poređenja:

```
int poredi_po_oceni(student st1, student st2)
{
return st1.ocena - st2.ocena;
}

int poredi_po_prezimenu(student st1, student st2)
{
return strcmp(st1.prezime, s2.prezime);
}

int poredi_po_imenu(student st1, student st2)
```

```
{
return strcmp(st1.ime, st2.ime);
}
```

Funkcija poređenja vraća 0 ukoliko su elementi jednaki, broj manji od nule ukoliko je prvi manji od drugog i broj veći od nule ukolikoje prvi veći od drugog.

```
void sort_po_imenu(student a[], int n)
{
int i, j;
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        /*if(poredi_po_prezimenu(a[i], a[j])<0)*/
        /*if(poredi_po_oceni(a[i], a[j])<0)*/
        if(poredi_po_imenu(a[i], a[j])<0)
        {
        student pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
}
```

Sada možemo da dodamo još jedan argument funkciji sortiranja i tako da dobijemo jednu funkciju umesto tri:

```
void sort_studente(student a[], int n, int (*f)(student, student))
{
int i, j;
for(i=0; i<n-1; i++)
    for(j=i+1; j<n; j++)
        if((*f)(a[i], a[j])<0)
        {
        student pom=a[i];
        a[i]=a[j];
        a[j]=pom;
        }
}
```

Šta dalje? Kako da dobijemo jednu funkciju sortiranja bez obzira na tip elemenata niza?

Teba da rešimo sledeće stvari:

- razmena mesta elemenata ne sme da zavisi od tipa elemenata koji se razmenjuju.
- potpis funkcije poređenja ne sme da zavisi od tipa elemenata koji se porede kako bi on bio jedinstven.
- prvi argument funkcije ne sme da zavisi od tipa elemenata niza.

Da bi smo razmenili dva elementa potrebna nam je pomoćna promenljiva u kojoj privremeno čuvamo neku vrednost. Ako ne znamo tip elementa onda ne možemo da napravimo pomoćnu promenljivu. Ali zato mozemo da koristeći

funkciju malloc odvojimo neko mesto u memoriji za smestanje elementa koji nam u datoj situaciji treba. Koliko je to mesto? Nekada 4 bajta, npr za int, a nekada dosta veće, npr za studenta. Kako funkcija sortiranja zna koliko mesta treba da odvoji? Znace tako sto ćemo joj tu veličinu proslediti kao argument. Sada, dakle umesto pomoćne promenljive, imamo blok u memoriji, a umesto naredbe dodele koristićemo funkciju `memcpy` koja kopira deo memorije sa jednog mesta na drugo mesto.

Dakle, razmenu ćemo da radimo na sledeći način:

```
void* tmp = malloc(size);
if (tmp==NULL) {printf("Greska prilikom alokacije memorije!\n");exit(1);}
memcpy(tmp, adresa_itog, size);
memcpy(adresa_itog, adresa_jtog, size);
memcpy(adresa_jtog, tmp, size);
free(tmp);
```

Potpis funkcije poređenja ne sme da zavisi od tipa elemenata koji se porede. To se može postići koristeći pokazivač na tip void.

Na primer, poređenje dva cela broja:

```
int poredi_br(void* a, void* b)
{
    int br_a = *(int*)a;
    int br_b = *(int*)b;

    return br_a-br_b;
}
```

Na primer, poređenje dva realna broja:

```
int poredi_br(void* a, void* b)
{
    float br_a = *(float*)a;
    float br_b = *(float*)b;

    if (br_a > br_b) return 1;
    else if (br_a < br_b) return -1;
    else return 0;
}
```

Na primer, poređenje dva studenta po oceni

```
int poredi_br(void* a, void* b)
{
    student student1 = *(studnet*)a;
    student student2 = *(studnet*)b;

    return student1.ocena-student2.ocena;
}
```

Sada funkcija poređenja ima uvek potpis

```
int poredi(void* a, void* b)
```

i može se kao parametar proslediti našoj funkciji sortiranja.

Primer 4.43.1
```
/* Genericka funkcija sortiranja -
   nezavisna od tipa elemenata niza
   koji se sortira */
#include <stdlib.h>

void sort(void* a, int n, int size, int (*poredi)(void*, void*))
{
int i, j;
for (i = 0; i<n-1; i++)
    for (j = i+1; j<n; j++)
    {
        void* adresa_itog = (char*)a+i*size;
        void* adresa_jtog = (char*)a+j*size;

        if (poredi(adresa_itog, adresa_jtog)<0)
        {
            void* tmp = malloc(size);
            if (tmp==NULL) {
            printf("Greska prilikom alokacije memorije!\n");
            exit(1);
            }
            memcpy(tmp, adresa_itog, size);
            memcpy(adresa_itog, adresa_jtog, size);
            memcpy(adresa_jtog, tmp, size);
            free(tmp);
        }

    }
}


int poredi_br(void* a, void* b) {
    int br_a = *(int*)a;
    int br_b = *(int*)b;

    return br_a-br_b;
}

int poredi_float(void* a, void* b) {
    float br_a = *(float*)a;
    float br_b = *(float*)b;

    if (br_a > br_b) return 1;
    else if (br_a < br_b) return -1;
    else return 0;
}
```

```
main() {
    int a[] = {8, 2, 1, 9, 3, 7, 6, 4, 5};
    float b[] = {0.3, 2, 5, 5.8, 8}
    int n = sizeof(a)/sizeof(int);
    int nf = sizeof(b)/sizeof(float);
    int i;

    sort(a, n, sizeof(int), poredi_br);

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    putchar('\n');

    sort(b, nf, sizeof(float), poredi_float);

    for (i = 0; i < n; i++)
        printf("%f ", b[i]);
    putchar('\n');
}
```

4.44 qSort funkcija iz standardne biblioteke

Primer 4.44.1 *qSort-Upotreba.*

```
/* Ilustracija upotrebe funkcije qsort iz stdlib.h
   Sortira se niz celih brojeva.
*/

#include <stdlib.h>
#include <stdio.h>

/* const znaci da ono na sta pokazuje a (odnosno b)
   nece biti menjano u funkciji */
int poredi(const void* a, const void* b)
{
/* Skracen zapis za
    int br_a = *(int*)a;
    int br_b = *(int*)b;

    return br_a-br_b;
*/
return *((int*)a)-*((int*)b);
}

int poredi_float(const void* a, const void* b)
{
    float br_a = *(float*)a;
    float br_b = *(float*)b;
```

```
    if (br_a > br_b) return 1;
    else if (br_a < br_b) return -1;
    else return 0;
}


main()
{
int i;
int niz[]={3,8,7,1,2,3,5,6,9};
float nizf[]={3.0,8.7,7.8,1.9,2.1,3.3,6.6,9.9};

int n=sizeof(niz)/sizeof(int);
qsort((void*)niz, n, sizeof(int), poredi);
for(i=0; i<n; i++)
            printf("%d",niz[i]);

n=sizeof(nizf)/sizeof(float);
qsort((void*)nizf, n, sizeof(float), poredi_float);
for(i=0; i<n; i++)
            printf("%f",nizf[i]);

}
```

Primer 4.44.2 *Binarno pretraživanje - korišćenje ugrađene bsearch funkcije.*

```
/* Funkcija ilustruje koriscenje ugradjene funkcije bsearch */
#include <stdlib.h>

int poredi(const void* a, const void *b)
{
return *(int*)a-*(int*)b;
}

main()
{
int x=-1;
int niz[]={1,2,3,4,5,6,7,8,9,10,11,12};

int* elem=(int*)bsearch((void*)&x,
                        (void*)niz,
                        sizeof(niz)/sizeof(int),
                        sizeof(int),
                        poredi);

if (elem==NULL)
    printf("Element nije pronadjen\n");
else
    printf("Element postoji na poziciji %d\n",elem-niz);
}
```

4.45 Generičko sortiranje reči

Primer 4.45.1 *Sortiranje reči. Ako se sortira niz stringova, onda svaki element je sam po sebi pokazivač tipa* `char *`*, te funkcija poređenja tada prima podatke tipa* `char **` *koji se konvertuju u svoj tip i derefenciraju radi dobijanja podataka tipa* `char *`*.*

```
/* Ilustracija upotrebe funkcije qsort iz stdlib.h
   Sortira se niz reci i to ili leksikografski
   ili po duzini
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int poredi(const void* a, const void* b)
{
char *s1 = *(char **)a;
char *s2 = *(char **) b;
return strcmp(s1, s2);

/* Prethodno je ekvivalentno sa:
return strcmp(*(char**)a,*(char**)b); */
}

int poredi_po_duzini(const void* a, const void* b)
{
char *s1 = *(char **) a;
char *s2 = *(char **) b;
return strlen(s1) - strlen(s2);
/* Prethodno je ekvivalentno sa:
return strlen(*(char**)b)-strlen(*(char**)a); */
}

main()
{
int i;
char* nizreci[]={"Jabuka","Kruska","Sljiva","Dinja","Lubenica"};

qsort((void*)nizreci, 5, sizeof(char*), poredi_po_duzini);

for (i=0; i<5; i++)
    printf("%s\n",nizreci[i]);

qsort((void*)nizreci, 5, sizeof(char*), poredi);

for (i=0; i<5; i++)
    printf("%s\n",nizreci[i]);
```

```
}
```

Primer 4.45.2 *Sa ulaza se unose reči. Program broji pojavljivanja svake od ključnih reči programskog jezika C. Na kraju se reči ispisuju opadajuće po broju pojavljivanja.*

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

/* Svaka kljucna rec se odlikuje imenom i brojem pojavljivanja */
typedef struct _keyword
{
     char* word;
     int num;
} keyword;

/* Kreiramo niz struktura sortiranih leksikografski
   po imenu kljucne reci, kako bismo ubrzali pronalazak reci */
keyword keywords[]={ {"break",0},
                     {"continue",0},
                     {"float",0},
                     {"for",0},
                     {"if",0},
                     {"return",0},
                     {"struct",0},
                     {"while",0}
                   };

/* Funkcija cita sledecu rec sa standardnog ulaza */
int getword(char word[], int lim)
{
    int c, i=0;
    while(!isalpha(c=getchar()) && c!=EOF)
                ;
    if (c==EOF)
        return -1;
    do
    {
      word[i++]=c;
    } while(--lim>0 && isalpha(c=getchar()));

    word[i]='\0';
    return i;
}

/* Funkcija leksikografskog poredjenja za bsearch */
int cmp(const void* a, const void* b) {
/* Funkcija strcmp prima kao argumente dva
   pokazivaca na karaktere. Prvi je rec koju
   trazimo u nizu, a drugi je element niza
   sa kojim se vrsi poredjenje. Pokazivac
```

```
   b konvertujemo u pokazivac na strukturu
   keyword a zatim posmatramo rec koja se tu
   cuva */
return strcmp((char*)a, (*(keyword*)b).word); }

/* Funkcija numerickog poredjenja za qsort */
int numcmp(const void* a, const void* b)
{
    return ((*(keyword*)b).num-(*(keyword*)a).num);
}

main()
{
    char word[80];
    int i;

    /* Broj kljucnih reci */
    int num_of_keywords=sizeof(keywords)/sizeof(keyword);

    /* Citamo reci */
    while (getword(word,80)!=-1)
    {
    /* Trazimo rec u spisku kljucnih reci binarnom pretragom */
    keyword* k=(keyword*)bsearch((void*)word,
                                 (void*)keywords,
                                 num_of_keywords,
                                 sizeof(keyword),
                                 cmp);
        /* Ukoliko je pronadjena uvecavamo broj pojavljivanja */
        if (k!=NULL)
           (*k).num++;
    }

    /* Sortiramo niz na osnovu broja pojavljivanja */
    qsort((void*)keywords, num_of_keywords, sizeof(keyword), numcmp);

    /* Vrsimo ispis */
    for (i=0; i<num_of_keywords; i++)
        printf("%s %d\n", keywords[i].word, keywords[i].num);
}
```

4.46 Argumenti komandne linije

Primer 4.46.1

```
/* Argumenti komandne linije programa */

/* Program pozivati sa npr.:
    a.out
    a.out prvi
```

```
    a.out prvi drugi treci
    a.out -a -bc ime.txt
*/

#include <stdio.h>

/* Imena ovih promenljivih mogu biti proizvoljna.
   Npr:
   main (int br_argumenata, char* argumenti[]);
   ipak, uobicajeno je da se koriste sledeca imena:
*/

main(int argc, char* argv[])
{
    int i;
    printf("argc = %d\n", argc);

    /* Nulti argument uvek je ime programa (a.out)*/
    for (i = 0; i<argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);
}
```

4.47 Datoteke

Primer 4.47.1 *Program demonstrira otvaranje datoteka (*`"r"` *- read i* `"w"` *- write mod) i osnovne tehnike rada sa datotekama.*

```
/* U datoteku se upisuje prvih 10 prirodnih
   brojeva, a zatim se iz iste datoteke
   citaju brojevi dok se ne stigne do kraja i
   ispisuju se na standardni izlaz */

#include <stdio.h>
#include <stdlib.h>

main()
{
int i;

/* Otvaramo datoteku sa imenom podaci.txt za pisanje */
FILE* f = fopen("podaci.txt", "w");

/* Ukoliko otvaranje nije uspelo, fopen vraca NULL.
   U tom slucaju, prijavljujemo gresku i zavrsavamo program */
if (f == NULL)
{
  printf("Greska prilikom otvaranja datoteke podaci.txt za pisanje\n");
  exit(1);
}
```

```
/* Upisujemo u datoteku prvih 10 prirodnih brojeva
   (svaki u posebnom redu) */
for (i = 0; i<10; i++)
    fprintf(f, "%d\n", i);

/* Zatvaramo datoteku */
fclose(f);

/* Otvaramo datoteku sa imenom podaci.txt za citanje */
f = fopen("podaci.txt", "r");

/* Ukoliko otvaranje nije uspelo, fopen vraca NULL.
   U tom slucaju, prijavljujemo gresku i zavrsavamo program */
if (f == NULL) {
  printf("Greska prilikom otvaranja datoteke podaci.txt za citanje\n");
  exit(1);
}

/* Citamo brojeve iz datoteke dok ne stignemo do kraja i ispisujemo ih
   na standardni izlaz */
while(1) {
    int br;
    /* Pokusavamo da procitamo broj */
    fscanf(f, "%d", &br);

    /* Ukoliko smo dosli do kraja datoteke, prekidamo */
    if (feof(f))
        break;

    /* Ispisujemo procitani broj */
    printf("Procitano : %d\n", br);
}

/* Funkciju feof ne treba pozivati pre pokusaja citanja.
   Sledeci kod moze dovesti do greske:
    while (!feof(f))
        fscanf(f,"%d",&br);
*/

/* Zatvaramo datoteku */
fclose(f);
}
```

Pokazivači `stdin`, `stdout` i `stderr` su definisani u okviru `stdio.h`.

```
FILE* stdin;
FILE* stdout;
FILE* stderr;
```

Primer 4.47.2 *Program demonstrira "a" - append mod datoteka - nadovezivanje.*

```
#include <stdio.h>

main()
{
FILE* datoteka;

/* Otvaramo datoteku za nadovezivanje
   i proveravamo da li je doslo do greske */
if ( (datoteka=fopen("dat.txt","a"))==NULL)
    {
    fprintf(stderr,"Greska prilikom otvaranja dat.txt\n");
    return 1;
    }

/* Upisujemo sadrzaj  u datoteku */
fprintf(datoteka,"Zdravo svima\n");

/* Zatvaramo datoteku */
fclose(datoteka);
}
```

Primer 4.47.3 *Program ilustruje rad sa datotekama. Program kopira datoteku čije se ime zadaje kao prvi argument komandne linije u datoteku čije se ime zadaje kao drugi argument komandne linije. Uz svaku liniju se zapisuje i njen redni broj.*

```
#include <stdio.h>

#define MAX_LINE 256

/* Funkcija fgets definisana je u stdio.h

   char* fgets(char *s, int n, FILE* stream)

   fgets ucitava najvise sledecih n-1 znakova
   u polje niza karaktera s, zaustavljajuci se
   ako naidje na novu liniju koju takodje
   upisuje u polje. Na kraju upisuje '\0'.
   Funkcija vraca s ili NULL ako dodje do kraja
   datoteke ili se pojavi greska

   Funkcija getline moze jednostavno da se
   realizuje preko funkcije fgets.

int getline(char s[], int lim)
{
    char* c = fgets(s, lim, stdin);
    return c==NULL ? 0 : strlen(s);
}
*/
```

```
main(int argc, char* argv[])
{

char line[MAX_LINE];
FILE *in, *out;
int line_num;

if (argc != 3) {
    fprintf(stderr,"Upotreba : %s ulazna_datoteka izlazna_datoteka\n",argv[0]);
    return 1;
}

if ((in = fopen(argv[1],"r")) == NULL)
{
    fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
    return 1;
}

if ((out = fopen(argv[2],"w")) == NULL)
{
    fprintf(stderr, "Neuspesno otvaranje datoteke %s\n",argv[2]);
    return 1;
}

/*  Prepisivanje karakter po karakter je moguce ostvariti preko:
    int c;
    while ((c=fgetc(in)) != EOF)
        putc(c,out);
*/

line_num = 1;

/* Citamo liniju po liniju sa ulaza*/
while (fgets(line, MAX_LINE, in) != NULL)
{
    /* Ispisujemo broj linije i sadrzaj linije na izlaz */
    fprintf(out, "%d :\t", line_num++);
    fputs(line, out);
}

/* Zatvaramo datoteke */
fclose(in);
fclose(out);
}
```

Primer 4.47.4 *Prodavnica - ilustruje čitanje niza struktura iz tektsualne datoteke.*

```
/* Datoteka, cije se ime zadaje kao argument komandne linije
   ili ako se ne zada onda se ime unosi sa standardnog
   ulaza, sadrzi podatke o proizvodima koji se prodaju
```

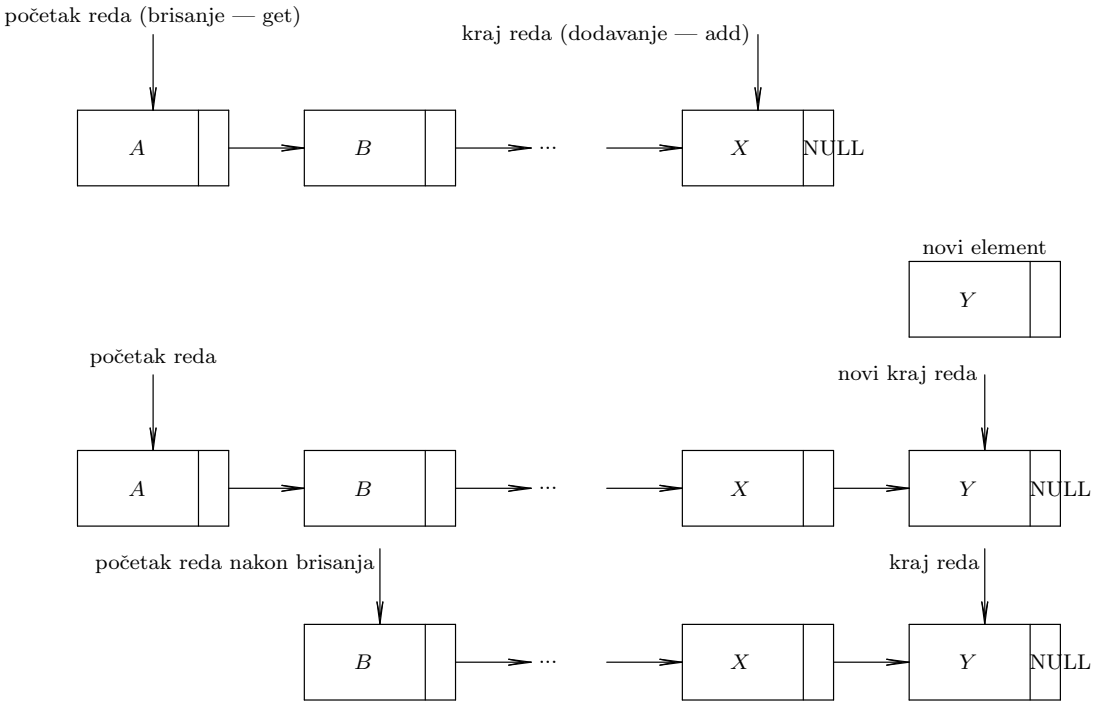
```
   u okviru odredjene prodavnice. Svaki
   proizvod se odlikuje sledecim podacima:
    bar-kod   - petocifreni pozitivan broj
    ime       - niska karaktera
    cena      - realan broj zaokruzen na dve decimale
    pdv       - stopa poreza - realan broj zaokruzen na dve decimale
   Pretpostavljamo da su podaci u datoteci
   korektno zadati.


   Pretpostavljamo da se u prodavnici ne
   prodaje vise od 1000 razlicitih artikala
   Na standardni izlaz ispisati podatke o
   svim proizvodima koji se prodaju u prodavnici.
   Zadatak je moguce resiti i bez koriscenja nizova
   (i takvo resenje je bolje). */

#include <stdio.h>

/* Maksimalna duzina imena proizvoda */
#define MAX_IME 30

/* Maksimalni broj artikala */
#define MAX_ARTIKALA 1000

/* Struktura za cuvanje podataka o jednom artiklu */
typedef struct _artikal {
    int bar_kod;
    char ime[MAX_IME];
    float cena;
    float pdv;
} artikal;

/* Niz struktura u kome se cuvaju podaci o artiklima */
artikal artikli[MAX_ARTIKALA];

/* Broj trenutno ucitanih artikala */
int br_artikala = 0;

/* Ucitava podatke o jednom artiklu iz date datoteke.
   Vraca da li su podaci uspesno procitani */
int ucitaj_artikal(FILE* f, artikal* a)
{
/* Citamo bar kod, ime, cenu, pdv */
fscanf(f, "%d%s%f%f", &(a->bar_kod), a->ime, &(a->cena), &(a->pdv));

/* Ukoliko smo dosli do kraja datoteke prilikom pokusaja ucitavanja
   prijavljujemo neuspeh */
if (feof(f))
    return 0;
```

```
/* Prijavljujemo uspeh */
return 1;
}

/* Izracunava ukupnu cenu datog artikla */
float cena(artikal a)
{
    return  a.cena*(1+a.pdv);
}

/* Ispisuje podatke o svim artiklima */
void ispisi_artikle()
{
int i;
for (i = 0; i<br_artikala; i++)
    printf("%-5d  %-10s %.2f %.2f    = %.2f\n",
        artikli[i].bar_kod, artikli[i].ime,
        artikli[i].cena, artikli[i].pdv,
        cena(artikli[i]));
}

main(int argc, char* argv[])
{
FILE* f;

/* Ukoliko nije navedeno ime kao argument komandne linije, trazimo
   od korisnika da ga unese */
if (argc<2) {
    /* Ucitavamo ime datoteke */
    char ime_datoteke[256];
    printf("U kojoj datoteci se nalaze podaci o proizvodima: ");
    scanf("%s", ime_datoteke);

    /* Otvaramo datoteku i proveravamo da li smo uspeli */
    if ( (f = fopen(ime_datoteke, "r")) == NULL)
    {
        printf("Greska prilikom otvaranja datoteke %s\n", ime_datoteke);
        return 1;
    }
}
/* Ime datoteke je prvi argument komandne linije */
else {
    /* Otvaramo datoteku i proveravamo da li smo uspeli */
    if ( (f = fopen(argv[1], "r")) == NULL)
    {
        printf("Greska : datoteka %s ne moze biti otvorena\n", argv[1]);
        return 1;
    }
```

```
}

/* Ucitavamo artikle */
while (ucitaj_artikal(f, &artikli[br_artikala]))
    br_artikala++;

/* Ispisujemo podatke o svim artiklima */
ispisi_artikle();

/* Zatvara se datoteka*/
fclose(f);
}
```

4.48 Liste

4.48.1 Red



Slika 4.1: Red

Primer 4.48.1 *Program formira listu brojeva i demonstrira osnovne elemente rada sa listom.*

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

```
#include <stdlib.h>

typedef struct elem {
   int broj;
   struct elem *sled;
} Elem;

/* Citanje brojeva i formiranje liste. Funkcija vraca pokazivac
   na pocetak liste. */
Elem *citaj (void)
{
Elem *lista = NULL, *poslednji = NULL, *novi;
int broj;

printf("\n\nUnesite elemente liste - 0 za kraj\n");
scanf ("%d", &broj);

while (broj) {
    /*Alocira se prostor za novi clan liste*/
    novi =(Elem*)malloc (sizeof (Elem));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }

    /* Postavljanje vrednosti */
    novi->broj = broj;
    novi->sled = NULL;

    /* Ukoliko lista nije prazna novi element se postavlja iza
       poslednjeg elemnta liste - na koji pokazuje poslednji */
    if (poslednji != NULL)
        poslednji->sled = novi;
    /* Inace se postavlja da bude pocetak liste */
    else
        lista = novi;

    /*Poslednji se postavlja da pokazuje na poslednji element liste */
    /*Ekvivalentno je sa
      poslednji = poslednji ->sledeci */
    poslednji = novi;
    /* Ucitavanje novog elementa liste 0 za kraj */
    scanf ("%d", &broj);
}
/* Funkcija vraca pokazivac na pocetak liste */
return lista;
}

/* Ispisivanje liste*/
```

```
void pisi (Elem *lista)
{
  while (lista != NULL)
  {
     printf ("%d ", lista->broj),
     lista = lista->sled;
  }
  putchar ('\n');
}

/* Oslobadjanje memorije koju lista zauzima*/
void brisi (Elem* lista)
{
  Elem *stari;
  while (lista != NULL)
  {
     stari = lista;
     lista = lista->sled;
     free (stari);
  }
}

/* Izbacivanje (brisanje) zadatog broja iz liste. Kako se
   ovime pocetak liste moze izmeniti, vrednost pocetka liste
   je povratna vrednost funkcije */
Elem* izbaci(Elem *lista, int k)
{
   Elem *preth = NULL, *tekuci = lista, *zaizbacivanje;

   while (tekuci != NULL)
      /* Ukoliko tekuci pokazuje na clana
         liste kojeg treba izbaciti */
      if (tekuci->broj == k)
      {
         zaizbacivanje = tekuci;
         tekuci = tekuci->sled;

         if (preth != NULL )
            preth->sled = tekuci;
         /* Ovaj slucaj odnosi se na izbacivanje
            elementa sa pocetka liste */
         else lista = tekuci;

        free (zaizbacivanje);
      }
      else
      {
      preth = tekuci;
      tekuci = tekuci->sled;
      }
```

```
  return lista;
}

main ()
{
Elem *lista;
int k;

lista = citaj ();
printf ("Ucitani lista   = ");
pisi (lista);

printf("Koji element liste zelite da izbacite?\n");
scanf ("%d", &k);
printf ("Izostavlja se = %d\n", k);
printf ("Novi lista      = ");
pisi (lista = izbaci (lista, k));
printf ("\n");

printf("Lista mi vise nije potrebna, oslobadjam memoriju!\n");
brisi (lista);
}
```

Primer 4.48.2 *Rad sa listama - celine izdvojene u funkcije.*

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

typedef struct cvor {
    int br;
    struct cvor* sl;
} CVOR;

/* Pomocna funkcija koja kreira cvor liste sa datim sadrzajem.
   Funkcija kreira cvor i postavlja mu sadrzaj na dati broj.
   Funkcija vraca pokazivac na kreirani cvor. */
CVOR* napravi_cvor(int br) {
    CVOR* novi = (CVOR*)malloc(sizeof(CVOR));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }
    novi->br = br;
    novi->sl = NULL;
    return novi;
}

/* ------------------------------------------- */
/* Ispisivanje liste: iterativna verzija */
void ispisi_listu_i(CVOR* l) {
```

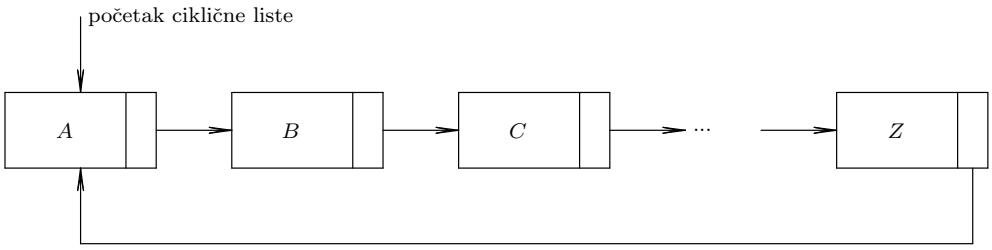
```
    CVOR* t;
    for (t = l; t != NULL; t=t->sl)
        printf("%d ", t->br);
}

/* Ispisivanje liste: rekurzivna verzija */
void ispisi_listu_r(CVOR* l) {
    if (l != NULL)
    {
        printf("%d ", l->br);
        ispisi_listu_r(l->sl);
    }
}

/* Ispisivanje liste unazad: rekurzivna verzija */
void ispisi_listu_unazad(CVOR* l) {
    if (l != NULL)
    {
        ispisi_listu_unazad(l->sl);
        printf("%d ", l->br);
    }
}

/* ------------------------------------------- */
/* Oslobadjanje liste : iterativna verzija */
void oslobodi_listu_i(CVOR* l) {
    while (l!=NULL)
    {
        CVOR* tmp = l->sl;
        free(l);
        l = tmp;
    }
}

/* Oslobadjanje liste : rekurzivna verzija */
void oslobodi_listu_r(CVOR* l) {
    if (l != NULL)
    {
        oslobodi_listu_r(l->sl);
        /* Prvo se oslobadja poslednji element liste */
        free(l);
    }
}

/* ------------------------------------------- */
/* Ubacuje dati broj na pocetak date liste.
   Funkcija pozivaocu eksplicitno vraca pocetak
   rezultujuce liste.*/
CVOR* ubaci_na_pocetak(CVOR* l, int br) {
    CVOR* novi = napravi_cvor(br);
```

```
    novi->sl = l;
    return novi;
}

/* Funkcija vraca pocetak rezultujuce liste, a ubacuje
   cvor na kraj bez pamcenja pokazivaca na kraj */
CVOR* ubaci_na_kraj(CVOR* l, int br) {
    CVOR* novi = napravi_cvor(br);

    if (l == NULL)
        return novi;
    else
    {
        CVOR* t;
        /* Prodjemo do kraja liste */
        for (t = l; t->sl!=NULL; t=t->sl)
            ;
        t->sl = novi;

        /* Pocetak se nije promenio */
        return l;
    }
}

/* Rekurzivna varijanta prethodne funkcije.
   I ova funkcija vraca pokazivac na pocetak
   rezultujuce liste  */
CVOR* ubaci_na_kraj_rekurzivno(CVOR* l, int br) {
    if (l == NULL)
    {
        CVOR* novi = napravi_cvor(br);
        return novi;
    }

    l->sl = ubaci_na_kraj_rekurzivno(l->sl, br);
    return l;
}

/* ------------------------------------------- */
/* Kljucna ideja u realizaciji ove funkcije je
   pronalazenje poslednjeg elementa liste ciji
   je kljuc manji od datog elementa br.
*/
CVOR* ubaci_sortirano(CVOR* pl, int br) {
    CVOR* novi = napravi_cvor(br);

    /* U sledeca dva slucaja ne postoji cvor
       ciji je kljuc manji od datog broja (br)
       - Prvi je slucaj prazne liste
       - Drugi je slucaj kada je br manji od prvog elementa
```

```
       U oba slucaja ubacujemo na pocetak liste.
    */
    if (pl == NULL || br < pl->br)
    {
        novi->sl = pl;
        pl = novi;
    }
    else
    {
    /* Krecemo od pocetka i idemo dalje sve dok t nije poslednji
       manji element liste ili eventualno bas poslednji */
    CVOR* t;
    for(t = pl; t->sl!=NULL && t->sl->br < br; t=t->sl)
        ;
    novi->sl = t->sl;
    t->sl = novi;
    }

    return pl;
}

main() {
    CVOR* l = NULL;
    CVOR* s = NULL;

    int i;
    for (i = 0; i<5; i++)
        l = ubaci_na_kraj(l, i);
    for (; i<10; i++)
        l = ubaci_na_kraj_rekurzivno(l, i);

    for (i = 0; i < 10 ; i++)
        l = ubaci_na_pocetak(l, i);

    ispisi_listu_i(l);
    putchar('\n');

    ispisi_listu_r(l);
    putchar('\n');

    ispisi_listu_unazad(l);
    putchar('\n');

    oslobodi_listu_i(l);

    s = ubaci_sortirano(s, 5);
    s = ubaci_sortirano(s, 8);
    s = ubaci_sortirano(s, 7);
    s = ubaci_sortirano(s, 6);
```

```
    s = ubaci_sortirano(s, 4);

    ispisi_listu_r(s);
    putchar('\n');

    oslobodi_listu_r(s);
}
```

4.48.2 Kružna lista



Slika 4.2: Kružna lista

Primer 4.48.3 (februar 2006.) *Grupa od n plesača (na čijim kostimima su u smeru kazaljke na satu redom brojevi od* 1 *do n) izvodi svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k-ti plesač (odbrojava se počev od plesača označenog brojem* 1 *u smeru kretanja kazaljke na satu). Preostali plesači obrazuju manji krug iz kog opet izlazi k-ti plesač (odbrojava se pocev od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu). Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi* `n`, `k` *(*$k < n$*) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga.*

PRIMER: za $n = 5$, $k = 3$ *redosled izlaska je* 3 1 5 2 4.

4.48.3 Stek

Primer 4.48.4 *Program proverava da li su zagrade (i) dobro uparene.*

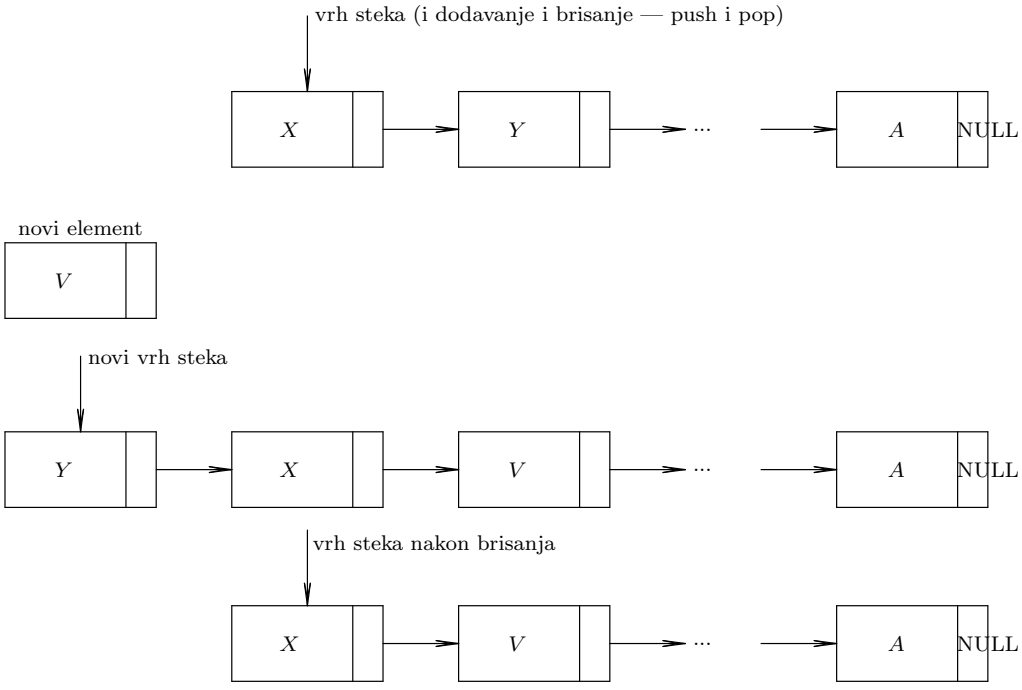
```
#include <stdio.h>
#include <stdlib.h>

main()
{
int c;
int br_otv = 0;
while((c=getchar()) != EOF)
{
    switch(c)
```

```
    {
        case '(':
            br_otv++;
            break;
        case ')':
            br_otv--;
            if (br_otv<0)
            {
            printf("Visak zatvorenih zagrada\n");
            exit(1);
            }
    }
}

if (br_otv == 0)
    printf("Zagrade su u redu\n");
else
    printf("Visak otvorenih zagrada\n");
}
```

Primer 4.48.5 *Program proverava da li su zagrade (, [, {, },] i) dobro uparene - statička implementacija steka.*

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_ZAGRADA 100

int odgovarajuce(char z1, char z2) {
    return (z1 == '(' && z2 == ')') ||
           (z1 == '{' && z2 == '}') ||
           (z1 == '[' && z2 == ']');
}

main()
{
int c;
char otv_zagrade[MAX_ZAGRADA];
int br_otv = 0;

while((c=getchar()) != EOF)
{
switch(c)
{
    case '(':
    case '{':
    case '[':
        {
        otv_zagrade[br_otv] = c;
        br_otv++;
        break;
```

```
        }
    case ']':
    case '}':
    case ')':
        if (br_otv>0 && odgovarajuce(otv_zagrade[br_otv-1], c))
        {
            br_otv--;
        }
        else
        {
        printf("Visak zatvorenih zagrada: %c u liniji %d\n", c, br_linija);
        exit(1);
        }
}
}

if (br_otv == 0)
    printf("Zagrade su u redu\n");
else
    printf("Visak otvorenih zagrada\n");
}
```



Slika 4.3: Stek

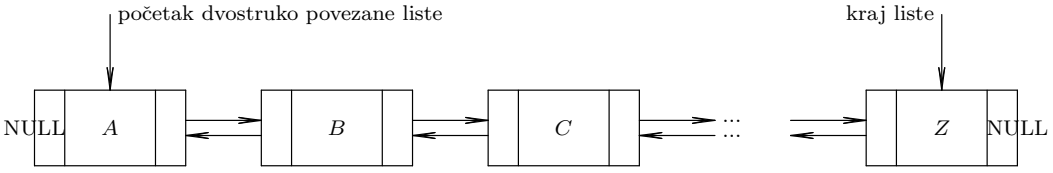
Primer 4.48.6 *Program ilustruje proveru validnosti HTML datoteke - proverava se da li su etikete dobro uparene pri cemu se stek implementira preko liste.*

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

/* Maksimalna duzina etikete */
#define MAX_TAG 100

#define OTVORENA 1
#define ZATVORENA 2
#define GRESKA 0

/* Funkcija ucitava sledecu etiketu i smesta njen naziv u niz s
   duzine max. Vraca OTVORENA za otvorenu etiketu, ZATVORENA za
   zatvorenu etiketu, odnosno GRESKA inace */
int gettag(char s[], int max) {
    int c, i;
    int zatvorenost=OTVORENA;

    /* Preskacemo sve do znaka '<' */
    while ((c=getchar())!=EOF && c!='<')
                   ;
    /* Nismo naisli na etiketu */
    if (c==EOF)
           return GRESKA;

    /* Proveravamo da li je etiketa zatvorena */
    if ((c=getchar())=='/')
       zatvorenost=ZATVORENA;
    else
       /* Funkcija ungetc vraca karakter c na standardni ulaz */
       ungetc(c,stdin);

    /* Citamo etiketu dok nailaze slova i smestamo ih u nisku */
    for (i=0; isalpha(c=getchar()) && i<max-1; s[i++] = c)
                     ;
    /* Vracamo poslednji karakter na ulaz jer je to bio neki karakter
       koji nije slovo*/
    ungetc(c,stdin);

    s[i]='\0';

    /* Preskacemo atribute do znaka > */
    while ((c=getchar())!=EOF && c!='>')
        ;

    /* Greska ukoliko nismo naisli na '>' */
    return c=='>' ? zatvorenost : GRESKA;
```

```
}

/* Stek ce biti implementiran koriscenjem liste */
typedef struct
cvor {
    char tag[MAX_TAG];
    struct cvor* sledeci;
} CVOR;

CVOR* stek = NULL;

main()
{
char tag[MAX_TAG];
int zatvorenost;
while ((zatvorenost = gettag(tag, MAX_TAG))>0)
{
    if (zatvorenost==OTVORENA)
    {
    /* Svaku otvorenu etiketu stavljamo na stek */

    CVOR* tmp = (CVOR*)malloc(sizeof(CVOR));
    if (tmp == NULL)
    {
        printf("Greska prilikom alokacije memorije!\n");
        return 1;
    }
    strcpy(tmp->tag, tag);
    tmp->sledeci = stek;
    stek = tmp;

    printf("Postavio <%s> na stek\n", stek->tag);
    }

    else
    {
    /* Za zatvorene etikete proveravamo da li je stek prazan
       odnosno da li se na vrhu steka nalazi odgovarajuca
       otvorena etiketa */
    if (stek != NULL && strcmp(stek->tag, tag) == 0)
    {
        /* Uklanjamo etiketu sa steka */
        CVOR* tmp = stek->sledeci;
        free(stek);
        stek = tmp;
    }
    else
    {
        /* Prijavljujemo gresku */
        printf("Neodgovarajuci tag : </%s>!\n",tag);
```

```
        exit(1);
    }
    }
}

if (stek == NULL) printf("Datoteka je ispravna\n ");
    else
    {
        printf("Visak otvorenih etiketa!\n");
        /* Oslobadjamo memoriju koja je ostala
           zarobljena na steku. */
        while(stek!=NULL)
        {
        CVOR* tmp = stek->sledeci;
        free(stek);
        stek = tmp;
        }
    }
}
```

4.48.4 Dvostruko povezane liste



Slika 4.4: Dvostruko povezana lista

Primer 4.48.7 *Napisati funkciju koja omogućava umetanje čvora u dvostruko povezanu kružnu listu kao i izbacivanje čora iz dvostruko povezane kružne liste. Omogućiti i štampanje podataka koje čuva lista.*

```
/* Program implementira deciju razbrajalicu eci-peci-pec i sluzi
   da ilustruje rad sa dvostruko povezanim kruznim listama */

#include <stdlib.h>
#include <stdio.h>

/* Dvostruko povezana lista */
typedef struct _cvor {
   int broj;
```

```
   struct _cvor* prethodni, *sledeci;
} cvor;

/* Umetanje u dvostruko povezanu listu */
cvor* ubaci(int br, cvor* lista) {
    cvor* novi=(cvor*)malloc(sizeof(cvor));
    if (novi==NULL)
    {   printf("Greska prilikom alokacije memorije \n");
        exit(1);
    }
    novi->broj=br;

    if (lista==NULL)
    {  novi->sledeci=novi;
       novi->prethodni=novi;
       return novi;
    }
    else
    {  novi->prethodni=lista;
       novi->sledeci=lista->sledeci;
       lista->sledeci->prethodni=novi;
       lista->sledeci=novi;
       return novi;
    }
}

/* Ispis liste */
void ispisi(cvor* lista)
{
if (lista!=NULL)
    {   cvor* tekuci=lista;
        do
        {   printf("%d\n",tekuci->broj);
            tekuci=tekuci->sledeci;
        } while (tekuci!=lista);
    }
}

/* Izbacivanje datog cvora iz liste, funkcija
   vraca pokazivac na novonastalu listu */
cvor* izbaci(cvor* lista) {
    if (lista!=NULL)
    {   cvor* sledeci=lista->sledeci;
        if (lista==lista->sledeci)
        {   printf("Pobednik %d\n",lista->broj);
            free(lista);
            return NULL;
        }

        printf("Ispada %d\n",lista->broj);
```

```
        lista->sledeci->prethodni=lista->prethodni;
        lista->prethodni->sledeci=lista->sledeci;
        free(lista);
        return sledeci;
    }
    else return NULL;
}

main() {
    /* Umecemo petoro dece u listu */
    cvor* lista = NULL;
    lista=ubaci(1,lista);
    lista=ubaci(2,lista);
    lista=ubaci(3,lista);
    lista=ubaci(4,lista);
    lista=ubaci(5,lista);
    lista=lista->sledeci;

    /*Proverimo da smo dobro formirali listu*/
    ispisi(lista);

    int smer = 0;
    /* Dok ima dece u listi */
    while(lista!=NULL)
    {   int i;

    /* brojimo 13 slogova u krug i u svakom brojanju
       menjamo smer obilaska*/
        for (i=1; i<=13; i++)
           lista = 1-smer ? lista->sledeci : lista->prethodni;

        lista=izbaci(lista);
        smer = smer ? 0 : 1;
    }
}
```

Primer 4.48.8 *Program ispisuje broj pojavljivanja za svaku od reči koja se pojavila u tekstu unetom sa standardnog ulaza. Verzija sa (sortiranom) listom.*

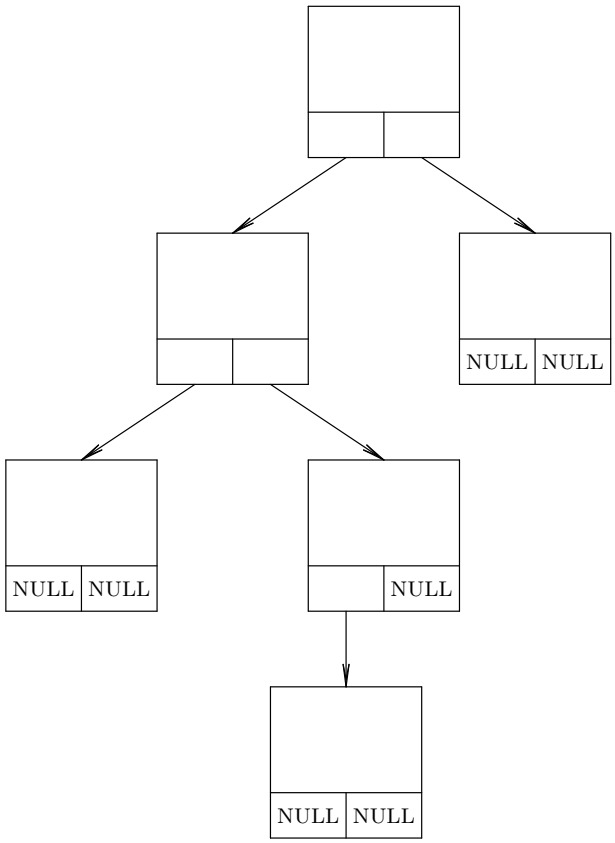
```
#include <stdlib.h>
#include <stdio.h>

/* Definicija cvora liste */
typedef struct _cvor
{
     char ime[80];
     int br_pojavljivanja;
     struct _cvor* sledeci;
} cvor;
```

```
/* Funkcija ispisuje listu rekurzivno, pocevsi od poslednjeg
elementa */
void ispisi_listu(cvor* pocetak)
{
if (pocetak!=NULL)
     {
        ispisi_listu(pocetak->sledeci);
        printf("%s %d\n",pocetak->ime,pocetak->br_pojavljivanja);
     }
}

/* Funkcija koja brise listu */
void obrisi_listu(cvor* pocetak)
{
if (pocetak!=NULL)
     {
        obrisi_listu(pocetak->sledeci);
        free(pocetak);
     }
}

/* Funkcija ubacuje rekurzivno datu rec u listu koja je
leksikografski sortirana, na odgovarajuce mesto i vraca
pokazivac na novi pocetak liste */
cvor* ubaci_sortirano(cvor* pocetak, char* rec)
{
    int cmp;
    /* Ukoliko je lista prazna ubacujemo na pocetak liste*/
    if (pocetak==NULL)
    {    pocetak=(cvor*)malloc(sizeof(cvor));
         if (pocetak == NULL)
         {
         printf("Greska prilikom alokacije memorije!\n");
         exit(1);
         }
         strcpy(pocetak->ime,rec);
         pocetak->br_pojavljivanja=1;
         return pocetak;
    }
    /* Ukoliko lista nije prazna poredimo rec sa elementom u glavi */
    cmp=strcmp(pocetak->ime,rec);
    /* Ukoliko je rec pronadjena samo uvecavamo njen broj
       pojavljivanja */
    if (cmp==0)
    {   pocetak->br_pojavljivanja++;
        return pocetak;
    }
    /* Ukoliko je rec koju ubacujemo veca od tekuce reci, ubacujemo je
       rekurzivno u rep */
    else if (cmp>0)
```

```
    {    pocetak->sledeci=ubaci_sortirano(pocetak->sledeci,rec);
         return pocetak;
    }
    /* Ukoliko je rec koju ubacujemo manja od tekuce reci, gradimo novi
       cvor i ubacujemo ga ispred pocetka */
    else
    {    cvor* novi=malloc(sizeof(cvor));
         if (novi == NULL)
         {
         printf("Greska prilikom alokacije memorije!\n");
         exit(1);
         }
         strcpy(novi->ime,rec);
         novi->br_pojavljivanja=1;
         novi->sledeci=pocetak;
         return novi;
    }
}


/* Pomocna funkcija koja cita rec sa standardnog ulaza i vraca
njenu duzinu, odnosno -1 ukoliko se naidje na EOF */
int getword(char word[], int lim) {
    int c, i=0;
    while (!isalpha(c=getchar()) && c!=EOF)
             ;

    if (c==EOF)
          return -1;
    do
    {   word[i++]=c;
    }while (i<lim-1 && isalpha(c=getchar()));

    word[i]='\0';
    return i;
}

/* Funkcija koja rekurzivno pronalazi datu rec u datoj listi.
   Funkcija vraca pokazivac na cvor u kome je nadjena rec, ili
   NULL ukoliko rec nije nadjena */
cvor* nadji_rec(cvor* lista, char rec[])
{
     if (lista==NULL)
          return NULL;
     if (strcmp(lista->ime,rec)==0)
          return lista;

     return nadji_rec(lista->sledeci,rec);
}
```

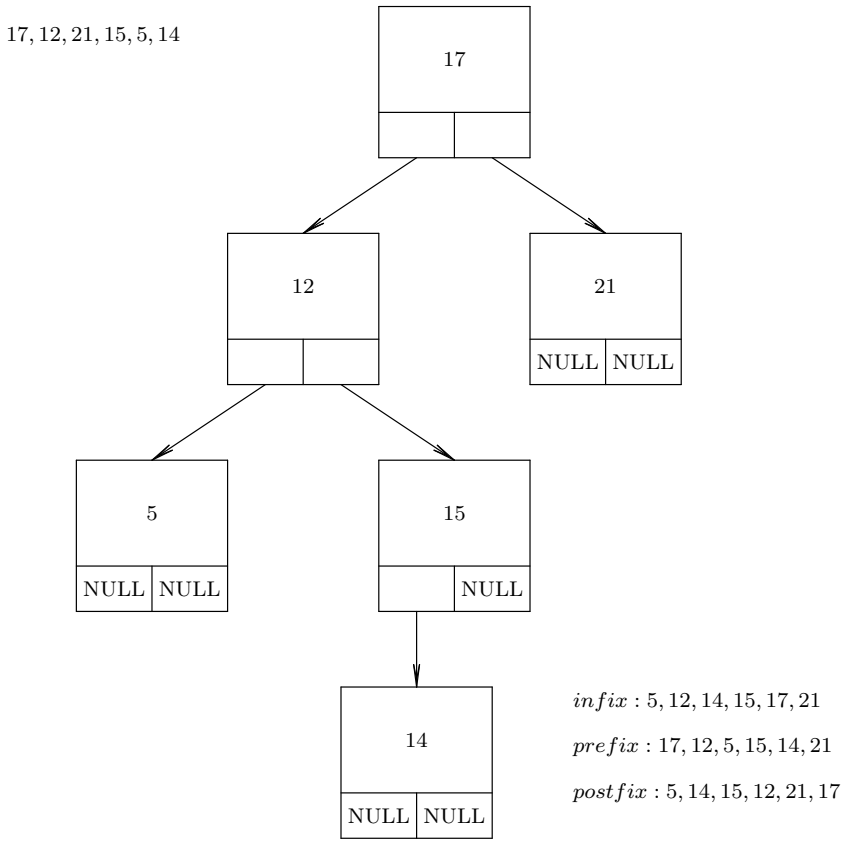
```
main()
{
   cvor* lista=NULL;
   char procitana_rec[80];
   while(getword(procitana_rec,80)!=-1)
   {     cvor* pronadjen=nadji_rec(lista,procitana_rec);
         if (pronadjen!=NULL)
               pronadjen->br_pojavljivanja++;
         else
               lista=ubaci_sortirano(lista,procitana_rec);
   }
   ispisi_listu(lista);
   obrisi_listu(lista);
}
```

4.49 Drvo



Slika 4.5: Stablo

Primer 4.49.1 *Binarno pretrazivacko drvo - drvo sadrži cele brojeve.*



Slika 4.6: Uređeno stablo

```
/* Program demonstrira rad sa binarnim pretrazivackim drvetima.
   Drveta sadrze cele brojeve i sortirana su po velicini.
   Za svaki cvor, levo podstabla sadrzi manje elemente, dok desno
   podstablo sadrzi vece.
*/

#include <stdlib.h>
#include <stdio.h>

typedef struct _cvor
{
    int broj;
    struct _cvor *l, *d;
} cvor;
```

```
cvor* napravi_cvor(int b) {
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom
                        alokacije memorije");
        exit(1);
    }
    novi->broj = b;
    novi->l = NULL;
    novi->d = NULL;
    return novi;
}

/* Funkcija umece broj br u drvo ciji je koren dat preko
   pokazivaca drvo. Funkcija vraca pokazivac na koren
   novog drveta */
cvor* ubaci_u_drvo(cvor* drvo, int b)
{
    if (drvo == NULL)
        return napravi_cvor(b);

    if (b < drvo->broj)
        drvo->l = ubaci_u_drvo(drvo->l, b);
    else
        drvo->d = ubaci_u_drvo(drvo->d, b);

    return drvo;
}

/* Funkcija proverava da li dati broj postoji u drvetu */
int pronadji(cvor* drvo, int b)
{
    if (drvo == NULL)
        return 0;

    if (drvo->broj == b)
        return 1;

    if (b < drvo->broj)
        return pronadji(drvo->l, b);
    else
        return pronadji(drvo->d, b);
}

/* Funkcija ispisuje sve cvorove drveta u infiksnom redosledu */
void ispisi_drvo(cvor* drvo) {
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
```

```
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


void obrisi_drvo(cvor* drvo) {
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}

/* Funkcija sumira sve vrednosti binarnog stabla */
int suma_cvorova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    return  suma_cvorova(drvo->l) +
        drvo->broj +
        suma_cvorova(drvo->d);
}

int broj_cvorova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    return  broj_cvorova(drvo->l) +
        1 +
        broj_cvorova(drvo->d);
}

int broj_listova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    if (drvo->l == NULL && drvo->d == NULL)
        return 1;
    return  broj_listova(drvo->l) +
        broj_listova(drvo->d);
}

int suma_listova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;

    if (drvo->l == NULL && drvo->d == NULL)
```

```
        return drvo->broj;

    return  suma_listova(drvo->l) +
        suma_listova(drvo->d);
}

void ispisi_listove(cvor* drvo)
{
    if (drvo == NULL)
        return;

    ispisi_listove(drvo->l);

    if (drvo->l == NULL && drvo->d == NULL)
        printf("%d ", drvo->broj);

    ispisi_listove(drvo->d);
}

/* Funkcija pronalazi maksimalnu vrednost u drvetu
   Koristi se cinjenica da je ova vrednost
   smestena u najdesnjem listu */
int max_vrednost(cvor* drvo)
{
    if (drvo==NULL)
       return 0;

    if (drvo->d==NULL)
       return drvo->broj;

    return max_vrednost(drvo->d);
}

/* Iterativna funkcija za pronalazenje maksimalne vrednosti. */
int max_vrednost_nerekurzivno(cvor* drvo)
{
   if (drvo==NULL)
        return 0;
   else
   {
    cvor* tekuci;
    for (tekuci=drvo; tekuci->d!=NULL; tekuci=tekuci->d)
                ;
    return tekuci->broj;
   }
}


/* Funkcija racuna "dubinu" binarnog stabla */
#define max(a,b) (((a)>(b))?(a):(b))
```

```
int dubina(cvor* drvo)
{
  if (drvo==NULL)
     return 0;
  else
  {  int dl=dubina(drvo->l);
     int dd=dubina(drvo->d);
     return 1+max(dl,dd);
  }
}

main()
{
cvor* drvo = NULL;
drvo = ubaci_u_drvo(drvo, 1);
drvo = ubaci_u_drvo(drvo, 8);
drvo = ubaci_u_drvo(drvo, 5);
drvo = ubaci_u_drvo(drvo, 3);
drvo = ubaci_u_drvo(drvo, 7);
drvo = ubaci_u_drvo(drvo, 6);
drvo = ubaci_u_drvo(drvo, 9);

if (pronadji(drvo, 3))
    printf("Pronadjeno 3\n");
if (pronadji(drvo, 2))
    printf("Pronadjeno 2\n");
if (pronadji(drvo, 7))
    printf("Pronadjeno 7\n");

ispisi_drvo(drvo);

putchar('\n');
printf("Suma cvorova : %d\n", suma_cvorova(drvo));
printf("Broj cvorova : %d\n", broj_cvorova(drvo));
printf("Broj listova : %d\n", broj_listova(drvo));
printf("Suma listova : %d\n", suma_listova(drvo));
printf("Dubina drveta : %d\n", dubina(drvo));
printf("Maximalna vrednost : %d\n", max_vrednost(drvo));

ispisi_listove(drvo);

obrisi_drvo(drvo);
}

/*
Pronadjeno 3
Pronadjeno 7
1 3 5 6 7 8 9
Suma cvorova : 39
```

```
Broj cvorova : 7
Broj listova : 3
Suma listova : 18
Dubina drveta : 5
Maximalna vrednost : 9
3 6 9
*/
```

Primer 4.49.2 *Program sa ulaza čita tekst i ispisuje broj pojavljivanja svake od reči koje su se javljale u tekstu. Radi poboljsanja efikasnosti, prilikom brojanja reci koristi se struktura podataka pogodna za leksikografsku pretragu - u ovom slucaju binarno pretrazivacko drvo.*

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

/* Cvor drveta sadrzi ime reci i
   broj njenih pojavljivanja */
typedef struct _cvor {
     char ime[80];
     int br_pojavljivanja;
     struct _cvor* levo, *desno;
} cvor;

/* Funkcija ispisuje drvo u inorder redosledu */
void ispisi_drvo(cvor* drvo)
{
if (drvo!=NULL)
    {
    ispisi_drvo(drvo->levo);
    printf("%s %d\n",drvo->ime,drvo->br_pojavljivanja);
    ispisi_drvo(drvo->desno);
    }
}

/* Funkcija uklanja binarno drvo iz memorije */
void obrisi_drvo(cvor* drvo)
{
if (drvo!=NULL)
     {
     obrisi_drvo(drvo->levo);
     obrisi_drvo(drvo->desno);
     free(drvo);
     }
}

/* Funkcija ubacuje datu rec u dato drvo
  i vraca pokazivac na koren drveta */
cvor* ubaci(cvor* drvo, char rec[]) {
    /* Ukoliko je drvo prazno gradimo novi cvor */
    if (drvo==NULL)
```

```
    {
    cvor* novi_cvor=(cvor*)malloc(sizeof(cvor));
    if (novi_cvor==NULL)
    {
    printf("Greska prilikom alokacije memorije\n");
    exit(1);
    }
    strcpy(novi_cvor->ime, rec);
    novi_cvor->br_pojavljivanja=1;
    return novi_cvor;
    }

    /* Uporedjujemo rec koju smo ucitali sa recju koja
       se nalazi u cvoru drveta*/
    int cmp = strcmp(rec, drvo->ime);

    /* Ukoliko rec vec postoji u drvetu
       uvecavamo njen broj pojavljivanja */
    if (cmp==0)
    {   drvo->br_pojavljivanja++;
        return drvo;
    }

    /* Ukoliko je rec koju ubacujemo leksikografski
       ispred reci koja je u korenu drveta, rec
       ubacujemo u levo podstablo */
    if (cmp<0)
    {   drvo->levo=ubaci(drvo->levo, rec);
        return drvo;
    }

    /* Ukoliko je rec koju ubacujemo
       leksikografski iza reci koja je u
       korenu drveta, rec ubacujemo u desno
       podstablo */
    if (cmp>0)
    {   drvo->desno=ubaci(drvo->desno, rec);
        return drvo;
    }
}

/* Pomocna funkcija koja cita rec sa
   standardnog ulaza i vraca njenu
   duzinu, odnosno -1 ukoliko se naidje na EOF.
   Ukoliko umesto funkcije getword koristimo
   funkciju gettag program ce ispisivati
   broj pojavljivanja svakog od tagova sa ulaza */
int getword(char word[], int lim) {
    int c, i=0;
    while (!isalpha(c=getchar()) && c!=EOF)
```

```
            ;

    if (c==EOF)
          return -1;
    do
    {
       word[i++]=c;
    } while (i<lim-1 && isalpha(c=getchar()));

    word[i]='\0';
    return i;
}

main() {
   /* Drvo je na pocetku prazno */
   cvor* drvo=NULL;
   char procitana_rec[80];

   /* Citamo rec po rec dok ne
      naidjemo na kraj datoteke i
      ubacujemo ih u drvo */
   while(getword(procitana_rec,80)!=-1)
      drvo=ubaci(drvo,procitana_rec);

   /* Ispisujemo drvo */
   ispisi_drvo(drvo);

   /* Uklanjamo ga iz memorije */
   obrisi_drvo(drvo);
}
```

Primer 4.49.3 *Program iz datoteke čita tekst i ispisuje* `n` *najfrekventnijih reči koje su se javljale u tekstu.*

Radi poboljšanja efikasnosti, prilikom brojanja reči koristi se struktura podataka pogodna za leksikografsku pretragu - u ovom slučaju binarno pretraživačko drvo.

Na kraju rada, čvorovi drveta se "presortiraju" na osnovu broja pojavljivanja. Zbog ovoga je potrebno čuvati niz pokazivača na različite čvorove drveta.

Da bismo ispisali 10 najfrekventnijih etiketa, potrebno je zameniti funkciju `getword` *funkcijom* `gettag`.

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

/* Cvor drveta sadrzi ime reci i
   broj njenih pojavljivanja */
typedef struct _cvor {
    char ime[80];
     int br_pojavljivanja;
     struct _cvor* levo, *desno;
```

```
} cvor;

/* Gradimo niz pokazivaca na cvorove drveta koji ce
   nam sluziti da po prihvatanju svih reci izvrsimo
   sortiranje po broju pojavljivanja */

#define MAX_BROJ_RAZLICITIH_RECI 1000

cvor* razlicite_reci[MAX_BROJ_RAZLICITIH_RECI];

/* Tekuci broj cvorova drveta */
int broj_razlicitih_reci=0;

/* Funkcija uklanja binarno drvo iz memorije */
void obrisi_drvo(cvor* drvo)
{
     if (drvo!=NULL)
     {  obrisi_drvo(drvo->levo);
        obrisi_drvo(drvo->desno);
        free(drvo);
     }
}

/* Funkcija ubacuje datu rec u dato drvo i vraca pokazivac na
   koren drveta */
cvor* ubaci(cvor* drvo, char rec[])
{
    /* Ukoliko je drvo prazno gradimo novi cvor */
    if (drvo==NULL)
    {
        cvor* novi_cvor=(cvor*)malloc(sizeof(cvor));
        if (novi_cvor==NULL)
        {
        printf("Greska prilikom alokacije memorije\n");
        exit(1);
        }
        strcpy(novi_cvor->ime, rec);
        novi_cvor->br_pojavljivanja=1;

        /* pokazivac na novo napravljeni cvor smestamo u niz
        pokazivaca na sve cvorove drveta */
        razlicite_reci[broj_razlicitih_reci++] = novi_cvor;

        return novi_cvor;
    }
    int cmp = strcmp(rec, drvo->ime);

    /* Ukoliko rec vec postoji u drvetu
       uvecavamo njen broj pojavljivanja */
    if (cmp==0)
```

```
    {   drvo->br_pojavljivanja++;
        return drvo;
    }

    /* Ukoliko je rec koju ubacujemo leksikografski ispred
       reci koja je u korenu drveta, rec ubacujemo
       u levo podstablo */
    if (cmp<0)
    {   drvo->levo=ubaci(drvo->levo, rec);
        return drvo;
    }

    /* Ukoliko je rec koju ubacujemo
       leksikografski iza reci koja je u
       korenu drveta, rec ubacujemo
       u desno podstablo */
    if (cmp>0)
    {   drvo->desno=ubaci(drvo->desno, rec);
        return drvo;
    }
}

/* Pomocna funkcija koja cita rec iz date datoteke i vraca
   njenu duzinu, odnosno -1 ukoliko se naidje na EOF */
int getword(char word[], int lim, FILE* ulaz)
{
    int c, i=0;
    /* Umesto funkcije getchar koristimo fgetc
      za rad sa datotekama */
    while (!isalpha(c=fgetc(ulaz)) && c!=EOF)
             ;
    if (c==EOF)
          return -1;
    do
    {   word[i++]=c;
    }while (i<lim-1 && isalpha(c=fgetc(ulaz)));

    word[i]='\0';
    return i;
}

/* Funkcija poredjenja za funkciju qsort. */
int poredi_br_pojavljivanja(const void* a, const void* b)
{
    return
        /* Konverzija pokazivaca b iz pokazivaca
           na tip void u pokazivac na cvor jer
           nam je svaki element niza koji sortiramo
           tipa pokazivaca na cvor */
        (*(cvor**)b)->br_pojavljivanja
```

```
        -
        (*(cvor**)a)->br_pojavljivanja;
}


main(int argc, char* argv[])
{
   int i;

   /* Drvo je na pocetku prazno */
   cvor* drvo=NULL;
   char procitana_rec[80];
   FILE* ulaz;

   if (argc!=2)
   {   fprintf(stderr,"Greska :
             Ocekivano ime datoteke\n");
       exit(1);
   }

   if ((ulaz=fopen(argv[1],"r"))==NULL)
   {
       fprintf(stderr,"Greska : nisam uspeo da otvorim datoteku %s\n");
       exit(1);
   }

   /* Citamo rec po rec dok ne naidjemo na kraj datoteke i
      ubacujemo ih u drvo */
   while(getword(procitana_rec,80,ulaz)!=-1)
      drvo=ubaci(drvo,procitana_rec);

   /* Sortiramo niz pokazivaca na cvorove
      drveta po broju pojavljivanja */
   qsort(razlicite_reci,
            broj_razlicitih_reci,
                  sizeof(cvor*),
                       poredi_br_pojavljivanja);

   /* Ispisujemo prvih 10 (ukoliko ih ima)
      reci i njihov broj pojavljivanja */
   for (i=0; i<10 && i<broj_razlicitih_reci; i++)
       printf("%s - %d\n",razlicite_reci[i]->ime,
             razlicite_reci[i]->br_pojavljivanja);

   /* Uklanjamo drvo iz memorije */
   obrisi_drvo(drvo);

   fclose(ulaz);
}
```

4.50 Grafovi

Graf[1] `G=(V,E)` sastoji se od skupa `V` čvorova i skupa `E` grana. Grane predstavljaju relacije između čvorova i odgovara paru čvorova. Graf može biti usmeren (orijentisan), ako su mu grane uređeni parovi i neusmeren (neorjentisan) ako su grane neuređeni parovi.

Uobičajena su dva načina predstavljanja grafova. To su *matrica povezanosti* grafa i *lista povezanosti*.

Matrica povezanosti je kvadratna matrica dimenzije `n`, pri čemu je `n` broj čvorova u grafu, takva da je element na preseku `i`-te vrste i `j`-te kolone jednak jedinici ukoliko postoji grana u grafu od `i`-tog do `j`-tog čvora, inače je nula.

Umesto da se i sve nepostojeće grane eksplicitno predstavljaju u matrici povezanosti, mogu se formirati povezane liste od jedinica iz `i`-te vrste za `i=1,2,...,n`. To je lista povezanosti. Svakom čvoru se pridružuje povezana lista, koja sadrži sve grane susedne tom čvoru. Graf je predstavljen vektorom lista. Svaki elemenat vektora sadrži ime (indeks) čvora i pokazivač na njegovu listu čvorova.

Prvi problem na koji se nailazi pri konstrukciji bilo kog algoritma za obradu grafa je kako pregledati ulaz. Postoje dva osnovna algoritma za obilazak grafa: pretraga u dubinu (`DFS`, skraćenica od depth-first-search) i pretraga u širinu (`BFS`, skraćenica od breadth-first-search).

Kod `DFS` algoritma, obilazak započinje iz proizvoljnog zadatog čvora `r` koji se naziva *koren pretrage u dubinu*. Koren se označava kao posećen. Zatim se bira proizvoljan neoznačen čvor `r1`, susedan sa `r`, pa se iz čvora `r1` rekurzivno startuje pretraga u dubinu. Iz nekog nivoa rekurzije izlazi se kad se naiđe na čvor v kome su svi susedi već označeni.

Primer 4.50.1 *Primer reprezentovanja grafa preko matrice povezanosti. U programu se unosi neorijentisan graf i DFS algoritmom se utvrđuju čvrovi koji su dostižni iz čvora 0.*

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

int** alociraj_matricu(int n)
{   int **matrica;
    int i;
    matrica=malloc(n*sizeof(int*));
    if (matrica==NULL)
    {
    printf("Greska prilikom alokacije memorije\n");
    exit(1);
    }

    for (i=0; i<n; i++)
       {
        /* Funkcija calloc popunjava rezervisan
           prostor u memoriji nulama. */
        matrica[i]=calloc(n,sizeof(int));
```

[1] Tekst i primeri preuzeti od Jelene Tomašević, url: www.matf.bg.ac.yu/~jtomasevic, zasnovano na materijalu *Algoritmi, Miodrag Živković* i http://www.matf.bg.ac.yu/~filip

```
        if (matrica[i]==NULL)
        {
        printf("Greska prilikom alokacije memorije\n");
        exit(1);
        }
       }
    return matrica;
}

void oslobodi_matricu(int** matrica, int n)
{   int i;
    for (i=0; i<n; i++)
        free(matrica[i]);
    free(matrica);
}

int* alociraj_niz(int n)
{   int* niz;
    niz=calloc(n,sizeof(int));
    if (niz==NULL)
    {
    printf("Greska prilikom alokacije memorije\n");
    exit(1);
    }
    return niz;
}

void oslobodi_niz(int* niz)
{   free(niz);
}

void unesi_graf(int** graf, int n)
{     int i,j;
      for (i=0; i<n; i++)
         for (j=i; j<n; j++)
         {   printf("Da li su element %d i %d povezani : ",i,j);
             do
             {  scanf("%d",&graf[i][j]);
                /* Radimo sa neusmerenim grafom */
                graf[j][i]=graf[i][j];
             } /* Obezbedjujemo ispravan unos */
             while (graf[i][j]!=0 && graf[i][j]!=1);
         }
}

void ispisi_graf(int** graf, int n)
{     int i,j;
      for (i=0; i<n; i++)
      {  for (j=0; j<n; j++)
             printf("%d",graf[i][j]);
```

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



/* Broj cvorova grafa (dimenzija matrice) */
int n;
/* Matrica povezanosti */
int **graf;


/* Pomocni vektor koji govori o tome koji su cvorovi posecivani
   tokom DFS obilaska */
int *posecen;

/* Rekurzivna implementacija DFS algoritma */
void poseti(int i)
{    int j;
     posecen[i]=1;
     printf("Posecujem cvor %d\n",i);
     for (j=0; j<n; j++)
        if (graf[i][j] && !posecen[j])
               poseti(j);
}


main()
{
     int i, j;
     printf("Unesi broj cvorova : ");
     scanf("%d",&n);

     graf=alociraj_matricu(n);
     unesi_graf(graf,n);
     ispisi_graf(graf,n);

     posecen=alociraj_niz(n);
     poseti(0);

     oslobodi_niz(posecen);
     oslobodi_matricu(graf,n);
}
```

Primer 4.50.2 *Primer predstavljanja grafa preko niza listi suseda svakog od čvorova grafa U programu se unosi graf i DFS algoritmom se utvrdjuje koji su čvorovi dostižni iz cvora 0.*

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

```
/* Cvor liste suseda */
typedef struct _cvor_liste
{  int broj; /* Indeks suseda */
   struct _cvor_liste* sledeci;
} cvor_liste;

/* Ubacivanje na pocetak liste */
cvor_liste* ubaci_u_listu(cvor_liste* lista, int broj)
{   cvor_liste* novi=malloc(sizeof(cvor_liste));
    if (novi==NULL)
    {
    printf("Greska prilikom alokacije memorije\n");
    exit(1);
    }
    novi->broj=broj;
    novi->sledeci=lista;
    return novi;
}

/* Rekurzivno brisanje liste */
void obrisi_listu(cvor_liste* lista)
{   if (lista)
    {  obrisi_listu(lista->sledeci);
       free(lista);
    }
}

/* Ispis liste */
void ispisi_listu(cvor_liste* lista)
{   if (lista)
    {  printf("%d ",lista->broj);
       ispisi_listu(lista->sledeci);
    }
}

/* Graf predstavlja niz pokazivaca na pocetke listi suseda */
#define MAX_BROJ_CVOROVA 100
cvor_liste* graf[MAX_BROJ_CVOROVA];
int broj_cvorova;


/* Rekurzivna implementacija DFS algoritma */
int posecen[MAX_BROJ_CVOROVA];
void poseti(int i)
{   cvor_liste* sused;
    printf("Posecujem cvor %d\n",i);
    posecen[i]=1;
    for( sused=graf[i]; sused!=NULL; sused=sused->sledeci)
        if (!posecen[sused->broj])
              poseti(sused->broj);
```

```
}


main()
{
    int i;
    printf("Unesi broj cvorova grafa : ");
    scanf("%d",&broj_cvorova);
    for (i=0; i<broj_cvorova; i++)
    {    int br_suseda,j;

         graf[i]=NULL;

         printf("Koliko cvor %d ima suseda : ",i);
         scanf("%d",&br_suseda);
         for (j=0; j<br_suseda; j++)
         {    int sused;
              do
              {
                printf("Unesi broj %d.-tog suseda cvora %d : ",j,i);
                scanf("%d",&sused);
              } while (sused<1 && sused>broj_cvorova);
              graf[i]=ubaci_u_listu(graf[i],sused-1);
         }
    }

    for (i=0; i<broj_cvorova; i++)
    {   printf("%d - ",i);
        ispisi_listu(graf[i]);
        printf("\n");
    }

    poseti(0);
}
```

4.51 Razni zadaci

Primer 4.51.1 *Stepenovanje prirodnog broja efikasno:* $n^k = (n^{\frac{k}{2}})^2$ *ako je* k *parno ili* $n^k = n(n^{\frac{k-1}{2}})^2$ *ako je* k *neparno.*

```
int stepen(int n, int k)
{
 int p, s;
 if (k==1) s=n;
 else
    {
    p=stepen(n,k/2);
    if(k%2==0) s=p*p;
    else s=p*p*n;
```

```
    }
return s;
}
```

Primer 4.51.2 *MINESWEEPER - primer jednostavne igrice.*

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

/* Dimenzija table */
int   n;

/* Tabla koja sadrzi 0 i 1 u zavisnosti
   od toga da li na polju postoji bomba */
int** bombe;

#define PRAZNO (-1)
#define ZATVORENO 0
#define ZASTAVICA 9

/* Tabla koja opisuje tekuce stanje igre.
    Moze da sadrzi sledece vrednosti :
    ZATVORENO - opisuje polje
            koje jos nije bilo otvarano
    PRAZNO - polje na kome ne
            postoji ni jedna bomba
    BROJ od 1-8 - polje koje je
            otvoreno i na kome pise
            koliko bombi postoji u okolini
    ZASTAVICA - polje koje je korisnik
            oznacio zastavicom
*/
int** stanje;

/* Ukupan broj bombi */
int broj_bombi;

/* Ukupan broj postavljenih zastavica */
int broj_zastavica = 0;

/* Pomocne funkcije za rad sa matricama */
int** alociraj(int n)
{
   int i;
   int** m=(int**)malloc(n*sizeof(int*));
   for (i=0; i<n; i++)
       m[i]=(int *)calloc(n,sizeof(int));
   return m;
}
```

```
void obrisi(int** m, int n)
{   int i;
    for (i=0; i<n; i++)
       free(m[i]);

    free(m);
}

/* Funkcija postavlja bombe */ void postavi_bombe()
{  broj_bombi=(n*n)/6;
   int kolona;
   int vrsta;
   int i;

   /* Inicijalizujemo generator slucajnih brojeva */
   srand(time(NULL));

   for (i=0; i<broj_bombi; i++)
   {    /* Racunamo slucajni polozaj bombe */
        kolona=rand()%n;
        vrsta=rand()%n;


        /* Ukoliko bomba vec postoji tu,
           opet idemo u istu iteraciju */
        if (bombe[vrsta][kolona]==1)
        {   i--;
            continue;
        }

        /* Postavljamo bombu */
        bombe[vrsta][kolona]=1;
  }
}

/* Funkcija ispisuje tablu sa bombama */
void ispisi_bombe()
{
    int i,j;
    for (i=0; i<n; i++)
    {   for (j=0; j<n; j++)
            printf("%d",bombe[i][j]);
        printf("\n");
    }
}

/* Funkcija ispisuje tekuce stanje */
void ispisi_stanje()
{   int i,j;
    for (i=0; i<n; i++)
```

```
    {   for (j=0; j<n; j++)
        {    if (stanje[i][j]==ZATVORENO)
                  printf(".");
             else if (stanje[i][j]==PRAZNO)
                  printf(" ");
             else if (stanje[i][j]==ZASTAVICA)
                  printf("*");
             else
                  printf("%d",stanje[i][j]);
        }
        printf("\n");
    }
}

/* Funkcija postavlja zastavicu na
   dato polje ili je uklanja
   ukoliko vec postoji */
void postavi_zastavicu(int i, int j)
{
     if (stanje[i][j]==ZATVORENO)
     {    stanje[i][j]=ZASTAVICA;
          broj_zastavica++;
     }
     else if (stanje[i][j]==ZASTAVICA)
     {    stanje[i][j]=ZATVORENO;
          broj_zastavica--;
     }
}

/* Funkcija izracunava koliko bombi
   postoji u okolini date bombe */
int broj_bombi_u_okolini(int v, int k)
{    int i, j;
     int br=0;
     /* Prolazimo kroz sva okolna polja */
     for (i=-1; i<=1; i++)
         for(j=-1; j<=1; j++)
         {  /* preskacemo centralno polje */
            if (i==0 && j==0)
                continue;
            /* preskacemo polja "van table" */
            if (v+i<0 || k+j<0 || v+i>=n || k+j>=n)
                continue;
            if (bombe[v+i][k+j]==1)
                br++;
         }

   return br;

}
```

```
/* Centralna funkcija koja vrsi otvaranje
   polja i pritom se otvaranje "siri"
   i na polja koja su oko datog */

void otvori_polje(int v, int k) {
    /* Ukoliko smo "nagazili" bombu
       zavrsavamo program */
    if (bombe[v][k]==1)
    {   printf("BOOOOOOOOOOOOOOOOM!!!!\n");
        ispisi_bombe();
        exit(1);
    }
    else
    {
         /* Brojimo bombe u okolini */
         int br=broj_bombi_u_okolini(v,k);

         /* Azuriramo stanje ovog polja */
         stanje[v][k]=(br==0)?PRAZNO:br;

         /* Ukoliko u okolini nema bombi,
            rekurzivno otvaramo
            sva polja u okolini koja su zatvorena */
         if (br==0)
         {
        /* Petlje indeksiraju sva okolna polja */
        int i,j;
        for (i=-1; i<=1; i++)
            for (j=-1; j<=1; j++)
            {
            /* Preskacemo centralno polje */
            /* if (i==0 && j==0)
                    continue;   */
            /* Preskacemo polja van table */
            if (v+i<0 || v+i>=n || k+j<0 || k+j>=n)
                    continue;
            /* Ukoliko je okolno polje
               zatvoreno, otvaramo ga */
            if (stanje[v+i][k+j]==ZATVORENO)
                    otvori_polje(v+i, k+j);
            }
         }
    }
}

/* Funkcija utrdjuje da li je partija gotova
   Partija je gotova u trenutku kada su sve
   bombe pokrivene zastavicama i
   kada nijedno drugo polje nije
```

```
   pokriveno zastavicom
*/

int gotova_partija()
{  int i,j;
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
      {   /* Ukoliko postoji nepokrivena bomba,
             partija nije zavrsena */
          if (bombe[i][j]==1 && stanje[i][j]!=ZASTAVICA)
              return 0;
      }

  /* Partija je zavrsena samo ukoliko je
     broj zastavica jednak broj bombi */
  return broj_zastavica==broj_bombi;

}

main() {

  /* Unosimo dimenziju table */
  printf("Unesite dimenziju table : ");
  scanf("%d",&n);


  /* Alociramo table */
  bombe=alociraj(n);
  stanje=alociraj(n);

  /* Postavljamo bombe */
  postavi_bombe();

  /* Sve dok partija nije gotova */
  while(!gotova_partija())
  {    int v,k;
       char akcija;

       /* Ispisujemo tekuce stanje */
       ispisi_stanje();

       /* Sve dok korisnik ne unese o ili z
          trazimo od njega da upise
          odgovarajucu akciju */
       do
       {
       getchar();
       printf("Unesi akciju (o - otvaranje polja,
                   z - postavljanje zastavice) : ");
       scanf("%c",&akcija);
```

```
      }  while (akcija!='o' && akcija!='z');

      /* Trazimo od korisnika da unese koordinate
         polja sve dok ih ne unese ispravno
         Korisnicke koordinate krecu od 1,
         a interne od 0 */
      do
      {
         printf("Unesi koordinate polja : ");
         scanf("%d",&v);
         scanf("%d",&k);
      } while(v<1 || v>n || k<1 || k>n);

      /* Reagujemo na akciju */
      switch(akcija)
      {   case 'o':
              otvori_polje(v-1,k-1);
              break;
          case 'z':
              postavi_zastavicu(v-1,k-1);
      }
  }

  /* Konstatujemo pobedu */
  ispisi_stanje();
  printf ("Cestitam! Pobedili ste\n");
  obrisi(stanje,n);
  obrisi(bombe,n);
}
```