dr Jozef Kratica

Skripta iz

Osnova Programiranja

na Matematičkom fakultetu

programski jezik C

Beograd

2004.

1. Osnovni deo

1.1. Uvod

Programski jezik C je nastao krajem 60-tih godina prošlog veka[1], kao prvi programski jezik visokog nivoa sposoban i za sistemsko programiranje (do tada rezervisano isključivo za asembler/mašinski jezik niskog nivoa). Zbog te težnje za (malim) približavanjem jezika C samim mašinama i mogućnošću pisanja što bržih programa, on ponekada malo čudno izgleda, ali kada se naviknemo na njega to nam upravo izgleda kao najlogičnije rešenje.

1.1.1. Modularno programiranje

Klasični C je baziran na konceptu tzv. *modularnog* programiranja. Program delimo na neke logičke celine – *module* (radi lakšeg pisanja) i njihovim pozivanjem sve sklapamo u jednu celinu – glavni program. Za razliku od ostalih programskih jezika gde imamo glavni program, procedure i funkcije (recimo kao u Pascal-u), u C-u je osnovna i jedina celina funkcija. U C-u se i glavni program i procedure pišu kao funkcije. Pri tome se pojam funkcije posmatra na sličan način (koliko je to moguće) kao u matematici, ali se sa tim nije moglo ići do kraja (kao u funkcionalnim programskim jezicima, npr. LISP-u) zbog želje za sistemskim programiranjem.
Veliki broj drugih programskih jezika (recimo Pascal) u okviru samog programskog jezika sadrži skoro sve što nam je skoro ikad potrebno, pa su oni vrlo glomazni (imaju više stotina naredbi). Za razliku od njih, jezgro C-a ima samo nekoliko desetina naredbi, a sve ostalo što nam je potrebno možemo naći u spoljnim bibliotekama. Već i pri pisanju najprostijih programa (zbog učitavanja i štampanja) je potrebno koristiti funkcije iz spoljnih biblioteka.
Slično kao u realnom životu, pri korišćenju biblioteka sa knjigama, potrebno je imati **katalog** (ili kataloge) date biblioteke - šta sve u njoj ima. U C su ti katalozi smešteni u posebne datoteke sa nastavkom **.h** , i smešteni u posebni direktorijum. U .h datotekama su smešteni samo zaglavlja (spiskovi) funkcija, dok se **cele funkcije** nalaze prevedene u biblitekama (u *lib* direktorijumu). U spoljnim bibliotekama ima veliki broj raznovrsnih funkcija, od kojih se neke i preklapaju.
Ovakva koncepcija ima posledicu, da za razliku od recimo Pascal-a gde svaki problem ima uglavnom jedan prirodni način rešavanja, u C skoro svaki problem se može rešiti na više načina. Iako nam se veliki izbor funkcija u spoljnim bibliotekama na prvi pogled čini kao prednost, nekada nije lako izabrati pravu, pogotovo što su razlike u brzini i utrošku memorijskog prostora među njima male, ali ne uvek i potpuno beznačajne. Još je veći problem, što rešenja istog (ili vrlo sličnog) problema mogu biti napisana na vrlo različite načine, pa da se stekne pogrešan utisak da su to potpuno različiti problemi.
Zbog toga se kroz ovu skriptu vidi jasna težnja autora da skriptu napiše što je moguće jednostavnije i da rešenja istih zadataka budu napisana na isti način, onoliko koliko je to moguće.
Uvek kada su alternative bile jednostavnost rešenja ili brzina izvršavanja, izabrano je ovo prvo (ionako se vremena izvršavanja u tim zadacima mere milisekundama, a samim tim i razlike u vremenu izvršavanja).
Takođe neke, vrlo retke, stvari nisu obrađene, pošto se ne pojavljuju na ispitima, a njihovo pojavljivanje bi moglo otežati shvatanje onoga što je bitnije u ovom trenutku. Onaj ko shvati suštinu, kasnije, kada položi ispit, relativno lako može da napiše i brže rešenje (ako mu je ono *ikada zaista* i potrebno), ili da koristi nešto od prethodnog.
Još jedna razlika C u odnosu na Pascal, je što službene reči i imena moraju biti uvek napisana istom vrstom slova (mala ili velika slova). Na primer C promenljive *Pera*, *PERA*, *pera* i *peRA* tretira kao **potpuno različite**. Takođe ako napišete *INT a*; prevodilac će javiti grešku (treba *int a*;).

1.1.2. Osnovne C naredbe

Deklaracije:
Pre korišćenja promenljivih, moramo ih deklarisati.

```
tip a,b,c;
```

gde je *tip* neki od tipova (int, long int, float, double, char, ...), a posle toga idu nazivi promenljivih razdvojeni zarezima. Posle kraja svake naredbe ide tačka-zarez (;).
Na primer: `int a,b,c`

[1] XX veka

Učitavanje i štampanje:

Učitavanje sa tastature nekih celobrojnih promenljivih *a*, *b* i *c*:

```
scanf("%d %d %d",&a,&b,&c);
```

Oznake ***%d*** označavaju da se učitavaju celi brojevi (što je skoro i jedino moguće ako su promenljive deklarisane kao celobrojne), sa po jednim praznim mestom (blankom) između brojeva. Možemo učitavati:

```
scanf("Brojevi: %d, %d, %d",&a,&b,&c);
```

ali tada pri učitavanju moramo kucati i ***Brojevi:*** kao i zareze posle svakog unesenog broja. Dakle sve što smo stavili pod znacima navoda u *scanf* funkciji moramo kucati pri unošenju, jedino se ***%d*** tretira kao neki ceo broj.

Štampanje na ekranu:

```
printf("Broj s je: %d a broj t je %d\n",s,t);
```

Tekst unesen u formatu *printf()* funkcije, obeležen navodnicima, štampa se onako kako je napisan, a umesto znakova ***%d*** upisuju se redom vrednosti promenljivih (u našem slučaju *s* i *t*). Znak ***\n*** označava kraj tekućeg reda i štampanje u sledećem redu.

Pre svega toga, na početku programa, moramo uključiti zaglavlje *stdio.h*

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

da bi mogli da koristimo funkcije *scanf()* i *printf()*.

Glavni program:

Svaka funkcija (pa i glavni program), mora da se piše u obliku:

```
tip  ime_funkcije(tip1 par1, .....)
{  deklaracije

   naredbe
}
```

Glavni program se naziva *main*(), i u 99.99% slučajeva ne vraća ništa. U prvih nekoliko poglavlja on neće imati nikakve parametre:

```
void main()
{
   . . .
}
```

1. Učitavaju se dva cela broja. Naći im i odštampati zbir i proizvod.

```
#include <stdio.h>

void main(void)
{ int a,b,z,p;

scanf("%d %d",&a,&b);
z = a+b;
p = a*b;
printf("Zbir je: %d\n Proizvod je: %d\n",z,p);
}
```

ulaz:

```
2 3
```

izlaz:

```
Zbir je 5
Proizvod je 6
```

–

1.1.3. Osnovni tipovi u C

void Predstavlja prazan tip, koristimo ga ako moramo nešto da napišemo, a ne znamo šta drugo da stavimo.

int Predstavlja ceo broj i zauzima 2 ili 4 bajta memorije. Razlika je velika jer ukoliko zauzima 2 bajta opseg je $[-2^{15}, 2^{15} – 1] = [-32768 , 32767]$, a u suprotnom $[-2^{31}, 2^{31}–1] \approx [-$ 2 mild, 2 mild]. [2] Na žalost to (broj bajtova *int*) zavisi od implementacije C, i retko se unapred zna. Da bi rešili taj problem ili koristimo *long int* koji ima sigurno 4 bajta ili funkcija sizeof(*tip*) vraća broj bajtova nekog tipa (*sizeof*(*int*) ----> 2 ili 4).

long int Ako nismo sigurni koliko int ima bajtova a želimo veći opseg ($[-2^{31}, 2^{31}–1]$).

short int Uglavnom je isto što i int, pa ga uglavnom ne koristimo.

unsigned int Neoznačen ceo broj

unsigned long int Neoznačen ceo broj sa 4 bajta opseg: $[0, 2^{32}–1] \approx [0,$ 4 mild]

Namena ovog tipa uglavnom nije da predstavljamo pozitivne brojeve, što bi imalo smisla samo ako bi imali brojeve u opsegu [2 mild, 4 mild], a to ćemo imati jednom u 1000 godina, već da nas obezbedi od moguće greške kada radimo sa bitovima (negativni brojevi se pamte u potpunom komplementu).

float Realan broj (4 bajta). Opseg je do 10^{38}, u + i – delu. Međutim on broj ne pamti tačno već sa **7 sigurnih (značajnih)** cifara. Na primer broj 2345.71234 se pamti samo kao 2345.712, što ne mora biti dovoljno u nekim numeričkim izračunavanjima (u numeričkoj analizi koju god operaciju +,-,*,/ primenili greška se nagomilava – povećava).

double Realan broj (8 bajta). Opseg je do 10^{308}, u + i – delu, a tačnost je **15-16 sigurnih (značajnih)** cifara, što je u 99% slučajeva dovoljno i za najzahtevnija izračunavanja.

char Označava jedan karakter i standardno zauzima 1 bajt. Karakter se u C predstavlja kao kod tog karaktera u ASCII tabeli (*char* [-128, 127], *unsigned char* [0, 255]). U ASCII tabeli se nalaze slova engleske abecede, brojevi i specijalni znaci (. . . , ‘0’ = 48, ‘1’ = 49, ..., ‘9’ = 57, . . ., ‘A’ = 65, ‘B’ = 66, . . . , ‘Z’ = 90, . . . , ‘a’ = 97, ‘b’ = 98, . . . ,’z’ = 122, . . .). Jedino se pri učitavanju/štampanju radi nad samim karakterima, a sve ostale operacije se izvode nad njihovim kodovima. Zbog toga ako imamo male cele brojeve (u gornjem opsegu), a želimo da uštedimo memorijski prostor, za njihovo smeštanje možemo koristiti i ovaj tip (*char*). Iako to još uvek nije u standardu, noviji kompajleri C dozvoljavaju i mogućnost korišćenja UNICODE 2-bajtnog ili UNICODE 4-bajtnog standarda (gde je moguće koristiti sva slova svih jezika sveta, i razne specijalne znake).

Logički tip ne postoji, i umesto njega se koristi celobrojni tip (*int*, a ako želimo da uštedimo može i *char*). U tom slučaju vrednost nula predstvlja netačnu vrednost, a sve različito od nule je tačno. Takođe ne postoji skupovni tip, a i nabrojivi tip je sa mnogo manje mogućnosti u odnosu na Pascal.

Operacije:

aritmetičke	+ sabiranje, - oduzimanje, * množenje, / deljenje
poređenja	<, >, <=, >=, ==, != (različito)
dodele	=
logičke	&& ($\wedge$ konjunkcija), || ($\vee$ disjunkcija), ! ($\neg$ negacija)

kasnije će biti objašnjene

pokazivači	* &
bitovske op.	& | ^ ~ >> <<

1.1.4. Naredbe grananja i ciklusa

naredba grananja:
if(uslov) naredba1;
else *naredba2*;

Ako je ispunjen *uslov* izvršava se *naredba1*, a ako nije ispunjen *naredba2*.

višestruka naredba grananja:
switch(*izraz*)
{
case *vr1*: *naredbe1*;

[2] mild = milijarda = 10^9

break;
case *vr2*: *naredbe2*;
break;
. . .
default: *naredbe_ostale*;
}

Računa se vrednost *izraz* i ako je ona jednaka vrednosti *vr1*, izvršavaju se *naredbe1*, ako je jednaka *vr2* izvršavaju se *naredbe2*, itd. Ako vrednost izraza nije jednaka nijednoj od vrednosti (*vr1*, *vr2*, . . .), izvršavaju se *naredbe_ostale*.

ciklusi:
for(postavljanje; uslov; promena) naredba;

Na početku se izvrši postavljanje. Posle toga dokle god je ispunjen uslov ponavlja se *promena* i *naredba*. Kada *uslov* više ne bude ispunjen, izlazi se iz *for* ciklusa.

while(uslov) naredba;

Sve dok je ispunjen *uslov*, ponavlja se *naredba*.

do { naredba } **while**(uslov);

Kao i prethodna, samo se prvo izvršava *naredba*, a zatim se ispituje da li važi *uslov*.

U svim prethodnim konstrukcijama ako želimo više naredbi (umesto jedne) pišemo ih unutar vitičastih zagrada, na primer:

while(uslov) { naredba1; naredba2; . . . }

2. Učitava se prirodan broj *n*, a zatim *n* realnih brojeva u dvostrukoj preciznosti. Naći im i odštampati zbir i proizvod.

```
#include <stdio.h>

void main(void)
{ double br,z,p;
  int i,n;

scanf("%d",&n);
z = 0;
p = 1;
for(i=0; i<n; i++)
 {
   scanf("%lf",&br);
   z = z + br;  /* moze i z += br; */
   p = p * br;  /* moze i p *= br; */
 }
printf("%lf %lf",z,p);
}
```

Zadaci za vežbu:

3. Sa tastature se učitava prirodan broj *n*. Naći i odštampati prvih *n* Fibonačijevih brojeva ($f_1 = 1$, $f_2 = 1$, $f_{n+2} = f_{n+1} + f_n$).
Primer: n=6 1 1 2 3 5 8

1.2. Datoteke

Rad sa datotekama je neznatno teži od učitavanja sa tastature/štampanja na ekran. U ovom odeljku ćemo opisati takozvani sekvencijalni pristup datoteci. U tom slučaju, ako želimo 10.-ti podatak u datoteci moramo pročitati i prethodnih 9. Postoji i tzv. direktan pristup datoteci, gde odmah pristupamo podatku koji nam treba, na primer koristeći funkciju *lseek*(). Međutim takav pristup ima problem sa tekstualnim datotekama (koje su najčešće), i ne pojavljuje se u dosadašnjim ispitnim rokovima iz OP, pa ga nećemo ovde detaljnije razmatrati.

Datoteke moramo prvo deklarisati:
FILE *ul;

zatim otvoriti:

ul = fopen("*imedat*","...");

gde umesto ... može stajati:
r – datoteka za čitanje - ulazna
w – datoteka za pisanje – izlazna (ukoliko datoteka sa takvim imenom već postoji njen sadržaj se briše)
a – datoteka za pisanje – izlazna (nadovezuje se na postojeći sadržaj ako datoteka već postoji)
t - tekstualna (sadržaj se može čitati i eventualno čak i menjati-editovati u notepad-u)
b – binarna (sadržaj se teško može pročitati, a još manje menjati)

Oni od ovih znakova koji nisu međusobno kontradiktorni mogu se kombinovati (rt, rb, wt, wb, at, ab).

Učitavanje/štampanje je skoro isto kao i sa tastature/ekrana samo umesto

scanf("....", ...); ------> fscanf(ul," ....", ...);
printf("....", ...); ------> fprintf(izl," ....", ...);

Datoteku eventualno možemo i zatvoriti

fclose(ul);

mada ako to i ne uradimo, po završetku programa, operativni sistem sam zatvori sve datoteke koje je dati program otvorio. Jedini je izuzetak ako smo istu datoteku 2 puta otvarali, onda bar jednom između toga moramo i da je zatvorimo. Drugi razlog zašto imamo potrebu da zatvaramo datoteke, je zato što to vole vaši asistenti.

Ostale funkcije za rad sa datotekama:
rewind(ul) - vraća datoteku na početak, što je ekvivalentno sa zatvaranjem i ponovnim otvaranjem;
feof(ul) - da li je nastupio kraj datoteke?

Poslednju funkciju vrlo često koristimo u obliku:
while(!feof(ul)) - dok ne stignemo do kraja datoteke uradi zadati posao sa svakim od podataka

4. Iz datoteke "brojevi.txt" se učitava prirodan broj *n*, a zatim *n* realnih brojeva u dvostrukoj preciznosti. Naći i odštampati u datoteku "rezultat.txt" zbir i proizvod tih brojeva.

```
#include <stdio.h>

void main(void)
{ double br,z,p;
  int i,n;
  FILE *ul,*izl;

ul = fopen("brojevi.txt","rt");
izl = fopen("rezultat.txt","wt");

fscanf(ul,"%d",&n);
z = 0;
p = 1;
for(i=0; i<n; i++)
 {
   fscanf(ul,"%lf",&br);
   z = z + br;
   p = p * br;
 }
fprintf(izl,"%lf %lf",z,p);
fclose(ul);
fclose(izl);
}
```

5. Iz datoteke "brojevi.txt" se učitavaju realni brojevi u dvostrukoj preciznosti do kraja datoteke. Naći i odštampati u datoteku "rezultat.txt" zbir i proizvod tih brojeva.

```
#include <stdio.h>

void main()
{ double br,z,p;
  FILE *ul,*izl;

ul = fopen("brojevi.txt","rt");
izl = fopen("rezultat.txt","wt");
z = 0;
p = 1;
while(!feof(ul))
 {
   fscanf(ul,"%lf",&br);
   z = z + br;
   p = p * br;
 }
fprintf(izl,"%lf %lf",z,p);
fclose(ul);
fclose(izl);
}
```

6. Iz datoteke "ulaz.txt" se učitavaju realni brojevi u dvostrukoj preciznosti do kraja datoteke. Naći i odštampati na ekranu njihovu aritmetičku, geometrijsku, harmonijsku i kvadratnu sredinu.

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

void main()
{ double br,a,g,h,k;
  int n;
  FILE *ul;

ul = fopen("ulaz.txt","rt");
a = 0;
g = 1;
h = 0;
k = 0;
n = 0;
while(!feof(ul))
 {
   fscanf(ul,"%lf",&br);
   a = a + br;
   g = g * br;
   h = h + 1/br;
   k = k + br*br;
   n++;
 }
a = a/n;
g = pow(g, 1/n);
h = n/h;
k = sqrt(k/n);
printf("%lf %lf %lf %lf",a,g,h,k);
fclose(ul);
}
```

Gorepomenuti program lepo izgleda ali nije sasvim tačan.

- Problem je deljenje nulom u izrazima *1/br, a/n, n/h, k/n* i *1/n*, pa treba date slučajeve posebno ispitati;
- Zatim izraz *1/n* vrši **celobrojno** deljenje pa mu je rezultat skoro uvek nula (osim kada je *n=1*), pa je g skoro uvek 1 (proizvod na nulti stepen);
- Funkcija *pow()* ne može da izračuna stepen ako je osnova negativna a eksponent nije celobrojan.

Popravljena (tačna) verzija programa:

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

void main()
{ double br,a,g,h,k;
  int n,hdobro;
  FILE *ul;

ul = fopen("ulaz.txt","rt");
a = 0;
g = 1;
h = 0;
k = 0;
hdobro = 1;
n = 0;
while(!feof(ul))
 {
   fscanf(ul,"%lf",&br);
   a = a + br;
   g = g * br;
   if(br!=0) h = h + 1/br;
   else hdobro = 0;
   k = k + br*br;
   n++;
 }
if(n==0) printf("Datoteka je prazna!");
else
 {
   a = a/n;
   k = sqrt(k/n);
   printf("Aritmeticka: %lf Kvadratna: %lf",a,k);
   if(hdobro && h!=0) printf("Harmonijska: %lf",n/h);
   else printf("Harmonijska sredina ne postoji");
   if(g>=0) printf("Geometrijska: %lf", pow(g, 1./n));
   else if(n % 2 == 0) printf("Geometrijska ne postoji");
        else printf("Geometrijska: %lf", -pow(-g, 1./n));
 }
fclose(ul);
}
```

Osim datoteka C može da radi i sa uređajima, i to na skoro isti način kao i sa datotekama. Jedina razlika je što za datoteku mi sami biramo da li je ulazna ili izlazna, dok je svaki uređaj svojom prirodom ograničen samo na jednu od tih mogućnosti. Takođe uređaji se ne mogu/moraju deklarisati, otvarati, ni zatvarati, već se odmah koriste:

stdin - tastatura (ulazni)
stdout - ekran (izlazni)
stderr - standardni izlaz za poruke o grešci

Gorepomenutu konstrukciju da radimo dok nije kraj standardnog ulaza (tastature) pišemo:
while(!feof(stdin)).
Fizički prekidamo unos podataka pritiskom na CTRL+Z (DOS ili Windows) odnosno CTRL+D (UNIX).

Ukoliko treba odštampati nešto na standardni izlaz za poruke o grešci:
fprintf(stderr," ..... ", ...);

Zadaci za vežbu:

7. Kako promeniti program iz zadatka **5.** tako da učitavanje bude sa tastature a ne iz datoteke?

8. Iz datoteke "br.txt" učitavaju se prirodni brojevi do kraja ulaza (datoteke). U datoteku "f.txt" odštampati njihove faktorijele.

Ispitni zadaci za vežbu:
Jul 2002. 1. tok - 3. zad., April 2002. 3. tok – 1. zad.

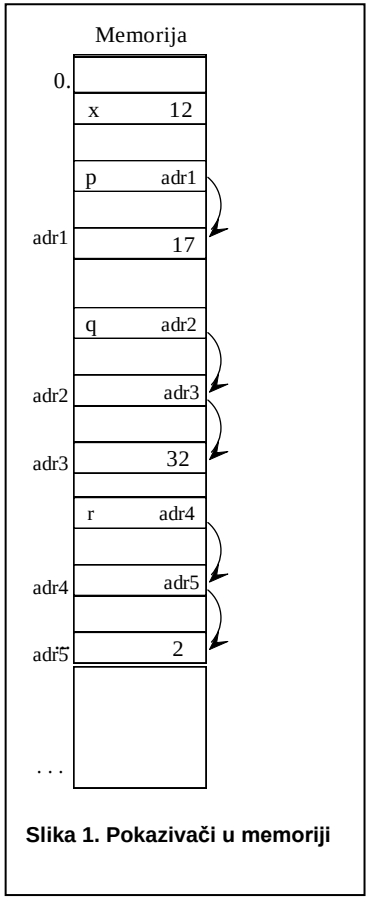
1.3. Pokazivači

1.3.1. Deklaracija pokazivača

int x, *p, **q,**r;

Uvodimo pojam ***nivo pokazivača***:

x – obična promenljiva (pokazivač 0. nivoa)
p – pokazivač 1. nivoa (ima jednu * ispred)
q - pokazivač 2. nivoa (ima dve * ispred)
r - pokazivač 2. nivoa (ima dve * ispred)



Slika 1. Pokazivači u memoriji

Svi podaci (promenljive) i prevedeni programi se smeštaju u memoriju, gde svaka lokacija u memoriji ima određenu adresu, kao na slici 1. Najbolje je ako je adresiranje bijekcija, pa svakoj adresi odgovara tačno jedna lokacija i obrnuto, i adrese idu od 0. adrese redom do kraja. Takvo adresiranje se naziva *linearno adresiranje* i implementirano je u operativnom sistemu UNIX. U Microsoft-ovim operativnim sistemima (DOS, Windows) adresiranje nije ni linearno ni bijekcija (neke adrese nemaju lokaciju, a neke lokacije imaju više od jedne adrese). To je i jedan od razloga njihovog finansijskog uspeha u prošlosti (kako više puta prodati jednu istu stvar pod drugim imenom i navodno poboljšanu). Međutim, u svim operativnim sistemima (pa i Microsoft-ovim) *adresiranje* mora biti **preslikavanje** odnosno *funkcija* (u matematičkom smislu), što znači da svakoj lokaciji u memoriji mora odgovarati bar jedna adresa. Pojam pokazivača predstavlja upravo tu adresu.

Na primer neka je dato stanje kao na slici 1. Promenljiva *x* je obična, pa sadrži neki ceo broj, neka je to recimo 12. Promenljiva *p* je pokazivač 1. nivoa, pa ona ne sadrži ceo broj, već neku adresu (adr1) lokacije gde se nalazi vrednost (recimo 17). Promenljiva *q* je pokazivač 2. nivoa, pa ona sadrži adresu (adr2) lokacije gde se nalazi adresa adr3 lokacije gde se nalazi vrednost (recimo 32). Grafički se to može prikazati strelicama kao na slici 1.

1.3.2. Operacije nad pokazivačima

Nad pokazivačima su definisane 2 operacije (***** i **&**), ali one nemaju uvek isto dejstvo, već njihovo dejstvo zavisi gde se oni nalaze u programu. Upravo to izaziva konfuziju i nerazumevanje kod onih koji se prvi put susreću sa programskim jezikom C. U tabeli 1. možemo videti njihovo dejstvo (oznaka **np +1** znači da se nivo pokazivača povećava za 1 a **np -1** da se smanjuje za 1.

U programskom jeziku C su deklaracije (tu ubrajamo i parametare funkcija i njihove promenljive) uvek pre svih izvršnih naredbi, kao što možemo videti u sledećem primeru u funkciji *racunaj()*:

```
int racunaj(int a, int b)
{ int i,j,n;
  float x,y;
  FILE *ul,*izl;
/* ovde se zavrsavaju sve deklaracije i pocinje izvrsni deo */
ul = fopen(....);
a = 1;
. . .
return n;
}
```

Napomena: U programskom jeziku C operacija **&** ne može da se pojavi u deklaracijama. U C++ je to promenjeno, pa se može **&** pojaviti i u deklaracijama, ali je njegovo značenje dosta drugačije i ne menja nivo pokazivača. Takođe u C++ ne moraju sve deklaracije biti pre svih izvršnih naredbi, već se mogu i preklapati, ali tako da je svaka promenljiva deklarisana pre nego što se koristi.

Tabela 1. Operacije nad pokazivačima

	*	&
deklaracije	**pokazivač** (adresa) **np +1**	----------
izvršni deo	***objekat*** ***np -1***	***adresa*** *(pokazivač)* ***np +1***

9. U konfiguraciji na slici 1. izvršiti sledeće promene (svaka promena se odnosi na početnu konfiguraciju, *a ne jedna za drugom*):
a) Na adresi *adr1* umesto vrednosti 17 upisati vrednost 22.
b) Postaviti pokazivač na adresi *adr2* tako da više ne pokazuje na *adr3* već da pokazuje na *adr1*.
c) Postaviti pokazivač *q* tako da više ne pokazuje na *adr2* već da pokazuje na *p*.
d) Promenljivoj *x* dodeliti vrednost koja piše na adresi adr5 (trenutno je ta vrednost 2, ali uraditi tako kao da, u opštem slučaju, ne znamo šta piše na adresi adr5).
e) Postaviti pokazivač *q* tako da više ne pokazuje na *adr2* već da pokazuje na *adr4*.

a) Pošto je adr1 povezan sa *p*, prvo što nam padne na pamet je da napišemo
```
  p = 22;
```
međutim leva i desna strana se ne slažu po tipu. Čak iako bi se leva i desna strana slagale po tipu, moramo ih uskladiti sa tipom operacije dodele koja se vrši. Pošto je ono što menjamo (a takođe i ono čime menjamo) brojna vrednost (pokazivač 0. nivoa), **operacija je 0. nivoa** pokazivača. Desna strana (22) se slaže, a leva strana *p* se ne slaže (*p* je pokazivač 1. nivoa) i zbog toga treba da ga smanjimo na 0. nivo primenom operatora * (pošto je ovo izvršni deo).
```
 *p = 22;
```

b) Pošto je adr2 povezan sa *q*, a adr1 sa *p*, prvo što nam padne na pamet je da napišemo
```
  q = p;
```
Ono što menjamo (adr2) je pokazivač 1. nivoa (a takođe i ono čime menjamo je 1. nivoa, jer treba da pokaže na adr1 koji je 0. nivoa). Zbog toga je **operacija 1. nivoa**. Desna strana (*p*) se slaže, a leva strana *q* se ne slaže (*q* je pokazivač 2. nivoa) i zbog toga treba da ga smanjimo na 1. nivo primenom operatora *.
```
 *q = p;
```

c) Prvo što nam padne na pamet je da napišemo
```
q = p;
```
Ono što menjamo (q) je pokazivač 2. nivoa (a takođe i ono čime menjamo je 2. nivoa, jer treba da pokaže na p koji je 1. nivoa). Zbog toga je **operacija 2. nivoa.** Leva strana (*q*) se slaže, a desna strana *p* se ne slaže (*p* je pokazivač 1. nivoa) i zbog toga treba da ga povećati primenom operatora &.
```
q = &p;
```

d) Pošto je adr5 povezan sa *r*, prvo što nam padne na pamet je da napišemo
```
x = r;
```
Ono što menjamo (promenljiva *x*) je broj odnosno pokazivač 0. nivoa (a takođe i ono čime menjamo je vrednost na adresi adr5). Zbog toga je **operacija 0. nivoa.** Leva strana (*x*) se slaže, a desna strana *r* se ne slaže (*r* je pokazivač 2. nivoa) i zbog toga treba da ga smanjimo na 0. nivo dvostrukom primenom operatora *.
```
x = **r;
```

e) Pošto je adr4 povezan sa *r*, prvo što nam padne na pamet je da napišemo
```
q = r;
```
Ono što menjamo (*q*) je pokazivač 2. nivoa (a takođe i ono čime menjamo). Zbog toga je **operacija 2. nivoa.** Obe strane (*q* i *r*) se slažu, pa nema nikakve promene
```
q = r;
```

za vežbu:

f) Na adresi *adr3* umesto vrednosti 32 upisati vrednost 19.
g) Postaviti pokazivač na adresi *adr4* tako da više ne pokazuje na *adr5* već da pokazuje na *adr3*.
h) Postaviti pokazivač *p* tako da više ne pokazuje na lokaciju *adr1* već da pokazuje na promenljivu *x*.
i) Na adresi *adr1* umesto vrednosti 17 upisati vrednost koja piše na adresi adr3.
j) Postaviti pokazivač *r* tako da više ne pokazuje na *adr4* već da pokazuje na *adr2*.
k) Postaviti pokazivač na adresi *adr2* tako da više ne pokazuje na *adr3* već da pokazuje na promenljivu *x*.
l) Na adresi *adr3* umesto vrednosti 32 upisati vrednost koja piše na adresi adr5.
m) Postaviti pokazivač *r* tako da više ne pokazuje na *adr4* već da pokazuje na *p*.
n) Postaviti pokazivač *p* tako da više ne pokazuje na lokaciju *adr1* već da pokazuje na *adr3*.
o) Na adresi *adr3* umesto vrednosti 32 upisati vrednost promenljive x.
p) Postaviti pokazivač na adresi *adr2* tako da više ne pokazuje na *adr3* već da pokazuje na *adr5*.
q) Na adresi *adr5* umesto vrednosti 2 upisati vrednost koja piše na adresi adr1.
r) Postaviti pokazivač na adresi *adr4* tako da više ne pokazuje na *adr5* već da pokazuje na promenljivu *x*.
s) Na adresi *adr1* umesto vrednosti 17 upisati vrednost promenljive x.
t) Postaviti pokazivač na adresi *adr4* tako da više ne pokazuje na *adr5* već da pokazuje na *adr1*.
u) Na adresi *adr5* umesto vrednosti 2 upisati zbir vrednost promenljive x i vrednosti koja piše na adresi adr3.
v) Postaviti pokazivač *p* tako da više ne pokazuje na *adr1* već da pokazuje na *adr5*.
w) Na adresi *adr3* umesto vrednosti 32 upisati proizvod vrednosti koje pišu na adresi *adr5* i adresi *adr1*.

1.4. Funkcije

Osim glavnog programa mogu se koristiti i druge funkcije. Ispred poziva neke funkcije moramo napisati tu celu funkciju ili njeno zaglavlje, da bi program znao da je ime funkcije korektno, kao i njeni parametri. Pri pisanju cele funkcije odnosno zaglavlja, kao i u slučaju glavnog programa prvo se mora navesti tip rezultata koji vraća data funkcija, zatim njeno ime i u malim zagradama parametri funkcije (tip parametra i njegovo ime). Posle toga ukoliko pišemo zaglavlje funkcije, stavljamo tačka-zarez (;), a ako pišemo celu funkciju njen zapis u vitičastim zagradama { ..... }. U pisanju zaglavlja funkcije se **ne moraju** (ali mogu) pisati imena parametara.

Parametri se u zapisu, pozivu i zaglavlju (ako ga imamo) funkcije moraju slagati po:

- broju
- tipu
- značenju

Prevodilac može proveriti slaganje po broju i tipu, i ako se ne slažu javlja grešku u fazi prevođenja. Ukoliko se parametri ne slažu po značenju, a značenje definišemo mi sami, prevodilac ne može da javi grešku, već će program raditi, ali pogrešno.
Podrazumevano je da se svaki parametar funkcije koji navedemo kopira. To znači da se vrednosti parametara mogu koristiti u funkciji, pa čak i promeniti, ali da promenjena vrednost **ne može biti vraćena**. Ukoliko ipak želimo da radimo nad originalom, odnosno da promenjenu vrednost parametra možemo i da vratimo, ispred tog parametra stavljamo *. Time se samo pravi još jedna kopija pokazivača koji pokazuje na isti objekat kao i originalni pokazivač. Time se može promeniti i objekat. Na primer u funkciji *radi*():
```
int radi(float a, int *b) { . . . }
```
će se kopirati parametar *a* i pokazivač *b*, ali se može raditi na objektu **b*.

Naredba *return* ima dve osobine:

- vraća vrednost funkciji (ako tip nije void)
- odmah se vraća u deo programa odakle je pozvana data funkcija (što znači da iza naredbe *return* ne sme ništa naći jer se to neće nikada izvršiti)

10. (isti kao 1.) Učitavaju se dva cela broja. Naći im i odštampati zbir i proizvod.
a) Korišćenjem 2 funkcije (jedna za računanje zbira i jedna za računanje proizvoda)
b) Korišćenjem samo jedne funkcije (i za zbir i za proizvod).

```
#include <stdio.h>

a) int zbir(int,int);
   int pr(int,int);
b) int zip(int, int, int*);

void main(void)
{ int a,b,z,p;

scanf("%d %d",&a,&b);

a) z = zbir(a,b);
   p = pr(a,b);
b) z = zip(a, b, &p);

printf("Zbir je: %d\nProizvod je: %d\n",z,p);
}

a)
int zbir(int x,int y)
{
  return x+y;
}

int pr(int a, int b)
{
  return a*b;
}

b)
int zip(int a, int b, int *p)
{
  *p = a * b;
  return a+b;
}
```

11. Date su funkcije:
```
void prva(int ***a, float *b, char c)   {     }
void druga(float *x, char ***y, int b)  {     }
void treca(char *y, float ***t, int *c) {     }
void cetvrta(float a, int **x, char *t) {     }
void peta(char **x, int ***y, float **z){     }
```

Ne vodeći računa o slaganju parametara po značenju (nije definisano u ovom zadatku), već samo po broju i tipu iz prve funkcije pozvati drugu i treću.
Napomena: Ovaj zadatak nema nikakvu realnu primenu, osim teorijskog učenja rada sa funkcijama i pokazivačima!

Pošto je u funkciji *druga*() prvi parametar tipa float 1. nivoa pokazivača, a u našoj (*prvoj*) funkciji je parametar *b* tipa float 1. nivoa, pa pišemo *b*.
U funkciji *druga*() drugi parametar je tipa char 3. nivoa pokazivača, a u našoj (prvoj) funkciji je parametar *c* tipa char 0. nivoa, pa podižemo nivo za tri i pišemo *&&&c*.
U funkciji *druga*() treći parametar je tipa int 0. nivoa pokazivača, a u našoj (prvoj) funkciji je parametar *a* tipa int 3. nivoa, pa smanjimo nivo za tri i pišemo ***a*.

```
void prva(int ***a, float *b, char c)
{ druga(b, &&&b, ***a);
}
```

Pošto je u funkciji *treca*() prvi parametar tipa char 1. nivoa pokazivača, a u našoj (prvoj) funkciji je parametar *c* tipa char 0. nivoa, podižemo nivo za jedan i pišemo *&c*.
U funkciji *treca*() drugi parametar je tipa float 3. nivoa pokazivača, a u našoj (prvoj) funkciji je parametar *b* tipa float 1. nivoa, podižemo nivo za dva i pišemo *&&b*.
U funkciji *treca*() treći parametar je tipa int 1. nivoa pokazivača, a u našoj (prvoj) funkciji je parametar *a* tipa int 3. nivoa, smanjimo nivo za dva i pišemo **a*.

```
void prva(int ***a, float *b, char c)
{ treca(&c, &&a, **a);
}
```

za vežbu:
Iz svake od funkcija pozvati sve ostale.

1.5. Nizovi

1.5.1. Statički nizovi

Deklaracije:
int x[50];

Ovim je deklarisan niz od najviše 50 elemenata, numerisanih od 0. člana do najviše 49. člana, kao na slici 2.

Korišćenje:
a = x[17];
x[i] = 5;

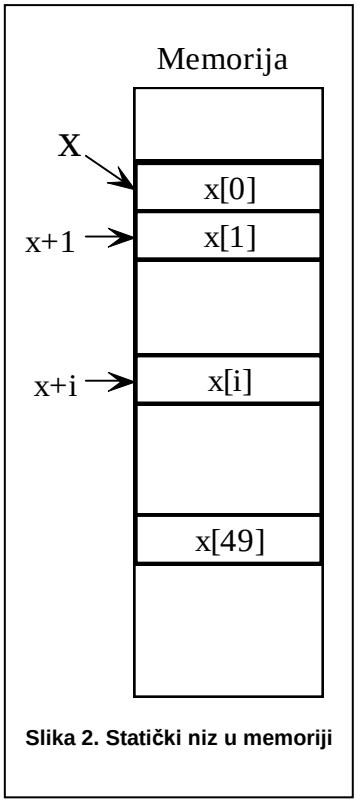
Niz je fizički gledano pokazivač na početak niza (bloka memorije). Blok memorije mora biti kompaktan (u jednom komadu). Zbog toga svi elementi idu jedan za drugim, pa važi:
x[0] <==> *x
x[1] <==> *(x+1)
x[2] <==> *(x+2)
. . .
x[i] <==> *(x+i)

Problemi koji se javljaju:
a) Broj članova mora unapred biti ograničen nekom konstantom
b) Ukoliko izađemo izvan opsega, prevodilac neće javiti grešku, već će se program izvršavati na pogrešan i nepredvidljiv način.

Primer: U gornjem primeru a = x[80]; nije toliko katastrofalno jer će samo promenljivoj *a* dodeliti neku vrednost izvan opsega niza (neku drugu promenljivu ili čak prevedeni kod nekog programa), ali nema uticaja na ostale promenljive (osim *a*). Za razliku od toga x[95] = 12; je mnogo opasnije pošto menja neki deo izvan opsega niza.

12. Iz datoteke brojevi.txt učitavaju se realni brojevi (ima ih najviše 7000) u dvostrukoj preciznosti do kraja ulaza. Sortirati uh u rastućem poretku i odštampati u datoteku sortiran.txt. Sortiranje implementirati u posebnoj funkciji.

```
#include <stdio.h>

void sort(double *a, int n)
{ int i,j;
  double pom;

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;
     }
}

void main()
{ double x[7000];
  int n,i;
  FILE *ul,*izl;

n = 0;
ul = fopen("brojevi.txt","rt");
izl = fopen("sortiran.txt","wt");
while(!feof(ul))
  fscanf(ul,"%lf",&x[n++]);
sort(x,n);
for(i=0;i<n;i++)
  fprintf(izl,"%lf ",x[i]);
fclose(ul);
fclose(izl);
}
```



Slika 2. Statički niz u memoriji

1.5.2. Dinamički nizovi

Korišćenjem dinamičkih nizova pokušavamo da ispravimo nedostatak statičkih nizova, da se maksimalna dimenzija mora unapred ograničiti konstantom. Oni se deklarišu kao pokazivači, a kasnije se funkcijom *malloc*() alocira (obezbeđuje) prostor u memoriji za dati niz. Pri alociranju i dalje moramo imati ograničenje za maksimalnu dimenziju (pošto memorija mora da bude u jednom komadu – bloku), ali to više ne mora biti konstanta, već može biti proizvoljna promenljiva ili izraz. Na kraju kada završimo korišćenje niza, možemo osloboditi dati prostor funkcijom *free*(). Kao i kod zatvaranja datoteka to najčešće nije neophodno (operativni sistem po završetku programa oslobađa svu memoriju koju je program zauzeo), ali nije loše pisati, jer to vole vaši asistenti. Dinamički nizovi kao i statički takođe imaju problem ako izađemo izvan opsega niza.

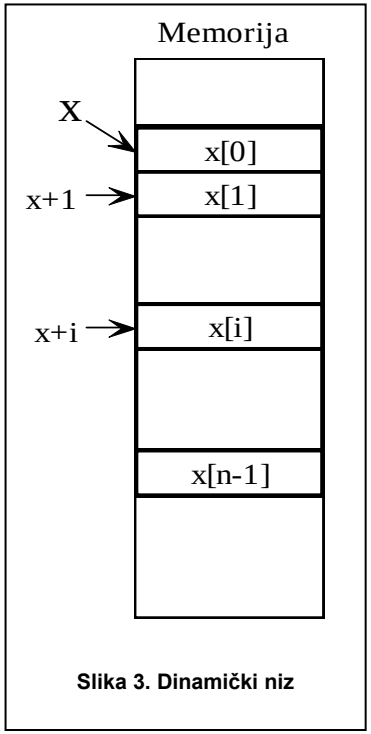
```
#include <stdlib.h>  /* zbog malloc() i  free() */
. . .
int *a;
. . .
a = malloc(n * sizeof(int));  /* pravimo niz od n clanova tipa int */
. . .
a[i] = 5;
. . .
free(a);
```

Napomena: **Promenljiva *a* fizički predstavlja niz** samo posle alociranja (*malloc*()) a pre oslobađanja (*free*()). Izvan toga ona je samo pokazivač.

13. Iz datoteke brojevi.txt učitavaju se realni brojevi u dvostrukoj preciznosti do kraja ulaza. Sortirati uh u rastućem poretku i odštampati u datoteku sortiran.txt. Sortiranje implementirati u posebnoj funkciji.
Napomena: Zadatak je skoro isti kao 12. ali nema ograničenje u dužini datoteke.

Rešenje sa dva čitanja datoteke:

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

void sort(double *a, int n)
{ int i,j;
  double pom;

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;
     }
}

void main()
{ double *x,pom;
  int n,i;
  FILE *ul,*izl;

n = 0;
ul = fopen("brojevi.txt","rt");
izl = fopen("sortiran.txt","wt");

while(!feof(ul))
  {
    fscanf(ul,"%lf",&pom);
    n++;
  }
x = malloc(n * sizeof(double));
rewind(ul);
for(i=0;i<n;i++)
  fscanf(ul,"%lf",&x[i]);
sort(x,n);
for(i=0;i<n;i++)
  fprintf(izl,"%lf ",x[i]);
fclose(ul);
fclose(izl);
free(x);
}
```



Slika 3. Dinamički niz

Zadaci za vežbu:

14. Iz datoteke brojevi.txt se učitavaju realni brojevi do kraja datoteke. Naći:

srednju vrednost $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

srednje kvadratno odstupanje $\overline{S_n^2} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})$.

15. Rešiti 13. zadatak tako da je čitanje sa tastature (ne može se dva puta čitati), a štampanje na ekranu.
Uputstvo: koristiti funkciju *realloc*() koja ukoliko popunimo sve elemente niza, može da alocira veći broj elemenata, i da niz premesti na tu lokaciju, a zatim polazni blok obriše. Jedino ako smo pamtili pokazivač na neki element u prethodnom bloku memorije, on primenom *realloc*() može postati "bajat".
Primer korišćenja funkcije realloc: a = realloc(a, m * sizeof(int));

rešenje:

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

void sort(double *a, int n)
{ int i,j;
  double pom;

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;
     }
}

void main(void)
{ double *x;
  int n,k,i;

n = 0;
k = 100;
x = malloc(k * sizeof(double));

while(!feof(stdin))
  {
    if(n>k)
     {
       k = 2*k;
       x = realloc(x, k * sizeof(double));
     }
    scanf("%lf",&x[n++]);
  }
sort(x,n);
for(i=0;i<n;i++)
  printf("%lf ",x[i]);
free(x);
}
```

Ispitni zadaci za vežbu:
Septembar 2003. 4. tok – 1. zad, Oktobar II 2000. - 1. zad., Novembar 2000. 2. tok - 1. zad., Jun 2003. 3-4 tok – 1. zad, April 2001. – 1. zad., Oktobar 2001. 1. tok – 1. zad., Januar 2003. (1-3 tok) 1a zad., Jun 2003. 2. tok – 2. zad (*teži*),

1.6. Matrice

1.6.1. Statičke matrice

Deklaracije:
int a[30][40];

Ovim je deklarisana matrica od najviše 30 vrsta (numerisanih od 0 do 29) i najviše 40 kolona (od 0 do 39).

Korišćenje:

```
t = a[17][12];
a[i][j] = 5;
```

Kao i kod statičkih nizova problemi koji se javljaju su:
a) Broj vrsta i kolona mora unapred biti ograničen nekom konstantom
b) Ukoliko izađemo izvan opsega, prevodilac neće javiti grešku, već će se program izvršavati na pogrešan i nepredvidljiv način.

16. Iz datoteke matrica.txt učitava se ceo broj n (n<=30), a zatim realna kvadratna matrica A_{nxn}. Sa tastature se učitava ceo broj k (k:>=0). Naći i odštampati na ekranu A^k.

```
#include <stdio.h>

void mnomat(int n, float a[30][30], float b[30][30], float c[30][30])
{ int i,j,k;

for(i=0;i<n;i++)
  for(j=0;j<n;j++)
   {
     c[i][j] = 0;
     for(k=0;k<n;k++) c[i][j] += a[i][k]*b[k][j];
   }
}

void main()
{ float a[30][30],b[30][30],c[30][30];
  int n,k,i,j,p;
  FILE *ul;

ul = fopen("matrica.txt","rt");
fscanf(ul,"%d",&n);
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
   {
     fscanf(ul,"%f",&a[i][j]);
     if(i==j) b[i][j] = 1;
     else b[i][j] = 0;  /* ili: b[i][j] = i==j ? 1 : 0; */
   }
scanf("%d",&k);
for(p=0;p<k;p++)
 {
    mnomat(n,a,b,c);
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
        b[i][j] = c[i][j];
 }
for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
     printf("%f ",b[i][j]);
   printf("\n");
 }
fclose(ul);
}
```

Napomena: Ovo nije najbrže rešenje, već služi samo za početak učenja rada sa matricama.

1.6.2. Dinamičke matrice

Recimo da želimo da napravimo matricu a_{mxn} (sa *m* vrsta i *n* kolona). To možemo uraditi na dva načina:
I način (koji mi ne koristimo), pomoću niza pokazivača na vrste

```
int **a;
. . .
a = malloc(m * sizeof(int *));
for(i=0; i<m; i++)
  a[i] = malloc(n * sizeof(int));
. . .
a[i][j] = 7;
. . .
for(i=0; i<m; i++)
  free(a[i]);
free(a);
```

II način (koji koristimo), matricu shvatamo kao "veliki" niz od m*n članova

```
int *a;
. . .
a = malloc(m * n * sizeof(int *));
. . .
a[i*n+j] = 7;  /* ili  *(a + i*n + j) = 7;
. . .
free(a);
```

Ako nam se ne sviđa način na koji je koristimo (pišemo), zapis možemo skratiti korišćenjem

```
#define a(i,j)  a[(i) * n + j]    ili    #define a(i,j) (*(a+ (i) * n + j))
        odnosno
#define mat(a,i,j)      (*(a+ (i) * n + j))
        odnosno
#define mat(a,i,j,n)    (*(a+ (i) * n + j))
```

Tada možemo skratiti i olakšati pisanje pa se tada gornja dodela piše kao:

```
a(i, j) = 7;
```

odnosno

```
mat(a, i, j) = 7;
```

odnosno

```
mat(a, i, j, n) = 7;
```

17. Iz datoteke matrica.txt učitava se ceo broj n, a zatim realna kvadratna matrica A_{nxn}. Sa tastature se učitava ceo broj k (k:>=0). Naći i odštampati na ekranu A^k.
Napomena: Zadatak je skoro isti kao 16. ali nema ograničenje za veličinu matrice.

```
#include <stdio.h>
#include <stdlib.h>
#define mat(a,i,j) (*(a+i*n+j))

void mnomat(int n, float *a, float *b, float *c)
{ int i,j,k;

for(i=0;i<n;i++)
  for(j=0;j<n;j++)
   {
     mat(c,i,j) = 0;
     for(k=0;k<n;k++) mat(c,i,j) += mat(a,i,k) * mat(b,k,j);
   }
}

void main()
{ float *a,*b,*c;
  int n,k,i,j,p;
  FILE *ul;

ul = fopen("matrica.txt","rt");
fscanf(ul,"%d",&n);
a = malloc(n * n * sizeof(float));
b = malloc(n * n * sizeof(float));
c = malloc(n * n * sizeof(float));
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
   {
```

```
      fscanf(ul,"%f",&mat(a,i,j));
      if(i==j) mat(b,i,j) = 1;
      else mat(b,i,j) = 0;
    }
scanf("%d",&k);
for(p=0;p<k;p++)
 {
    mnomat(n,a,b,c);
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
        mat(b,i,j) = mat(c,i,j);
 }
for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
     printf("%f ",mat(b,i,j));
   printf("\n");
 }
fclose(ul);
free(a);
free(b);
free(c);
}
```

Napomena: Ovo takođe nije najbrže rešenje, a bržu varijantu možete videti u rešenju: Januar 2001. - 1 zad – I način rešavanja.

Zadaci za vežbu:

18. Iz datoteke mat.txt učitavaju se celi brojevi *m* i *n*, a zatim realna pravougaona matrica A_{mxn}. Naći:
a) Maksimum i minimum svih elemenata matrice A.
b) Najmanji od svih maksimuma vrsta i najveći od svih minimuma kolona matrice A.
c) Pozicije i vrednosti elemenata koji su istovremeno najmanji u svojoj vrsti i najveći u svojoj koloni (ovi elementi se zovu sedla)
d) Sve tri norme matrice

$$||A||_1 = \max_{1 \le i \le m} \left(\sum_{j=1}^{n} | a_{ij} | \right) \qquad |A||_2 = \max_{1 \le j \le n} \left(\sum_{i=1}^{m} | a_{ij} | \right) \qquad ||A||_3 = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^{\,2}}$$

Ispitni zadaci za vežbu:

Septembar 2003. 1 tok – 2. zad, Januar 2002. 2. tok – 2. zad., Februar 2002. 1. tok – 1. zad., Februar 2002. 3. tok – 1. zad., April 2002. 1. tok – 1. zad, Jun 2002. 1. tok – 2. zad, Septembar 2003. 3. tok – 1. zad

1.7. Stringovi

1.7.1. Deklaracija

Pošto ne postoji tip string u C-u se koristi niz karaktera. Znak '\0' označava da se tu string završava (iako niz može imati više karaktera, preostali karakteri se ignorišu).

Deklaracije:
char s[70], *p, q[]=”Pera Mika”;

p = malloc(n);

q[0] = ‘P’;
q[1] = ‘e’;
q[2] = ‘r’;
q[3] = ‘a’;
q[4] = ‘ ‘;
. . .
q[9] = ‘\0’;

Korišćenje:
Stringove možemo koristiti na dva načina:
1. ručno – kao nizove karaktera
2. korišćenjem funkcija opisanih u <string.h>

Svaki zadatak se može rešiti ručno, gde radimo sa pojedinačnim karakterima u stringu, ali to ne mora biti najlakše rešenje, pošto neke funkcije za rad sa stringovima postoje u zaglavlju <string.h>

1.7.2. Funkcije za rad sa stringovima

strlen(s) - vraća dužinu stringa (ne računajući ‘\0’) *Primer*: strlen(q) = 9

Umesto dodele s1 = s2 (pogrešno) koristimo
strcpy(s1, s2) - kopira s2 u s1 ili
_strdup(s1, s2) - kopira s2 u s1 sa alokacijom *s1*

Umesto s1 = s1 + s2 (pogrešno) koristimo
strcat(s1, s2) - dodaje s2 na s1 *Primer*: strcat(q,”Zika”); ---> q je “Pera MikaZika”

Umesto s1 == s2 (pogrešno) koristimo
strcmp(s1, s2) - poredi s1 i s2 u leksikografskom poretku (kao u rečniku). Ako je s1 veći od s2 vraća se pozitivni broj, ako je s1 manji od s2 vraća se negativni broj, a ako su jednaki vraća se 0. Poređenje u leksikografskom poretku se vrši poređenjem prvih slova u stringovima. Ako je neko od prvih slova veće (kasniji odnosno veći kod u ASCII tabeli) taj string je veći. Tek ako su oba prva slova ista, ispituje se drugo slovo. Ako je i ono isto ide se dalje. Tek ako su sva slova ista stringovi su isti.
Primer: strcmp(“Mika,”Zika”) ---> negativan broj, jer je ‘M’ < ‘Z’
strcmp(“Mika”,”Mica”) --> pozitivan broj, jer je ‘k’ > ‘c’
strcmp(“Ana”,”Analiza”) --> negativan broj, jer je ‘\0’ < ‘l’
strcmp(“mika”,”Zika”) --> pozitivan broj, jer je ‘m’ > ‘Z’

strncpy(s1, s2, n) - kopira prvih *n* znakova s2 u s1 *Primer*: strncpy(s,”Zika”,2); ---> s je “Zi”
strncat(s1, s2, n) - dodaje prvih *n* znakova s2 na s1
Primer: strncat(q,”Zika”,2); ---> q je “Pera MikaZi”
strncmp(s1, s2, n) - poredi prvih *n* znakova s1 i s2.
Primer: strncmp(“Mika,”Mica”, 2) ---> 0, jer je “Mi” == “Mi”
strncmp(“Mika,”Mica”, 3) ---> pozitivan broj, jer je “Mik” > “Mic”

strupr(s1) - sve znake *s1* pretvara u velika slova. *Primer*: strcpy(s, strupr(”Zika”)); ---> s je “ZIKA”
strlwr(s1) - sve znake *s1* pretvara u mala slova. *Primer*: strcpy(s, strlwr(”Zika”)); ---> s je “zika”

strchr(s, c) - nalazi pokazivač na prvu pojavu karaktera *c* u stringu *s*. Ako *c* ne postoji u *s* vraća se NULL pokazivač. Ako želimo da dobijemo prvu poziciju karaktera *c* u stringu *s*, a ne pokazivač, tada oduzmemo *s*.
Primer: strchr(q,’a’) – q ---> 3 (prva pojava slova ‘a’ je 3 slova posle početka stringa *q*).
strchr(q,’u’) ----> NULL

strstr(s1, s2) - nalazi pokazivač na prvu pojavu stringa *s2* kao podreči u stringu *s1*. Ako *s2* ne postoji u *s1* vraća se NULL pokazivač.
Primer: strstr(q,”ra”) – q ---> 2 (prva pojava “ra” počinje 2 slova posle početka stringa *q*).
strchr(q,”mi”) ----> NULL

<stdlib.h>

atoi(s) - string s konvertuje u ceo broj, a ako to ne predstavlja jedan broj vraća 0.
Primer: n = atoi(“12”); ---> n = 12
m = atoi(“12.45a”); ---> m = 0 jer nije ceo broj

atof(s) - string s konvertuje u realan broj tipa double, a ako to ne predstavlja jedan broj vraća 0.
Primer: x = atof("12"); ---> x = 12
a = atof("12.45"); ---> a = 12.45;
b = atof("1.35E-2) ---> b = 0.0135
c = atof("+13.2+") ----> c = 0 jer nije realan broj

<stdio.h>

Postoji mogućnost čitanja/pisanja podataka iz/u string na isti način kao sa tastature ili datoteke pomoću f-ja **sscanf**() ili **sprintf**().
Primer: Neka *s* ima vrednost "12 13.5 7" sscanf(s,"%d %f",&n, &x) ---> n = 12, x = 13.5
sprintf(s, "Mika ima %d polozenih ispita", n); ---> s je "Mika ima 12 polozenih ispita"

1.7.3. Izdvajanje reči

Iako postoje i druge alternative, štampanje stringova uvek možemo uraditi koristeći funkciju *printf*(). Ako imamo: s1 je "Pera" s2 je "Laza" tada

printf("%s i %s su prijatelji",s1,s2)

štampa:
Pera i Laza su prijatelji

Ukoliko učitavamo stringove problem je što scanf() radi samo u manjem broju slučajeva. Na primer
scanf("%s %s",s1,s2);
na ulazne podatke daje sledeće rezultate:
Zika Joca ---> s1 je "Zika" s2 je "Joca" (kao sto i treba)

Zika Joca ---> s1 je "Zika" s2 je "" (ne valja)

Zika Joca ---> s1 je "" s2 je "" (ne valja)

Zika
Joca ---> s1 je "" s2 je "" (ne valja)

Zbog toga koristimo i funkcije:

c = **getchar**(); - čita jedan karakter sa tastature i smešta u *c*
c = **fgetc**(ul); - čita jedan karakter iz datoteke *ul* i smešta u *c*
gets(s) - čita ceo red sa tastature i smešta u string *s*
fgets(s, 120, ul) - čita ceo red (a najviše 120 karaktera) iz datoteke *ul* i smešta u string *s*

Vrlo često nam se javlja potreba da treba izdvojiti neke reči koje se sastoje od nekih simbola, a razdvojene su nekim drugim simbolima. Tada možemo iskoristiti sledeću funkciju:

```
void izdvojrec(FILE *ul, char *rec)
{ char c;
  int k;

c = fgetc(ul);
while( slova koja ne izdvajamo ) c = fgetc(ul);
k = 0;
while( slova koja izdvajamo )
 {
   rec[k++] = c;
   c = fgetc(ul);
 }
rec[k] = '\0'; /* zatvaramo izdvojen string */
}
```

Primer 1. Izdvojiti reči koje su razdvojene belinama (blankovima, tabulatorima, novim redovima).

```
void izdvojrec(FILE *ul, char *rec)
{ char c;
  int k;

c = fgetc(ul);
while(c == ' ' || c ==’\t’ && c!='\n' && !feof(ul) ) c = fgetc(ul);
k = 0;
while(c!=' ' && c!= '\t' && c!='\n' && !feof(ul))
 {
   rec[k++] = c;
   c = fgetc(ul);
 }
rec[k] = '\0';
}
```

Primer 2. Izdvojiti reči koje počinju slovom, a završavaju se belinama (blankovima, tabulatorima, novim redovima).

```
#define SLOVO(c) ((c>='a' && c<='z') || (c>='A' && c<='Z'))

void izdvojrec(FILE *ul, char *rec)
{ char c;
  int k;

c = fgetc(ul);
while(!SLOVO(c) && c!='\n' && !feof(ul) ) c = fgetc(ul);
k = 0;
while(c!=' ' && c!= '\t' && c!='\n' && !feof(ul))
 {
   rec[k++] = c;
   c = fgetc(ul);
 }
rec[k] = '\0';
}
```

Primer 3. Izdvojiti html etikete (tagove). Oni su oblika <*tag*>, na primer: <html>, </html>, <body>, </body>, <h1>, <h2>, <h3>, <title>, itd.

```
void izdvojrec(FILE *ul, char *rec)
{ char c;
  int k;

c = fgetc(ul);
while(c!='<' && c!='\n' && !feof(ul) ) c = fgetc(ul);
k = 0;
while(c!='>' && c!='\n' && !feof(ul))
 {
   if(c!='<') rec[k++] = c;
   c = fgetc(ul);
 }
rec[k] = '\0';
}
```

Primer 4. Izdvaja reč i broji samoglasnike

```
int izdvojrec(FILE *ul, char *rec)
{ char c;
  int k;
  int rez = 0;

c = fgetc(ul);
```

```
while(c==' ' || c=='\t' || c==',' || c=='.' || c=='?' || c=='!' && c!='\n' &&
!feof(ul) )
   c = fgetc(ul);
k = 0;
while(c!=' ' && c!='\t' && c!=',' && c!='.' && c!='?' && c!='!' && c!='\n' &&
!feof(ul))
 {
   if(SAMOGL(c)) rez++;
   rec[k++] = c;
   c = fgetc(ul);
 }
rec[k] = '\0';
return rez;
}
```

Primer 5. Izdvojiti reči iz stringa (a ne iz datoteke ili sa tastature) koje su razdvojene belinama (blankovima, tabulatorima, novim redovima).

```
void izdvojrec(char *red, char* rec, int *poz)
{int i,k;

  i = *poz;
  while(red[i]==' ' || red[i]=='\t' || red[i]==',') i++;
  k = 0;
  while(red[i]!=' ' && red[i]!='\t' && red[i]!='\0' && red[i]!=',')
    rec[k++] = red[i++];
  rec[k] = '\0';
  *poz = i+1;
}
```

Reč je moguće izdvojiti ne u string već u povezanu listu slova (Septembar 2003. 4. tok -2. zad ili April 2002. 1. tok – 2. zad), ali to možete pogledati posle odeljka 1.9. (kada uradimo povezane liste).

1.7.4. Parametri komandne linije

Do sada smo glavni program pisali bez parametara (argumenata)

void main() {. . .} ili void main(void) {. . .}

Postoji mogućnost da navedemo i neke parametre. Oni se navode ako program pozivamo iz komandne linije. Tada glavni program treba pisati

void main(int argc, char **argv) {. . .}

argc - broj parametara (argumenata)
argv[0] - ime programa koji izvršavamo
argv[1] – 1. argument (pamti se kao string)
. . .
argv[argc-1] - poslednji argument

Primer: Pri izvršavanju
uradi pera MIKA 34 Laza 2

argc = 6
argv[0] = “uradi”
argv[1] = “pera”
argv[2] = “MIKA”
argv[3] = “34”
argv[4] = “Laza”
argv[5] = “2”

Ako želimo da treći (takođe i peti) parametar dobijemo kao brojeve treba primeniti

atoi(argv[3]) ili atof(argv[3])

Treba obratiti pažnju da se unošenje parametara komandne linije vrši pri izvršavanju (posle prevođenja), pa programer ne zna unapred šta će korisnik uneti pri izvršavanju datog programa. Ukoliko korisnik ne unese ono što je programer očekivao, to može biti bezopasno (kao u zadatku 18), ali može biti i opasnije (ako treba da se unesu imena ulaznih/izlaznih datoteka, a korisnik ih ne unese).

19. Naći zbir brojeva iz komandne linije.

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

void main(int argc, char **argv)
{ int i;
  double z;

for(i=1; i<argc; i++)
  z = z + atof(argv[i]);
printf("%lf",z);
}
```

Ispitni zadaci za vežbu:
Januar 2002. 3. tok – 2. zad., Februar 2002. 2. tok – 2. zad., Februar 2002. 3. tok – 2. zad, Januar 2003. 1. tok – 3. zad., Septembar 2003. 2. tok – 1. zad, Februar 2003. 2-3 tok – 2. zad, Februar 2002. 1. tok – 2. zad., Jun 2001. 1. grupa – 2. zad, April 2002. 2. tok - 1. zad., Oktobar 2001. 3. tok – 2. zad (April 2002. 3. tok – 2. zad, Oktobar 2000. – 1. zad.), Jun 2003. 1. tok – 1. zad, Septembar 2003. 1. tok – 3. zad,

1.8. Strukture i nabrojivi tipovi

Nabrojivi tipovi:
Umesto da koristimo tip *int* nekim vrednostima možemo da dodelimo i opisna imena. Jedno veliko ograničenje je što ih učitavati/štampati možemo samo kao cele brojeve, pa tada ni njihovo korišćenje uglavnom nema baš mnogo smisla.

Primer:
typedef enum boja boja;
enum boja { crna, bela, zelena=10, crvena, plava };

. . .
boja x,y;

x = crna; /* x = 0; */
x = bela; /* x = 1; */
x = zelena; /* x = 10; */
if(x == crvena) /* if(x == 11) */
if(x != plava) /* if(x != 12) */

Strukture:
Za razliku od nabrojivih tipova čije korišćenje nije obavezno, strukture su mnogo značajnije. I do sada ukoliko smo imali podatke istog tipa, grupisali smo ih u niz (ili matricu). Sada možemo grupisati i podatke različitih tipova u jednu celinu (strukturu), koja sada predstavlja novi tip ravnopravan sa osnovim tipovima. Ukoliko želimo neki deo strukture tada iza promenljive koja označava ceo objekat, stavljamo tačku, a zatim ime dela strukture. Ukoliko promenljiva nije objekat već pokazivač tada stavljamo ->.
Primer:

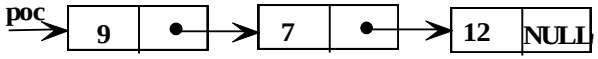
typedef struct student student;
struct student {
char ime[30], pr[30];
int gupisa, grod, brpol;
int ocena[40];
};

```
...

student a,b[20],*c,*p;

strcpy(a.ime, “Pera”);          /* ime postavljamo na Pera */
a.gupisa = 2001;                /* godinu upisa na 2001 */
a.grod = 1982;                  /* godinu rodjenja na 1982 */
a.brpol = 11;                   /* broj polozenih ispita na 11 */
a.ocena[0] = 7;                 /* prva (nulta) ocena mu je 7 */

printf(“%s”,b[5]);              /* stampamo ime 5. studenta */
b[7].brpol = 8;                 /* 7. student u nizu b ima 8 polozenih ispita */
b[7].ocena[7] = 10;             /* 7. ma (poslednja polozena) ocena mu je  10 */

c = malloc(n * sizeof(student)); /* alociranje dinamickog niza studenata c od n elemenata*/
c[0].brpol = 20;                /* prvi student u nizu c ima 20 polozenih ispita */
c[0].ocena[19] = 6;             /* poslednja polozena ocena mu je  6 */
free(c);

p->gupisa = 2000;               /* p pokazuje na studenta kome godinu upisa postavljamo na 2000*/
```

Ispitni zadaci za vežbu:
April 2000 – 1. zad., Septembar 2001. 1. tok – 2. zad.

1.9. Povezane liste

Pored nizova (statičkih, dinamičkih i vektora) i matrica, za smeštanje podataka možemo koristiti i povezane liste, gde za svaki član imamo pokazivač na sledeći član, osim poslednjeg (koji je jednak NULL), kao što je prikazano na slici 4.



Slika 4. Primer jednostruko povezane lista

Osim takvih moguće je i korišćenje dvostruko povezanih listi (svaki član ima pokazivač i na sledeći i na prethodni element), kružnih listi (poslednji član nema pokazivač NULL, već pokazuje na početak), dvostruko povezanih kružnih listi, itd. Međutim, za rešavanje zadataka iz OP-a koji su se do sada pojavljivali, uglavnom su bile dovoljne jednostruko povezane liste, pa ćemo obrađivati samo njih. Uostalom, rad sa ostalim tipovima listi je vrlo sličan, samo treba voditi računa o dodatnim pokazivačima.

Deklaracije:
Na primer ako želimo da u listu smestimo jedan ceo broj *n* i jedan realan broj *br*:

```
typedef struct lista lista;
struct lista
{ int n;
  float br;
  lista *sl;
};
```

Ova deklaracija u stvari predstavlja jedan član liste, a rad sa celom listom se kasnije obavlja u nekoj funkciji (može i u glavnom programu). Na taj način možda pišemo program malo duže (5-10%), ali je sve potpuno jasno, i lako se piše većina funkcija za rad sa listama.

1.9.1. Umetanje

Umetanje na početak liste:
Najlakši način umetanja u jednostruko povezanu listu je umetanje svakog novog elementa na početak liste. Ukoliko je data gorepomenuta lista, i pokazivač *poc* pokazuje na početak takve liste, tada će umetanje jednog elementa biti izvršeno sa:

```
pom = malloc(sizeof(lista));       /* pravimo novi element */
scanf("%d %f",&pom->n,&pom->br);   /* ucitavamo brojeve */
pom->sl = poc;                     /* prepovezivanje */
poc = pom;                         /* pom je novi pocetak liste */
```

Jedina mana ovog pristupa je što će elementi biti u listi raspoređeni u obrnutom poretku. Nekada nam je potreban baš takav način (recimo da želimo da odštampamo podatke okrenute naopačke), nekada nam je svejedno (kao u donjem zadatku gde posle unošenja sortiramo niz pa nije bitan poredak pri unošenju), a nekada je potrebno da sačuvamo poredak pri unosu.
Primer: Ako se unose brojevi 12, 7, 9 rezultat će biti lista kao na slici 4.

Umetanje na kraj liste:
Ukoliko je potrebno očuvati poredak pri unosu tada moramo umetati na kraj liste. Ukoliko je data gorepomenuta lista, pokazivač *poc* pokazuje na početak, a *kraj* na kraj takve liste, tada će umetanje jednog elementa biti izvršeno sa:

```
if(poc!=NULL)
 {
   kraj->sl = malloc(sizeof(lista));  /* dodajemo na kraj jos jedan el */
   kraj = kraj->sl;                   /* on je novi kraj */
 }
else
 {
   poc = malloc(sizeof(lista)); /*lista je bila prazna, pa ce imati 1 el.*/
   kraj = poc;                  /* koji je i pocetak i kraj */
 }
scanf("%d %f",&kraj->n,&kraj->br);        /* ucitavamo brojeve */
kraj->sl = NULL;                          /* posle kraja nema nista */
```

Ukoliko lista na početku nije bila prazna, već ima već neke elemente, a mi je dopunjujemo, pre umetanja je potrebno da joj nađemo *kraj*:

```
for(kraj=poc; kraj->sl!=NULL; kraj=kraj->sl);
```

a kasnije kod umetanja elemenata ne moramo da ispitujemo slučaj ako je prazna:

```
kraj->sl = malloc(sizeof(lista));  /* dodajemo na kraj jos jedan el */
kraj = kraj->sl;                   /* on je novi kraj */
scanf("%d %f",&kraj->n,&kraj->br); /* ucitavamo brojeve */
kraj->sl = NULL;                   /* posle kraja nema nista */
```

Oslobađanje cele liste:
Pošto smo listu pravili korak po korak (za razliku od niza ili matrice), moramo tako i da je brišemo:
I način:

```
for(pi=poc; pi!=NULL; )
 {
   pom = pi->sl;       /* pamtimo sledeci element */
   free(pi);           /* brisemo objekat od tekuceg elementa */
   pi = pom;           /* uzimamo zapamceni element */
 }
```

II način:

```
for(pi=poc; pi!=NULL; )
 {
   pom = pi;       /* pamtimo tekuci element */
   pi = pi->sl;    /* prelazimo na sledeci elemen */t
   free(pom);      /* brisemo objekat prethodno zapamcenog el. */
 }
```

Oslobađanje cele liste se uvek isto piše, a vi možete izabrati koji vam se od prethodnih načina više dopada.

20. Rešiti 13. zadatak tako da je čitanje sa tastature (ne može se dva puta čitati), a štampanje na ekranu.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct lista lista;
struct lista {
  float br;
  lista *sl;
};

void main(void)
```

```
{ lista *poc,*pom,*pi,*pj;
  float fpom;

poc = NULL;
while(!feof(stdin))
 {
   pom = malloc(sizeof(lista)); /* ubacivanje na pocetak liste */
   scanf("%f",&pom->br);
   pom->sl = poc;
   poc = pom;
 }
if(poc!=NULL) /* sortiranje */
  for(pi=poc; pi->sl!=NULL; pi=pi->sl)
    for(pj=pi->sl; pj!=NULL; pj=pj->sl)
      if(pi->br > pj->br)
       {
         fpom = pi->br;
         pi->br = pj->br;
         pj->br = fpom;
       }

for(pi=poc; pi!=NULL; pi=pi->sl)
  printf("%f ",pi->br);
for(pi=poc; pi!=NULL; ) /* oslobadjanje liste */
 {
   pom = pi->sl;
   free(pi);
   pi = pom;
 }
```

NULL pokazivač

Ukoliko neki pokazivač nema neku smislenu lokaciju na koju pokazuje, njegova vrednost je NULL.
U programiranju postoje dva vrlo važna pravila:

1. Od NULL pokazivača, u izvršnom delu programa, **ne smemo** praviti objekte sa ***** ni **->**, a ni **&** nije smislen.
2. Uvek moramo sprečiti beskonačnu rekurziju

Naravno da nam se nikada neće dogoditi da napišemo *NULL->br* (ni **NULL* ili *&NULL*). Međutim ako napišemo *p->br* (ili **p* ili *&p*), a pokazivač *p* ima vrednost NULL, odnosno *p* ne pokazuje na neku smislenu lokaciju, to će dati vrlo loše (ponekad i katastrofalne) rezultate.

Pravilo 2. može biti narušeno samo ako imamo rekurziju. Rekurzivno rešenje je u praksi **nekoliko puta sporije** (3-5 puta) od iterativnog načina rešavanja istog zadatka (for, while). Pošto je pri rekurzivnom rešenju još moguće i narušavanje pravila 2., a i idejno je teže od iterativnog rešenja, mi ćemo ga maksimalno izbegavati, i koristiti samo kada je to zaista neophodno. I kada ga eventualno koristimo, iscrpno ćemo proveriti da nije narušeno pravilo 2.

&	**0**	**1**	**|**	**0**	**1**
0	0	0	**0**	0	1
1	0	1	**1**	1	1

1.9.2. Brisanje

Brisanje prve pojave:

```
if(poc!=NULL)
  if(poc->br==br)
   {
     pom = poc->sl;
     free(poc);
     poc = pom;
   }
  else
   {
     for(pi=poc; pi!=NULL && pi->br!=br; pi=pi->sl)
       pom = pi;
     if(pi!=NULL)
      {
        pom->sl = pi->sl; /* pom->sl = pom->sl->sl; */
        free(pi);
      }
   }
```

Brisanje svih pojava:
I način – višestrukim ponavljanjem prethodnog slučaja:

```
int ima = 1;
while(ima)
  . . .
```

II način – modifikacijom prethodnog slučaja:

```
if(poc!=NULL)
 {
  if(poc->br==br)
   {
     pom = poc->sl;
     free(poc);
     poc = pom;
   }
  for(pi=poc; pi!=NULL; pi=pi->sl)
    if(pi->br==br)
     {
       pom->sl = pi->sl; /* pom->sl = pom->sl->sl; */
       free(pi);
       pi = pom;
     }
    else pom = pi;
 }
```

Ispitni zadaci za vežbu:
Januar 2002. 2. tok – 1. zad., Januar 2002. 3. tok – 1. zad., Jul 2002. 1. tok – 2. zad., Septembar 2001. 1. tok – 1. zad, Januar 2003. 2-3 tok – 3. zad., Septembar 2003. 4. tok – 2. zad, Februar 2003. 1. tok – 3. zad, Septembar 2003. 3. tok – 3. zad, April 2000 – 2. zad. (Septembar 2002. 3. tok – 1. zad), Septembar 2003. 4. tok – 3. zad, Oktobar 2002. 1. tok – 3. zad., April 2002. 1. tok – 2. zad., April 2002. 2. tok – 2. zad, Jun 2001. 2. grupa – 2. zad

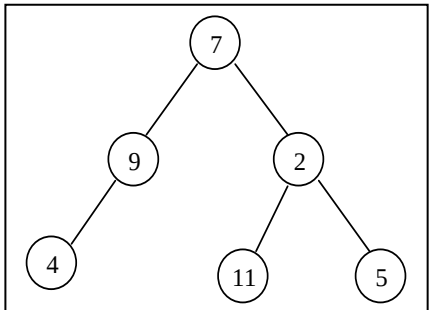
1.9.3. Implementacija steka i reda

1.9.4. Grafovi

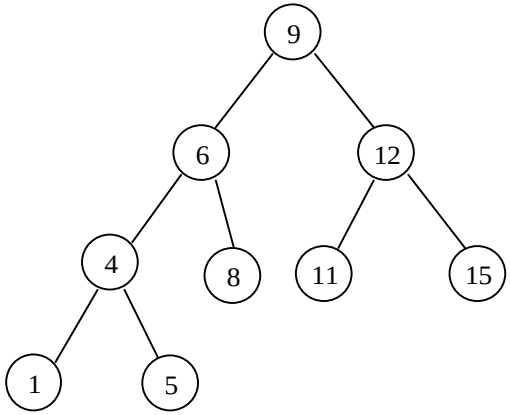
Ispitni zadaci za vežbu:
Oktobar 2000. 2. zad, Septembar 2001. 2. tok – 1. zad (Januar 2002. 1. tok – 1. zad), Septembar 2003. 1. tok – 1. zad,

^	0	1
0	0	1
1	1	0

1.10. Drveta



Slika 5. Neuređeno binarno drvo



Slika 6. Uređeno binarno drvo

1.10.1. Deklaracije

1.10.2. Neke operacije

21. Napisati funkcije koje dato binarno drvo štampaju u prefiksnom, infiksnom i postfiksnom zapisu, kao i funkciju koja vraća broj čvorova datog drveta.
Napomena: Smatramo da je drvo već napravljeno

1.10.3. Uređena binarna drveta

1.10.4. Neuređena binarna drveta

Ispitni zadaci za vežbu:

Januar 2002. 1. tok – 2. zad., Februar 2003. 2-3 tok – 1. zad, Jun 2003. 1. tok – 2. zad, Septembar 2001. 3. tok – 1. zad, Oktobar 2001. 3. tok – 1. zad, Septembar 2000. – 1. zad, Jun 2001. 1. grupa – 1. zad

1.11. Ostalo

1.11.1. Rad sa bitovima

Ukoliko imamo mnogo stanja sa samo nekoliko mogućih vrednosti, a želimo da uštedimo prostor, pa da ne trošimo ceo char (ili int), za memorisanje svakog stanja, tada koristimo rad sa bitovima.
Primer: Ako imamo sistem od 100 000 uređaja koji mogu biti samo u jednom od 2 stanja uključeno/isključeno, tada nam ne treba niz od 100 000 elemenata tipa *char*, već samo 12 500[3].

Pri radu sa bitovima važno je da koristimo neoznačene brojeve (*unsigned char* ili *unsigned int* a najbolje *unsigned long int*[4]) koji mogu biti samo pozitivni, jer negativni brojevi se memorišu u potpunom komplementu[5].

Operacije:
~x Komplement, gde se svaki bit u broju invertuje - menja u suprotni (1 <---> 0)
x & y Bitovska konjunkcija
x | y Bitovska disjunkcija
x ^ y XOR – ekskluzivna disjunkcija
Tablice ovih operacija možete videti na slici 6.
x >> n Pomeranje udesno *x* za *n* bit pozicija. Pri tome se *n* bit pozicija s desne strane brisu, a *n* bit pozicija s leve strane popunjavaju 0.
x << n Pomeranje ulevo *x* za *n* bit pozicija (inverzno od prethodnog).

Slika 6. Tablica operacija &, | i ^

Primer: Neka je
unsigned char x=21, y=19, z = 9; [6]

Tabela 2. Primer operacija nad bitovima

Izraz	binarna vrednost	dekadna vrednost
x	**00010101**	21
y	**00010011**	19
z	**00001001**	9
~x	**11101010**	234
~y	**11101100**	236
~z	**11110110**	246
x & y	**00010001**	17
x | y	**00010111**	23
x | y | z	**00011111**	31
x ^ y	**00000110**	6
(x & y) ^ z	**00011000**	24
x >> 2	**00000101**	5
y >> 3	**00000010**	2

[3] Jedan bit je dovoljan za smeštanje jednog uređaja, a jedan *char* ima 1bajt = 8 bitova.
[4] Pošto on ima 4 bajta = 32 bita.
[5] U tom slučaju mogli bi dobiti i ono što nismo želeli.
[6] *unsigned char* koristimo u ovom primeru zato što je najkraći (1 bajt = 8 bitova), a inače najčešće se koristi *unsigned long int* koji ima 4 bajta = 32 bita.

x << 1	**00101010**	42
z << 2	**00100100**	36
(x>>2) ^ (z<<2)	**00100001**	33

Kao što možemo videti:
~x ⇔ 255 – x = 2^8 – 1 – x (u slučaju *unsigned long int* ~x = 2^{32} - 1 - x)
x << k ⇔ x * 2^k ukoliko se ne javlja prekoračenje
x >> k ⇔ x / 2^k (celobrojno deljenje)

Pošto za neki bit B u bitovskoj aritmetici važi:
B & 1 = B, B & 0 = 0
B | 1 = 1, B | 0 = B
B ^ 0 = B, B ^ 1 = ¬B,
(B ^ A) ^ A = B ^ je inverzan samom sebi

tada ako želimo da:
- neke bitove postavimo na 0, a druge da sačuvamo ---> primenjujemo &
- neke bitove postavimo na 1, a druge da sačuvamo ---> primenjujemo |
- neke bitove invertujemo, a druge da sačuvamo ---> primenjujemo ^

Primer: Ukoliko je x = MMSSSMMS zapisan u binarnom sistemu jedan 8-bitni broj (*unsigned char*), i ako M označava bit koji želimo da promenimo, a S koji želimo da sačuvamo (ostaje isti), tada:
a) **x & 00111001** M želimo da postavimo na 0, S sačuvamo
b) **x | 11000110** M želimo da postavimo na 1, S sačuvamo
c) **x ^ 11000110** M želimo da invertujemo, S sačuvamo

21. Iz komandne linije (prvi argument) se uzima stanje *st* nekog sistema od 30 uređaja, u binarnom brojnom sistemu. Svaki uređaj može biti uključen/isključen (1 / 0). Učitavaju se sa tastature prirodni brojevi m, n, k, l (svi su <= 30).
a) Odštampati da li je 5. uređaj uključen?
b) Odštampati da li je m-ti uređaj uključen?
c) Odštampati koliko je ukupno uređaja uključeno, a koliko isključeno.
d) Uključiti n-ti uređaj (ako je već bio uključen ostaviti ga uključenog).
e) Isključiti k-ti uređaj (ako je već bio isključen ostaviti ga isključenog).
f) l-tom uređaju promeniti stanje (ako je bio uključen isključiti ga i obrnuto).
Posle operacija pod d)-f) odštampati novo stanje na ekranu u binarnom sistemu.

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

void main(int argc, char **argv)
{ unsigned long int st,clan;
  int i,m,n,k,l,duz,br1=0;

st = 0UL;
clan = 1UL;
duz = strlen(argv[1]);
for(i=0; i<duz; i++)
 {
    if(argv[1][i]=='1') st = st + clan;
    clan = clan << 1;         /* ili clan = clan * 2; */
 }

scanf(%d %d %d %d",&m,&n,&k,&l);

if( (st >> 5) & 1UL ) printf("5. uredjaj je ukljucen!\n");
else printf("5. uredjaj nije ukljucen!\n");

if( (st >> m) & 1UL ) printf("%d. uredjaj je ukljucen!\n",m);
else printf("%d. uredjaj nije ukljucen!\n",m);

for(i=0; i<30; i++)
  if( (st >> i) & 1UL ) br1++;
                        /* moze i:  br1 = br1 + ((st >> i) & 1UL); */
printf("Bitova 1 je %d a Bitova 0 je " ,br1,30-br1);

st = st | (1UL << n);
st = st & ~(1UL << k);

st = st ^ (1UL << l);

printf("Novo stanje: ");
for(i=0; i<30; i++)
 {
   if( (st >> i) & 1UL ) printf('1');
               /* moze i:  printf("%ld",(st >> i) & 1UL); */
   else printf('0');
 }
printf('\n');
}
```

22. Napisati funkciju unsigned *rightrot*(*unsigned x*, *unsigned n*) koja vraća vrednost *x* rotiranu udesno za *n* bit pozicija.

```
unsigned rightrot(unsigned x, unsigned n)
{ unsigned pom,nbit;

if(sizeof(unsigned) == 4) nbit = 32;
else nbit = 16;

n = n % nbit;
return (x << (nbit - n)) | (x >> n);
}
```

1.11.2. Pokazivači na funkcije

1.11.3. Bit polja

Ispitni zadaci za vežbu:
Januar 2003. (1.-3. tok) – 1b. zad,

1.12. Rešenja zadataka za vežbu

3. Fibonačijevi brojevi

```
#include <stdio.h>

void main(void)
{ int n,i;
  long int f,f1,f2;

scanf("%d",&n);
f1 = 1;
f2 = 1;
if (n==1) printf("1");
else printf("1 1");

for(i=3; i<=n; i++)
 {
    f = f1 + f2;
```

```
    f1 = f2;
    f2 = f;
    printf(" %ld",f);
 }
}
```

7. Jedina značajnija promena je umesto *feof*(*ul*) zameniti *feof*(*stdin*).

```
#include <stdio.h>

void main(void)
{ double br,z,p;
  int i,n;
  FILE *izl;

izl = fopen("rezultat.txt","wt");
z = 0;
p = 1;
while(!feof(stdin))
 {
   scanf("%lf",&br);
   z = z + br;
   p = p * br;
 }
fprintf(izl,"%lf %lf",z,p);
fclose(izl);
}
```

8. Faktorijel

```
#include <stdio.h>
void main(void)
{int i,br;
 long int f;
 FILE *ul,*izl;

ul = fopen("br.txt","rt");
izl = fopen("f.txt","wt");

while(!feof(ul))
 {
   fscanf(ul,"%d",&br);
   f=1;
   for(i=1;i<=br;i++)
     f = f * i;
   fprintf(izl,"%d ",f);
 }
fclose(ul);
fclose(izl);
}
```

9. Pokazivači:
f) Operacija je 0. nivoa pa je **q = 19;
g) Operacija je 1. nivoa pa je *r = *q;
h) Operacija je 1. nivoa pa je p = &x;
i) Operacija je 0. nivoa pa je *p = **q;
j) Operacija je 2. nivoa pa je r = q;
k) Operacija je 1. nivoa pa je *q = &x;
l) Operacija je 0. nivoa pa je **q = **r;
m) Operacija je 2. nivoa pa je r = &p;
n) Operacija je 1. nivoa pa je p = *q;
o) Operacija je 0. nivoa pa je **q = x;
p) Operacija je 1. nivoa pa je *q = *r;
q) Operacija je 0. nivoa pa je **r = *p;
r) Operacija je 1. nivoa pa je *r = &x;
s) Operacija je 0. nivoa pa je *p = x;
t) Operacija je 1. nivoa pa je *r = p;
u) Operacija je 0. nivoa pa je **r = x + (**q); /* zagrada i nije neophodna */
v) Operacija je 1. nivoa pa je p = *r;
w) Operacija je 0. nivoa pa je **q = (**r) * (*p); /* ovde zagrade nije lose napisati */

11. Funkcije i pokazivači:

```
void prva(int ***a, float *b, char c)
{ druga(b, &&&c, ***a);
  treca(&c, &&b, **a);
  cetvrta(*b, *a, &c);
  peta(&&c, a, &b);
}
void druga(float *x, char ***y, int b)
{ prva(&&&b, x, ***y);
  treca(**y, &&x, &b);
  cetvrta(*x, &&b, **y);
  peta(*y, &&&b, &x);
}
void treca(char *y, float ***t, int *c)
{ prva(&&c, **t, *y);
  druga(**t, &&y, *c);
  cetvrta(***t, &c, y);
  peta(&y, &&c, *t);
}
void cetvrta(float a, int **x, char *t)
{ prva(&x, &a, *t);
  druga(&a, &&t, **x);
  treca(t, &&&a, *x);
  peta(&t, &x, &&a);
}
void peta(char **x, int ***y, float **z)
{ prva(y, *z, **x);
  druga(*z, &x, ***y);
  treca(*x, &z, **y);
  cetvrta(**z, *y, *x);
}
```

14. Srednja vrednost i odstupanje

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

float svred(float*, int);
float skvod(float*, int, float);

void main(void)
{float *x,sv,sko,pom;
 int n,i;
 FILE *ul;

ul = fopen("brojevi.txt","rt");
n = 0;
while(!feof(ul))
 {
   fscanf(ul,"%f",&pom);
   n++;
 }
x = malloc(n * sizeof(float));
rewind(ul);
```

```
for(i=0;i<n;i++)
    fscanf(ul,"%f",&x[i]);

sv = svred(x,n);
sko = skvod(x,n,sv);
printf("Sr. vrednost je: %f\nSr. kv. odstupanje je: %f\n",sv,sko);
}

float svred(float *x,int n)
{int i;
 float sv;

sv=0;
for(i=0;i<n;i++)
  sv = sv + x[i];
sv = sv/n;
return sv;
}

float skvod(float *x, int n, float sv)
{ float sko;
  int i;

sko=0;
for(i=0;i<n;i++)
  sko = sko + (x[i]-sv) * (x[i]-sv);
sko = sko / n;
return sko;
}
```

18. Minimumi, maksimumi, sedla i norme matrice

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define mat(a,i,j) (*(a+i*n+j))

void main(void)
{ float *a,maks,min,maxvr,minkol,najmmaxvr,najvminkol,no1,no2,no3,zbir;
  int m,n,i,j,k,l;
  FILE *ul;

ul = fopen("mat.txt","rt");
fscanf(ul,"%d %d",&m,&n);
a = malloc(m * n * sizeof(float));
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
    fscanf(ul,"%f",&mat(a,i,j));

maks = mat(a,0,0);
min = mat(a,0,0);
no3 = 0;
for(i=0;i<m;i++)
 {
   zbir = 0;
   for(j=0;j<n;j++)
    {
      no3 = no3 + mat(a,i,j) * mat(a,i,j);
      zbir = zbir + fabs(mat(a,i,j)); /* za normu1 */
      if(mat(a,i,j) > maks) maks = mat(a,i,j);
      if(mat(a,i,j) < min) min = mat(a,i,j);
      if(j==0 || mat(a,i,j) > maxvr) maxvr = mat(a,i,j);
    }
   if(i==0 || zbir>no1) no1 = zbir;
   if(i==0 || maxvr < najmmaxvr) najmmaxvr = maxvr;
 }

no3 = sqrt(no3);

for(j=0;j<n;j++)
 {
   zbir = 0;
   for(i=0;i<m;i++)
    {
      zbir = zbir + fabs(mat(a,i,j)); /* za normu2 */
      if(i==0 || mat(a,i,j) < minkol) minkol = mat(a,i,j);
    }
   if(j==0 || zbir>no2) no2 = zbir;
   if(j==0 || minkol > najvminkol) najvminkol = minkol;
 }

printf("Maks = %f Min = %f \nNajm max vrsta = %f  Najv min kolona = %f\n
       Norma1 = %f Norma2 = %f Norma3 = %f",maks, min, najmmaxvr, najvminkol,
       no1, no2, no3);

for(i=0; i<m; i++)
  for(j=0; j<n; j++)
    {
      for(l=0; l<m; l++)
        if(l!=i && mat(a,i,j)<=mat(a,l,j)) break;
      for(k=0; k<n; k++)
        if(k!=j && mat(a,i,j)>=mat(a,i,k)) break;
      if(l==m && k==n) printf("Sedlo: (%d,%d) Vrednost: %f\n",i,j,mat(a,i,j));
    }
fclose(ul);
free(a);
}
```