Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za primjenjenu matematiku
Grupa *Računarska znanost*

AUDITORNE
i
LABORATORIJSKE VJEŽBE IZ

PROGRAMIRANJA

šk. god. 2004./2005.

U Zagrebu, veljača 2005.

1. Auditorne vježbe

BINARNI BROJEVNI SUSTAV

Brojevni sustav s bazom brojanja **B** ima znamenke **0, 1, 2, ... , B-1**
Npr. u dekadskom sustavu **B=10**, a znamenke su **0, 1, 2, ..., 8 i 9**.

Ako je baza **B=2** dobiva se **binarni brojevni sustav**, čije znamenke su **0** i **1**.
Iz engleskog **BI**nary digi**T** nastalo je ime za najmanju količinu informacije **BIT**.

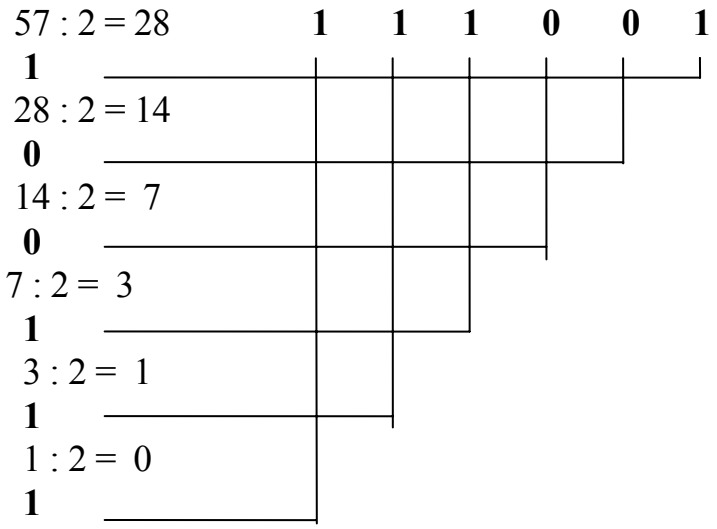
Primjer zapisivanja brojeva:
$\mathbf{57}_{10} = \mathbf{5} * 10^1 + \mathbf{7} * 10^0 = \mathbf{1}*2^5 + \mathbf{1}*2^4 + \mathbf{1}*2^3 + \mathbf{0}*2^2 + \mathbf{0}*2^1 + \mathbf{1}*2^0 = \mathbf{1\ 1\ 1\ 0\ 0\ 1}_{\ 2}$

Za binarno prikazivanje informacija je potreban najveći broj elemenata u usporedbi s ostalim prikazima, a broj bita za prikaz brojeva je iz tehničkih razloga ograničen .

Uređaji koji obrađuju i pohranjuju binarne informacije fizički se izvode pomoću elektroničkih elementa s 2 stabilna stanja (bistabil), koji su vrlo brzi i jeftini.

Pretvorba dekadskog broja u binarni

Binarni broj tvore ostaci dijeljenja s 2, odozdo prema gore



NEGATIVNI BINARNI BROJEVI

Registar je skup memorijskih elemenata koji pamte znamenke binarnog broja što znači da je za određeno računalo unaprijed propisana dužina registra, a time i brojevno područje unutar kojeg se kreću brojevi.

Primjer: Operacija 7 - 5 u računalu s registrom od 4 bita obavit će se kao 7 + (-5). Binarni prikaz broja -5 je sljedeći:

Pozitivni broj **0 1 0 1**

Komplement do baze-1 (jedinični komplement)		Komplement do baze (dvojni komplement)	
1 1 1 1		**1 0 1 0**	
0 1 0 1	-	**0 0 0 1**	+
1 0 1 0		**1 0 1 1**	

Dokaz da je dobiveni broj - 5

1 0 1 1 (- 5)
\+ **0 1 0 1** (+5)
0 0 0 0
Preljev **1**

Operacija oduzimanja 7 - 5

0 1 1 1 (7)
\+ **1 0 1 1** (-5)
0 0 1 0
Preljev **1**

U registru s 3 bita, ako je prvi bit predznak mogu se prikazati sljedeći brojevi:

Dekadski broj	Binarni broj
0	000
1	001
2	010
3	011
-4	100
-3	101
-2	110
-1	111

Za n = 3 dobije se interval $[-2^2, 2^2 - 1]$,
Za n = 8 taj je interval $[-2^7, 2^7 - 1]$, tj. [-128, 127].
Općenito: $[-2^{n-1}, 2^{n-1} - 1]$.

Dodavanjem jedinice najvećem prikazovom cijelom broju, dobit će se najmanji prikazivi cijeli broj (npr. za n=8 , 127 + 1 → 128) , odnosno oduzimanjem jedinice od najmanjeg prikazivog cijelog broja dobit će se najveći prikazivi broj (npr. za n=8 , 128 – 1 → 127)

OKTALNI BROJEVNI SUSTAV

Baza sustava je **B=8** a znamenke su **0, 1, 2, 3, 4, 5, 6, 7**.
Koristi se za skraćeno zapisivanje binarnih sadržaja kada je to spretno.

Primjer:

36-bitni broj	001	110	000	101	111	001	010	011	111	000	100	001
oktalni ekviv.	**1**	**6**	**0**	**5**	**7**	**1**	**2**	**3**	**7**	**0**	**4**	**1**

HEKSADEKADSKI BROJEVNI SUSTAV

Baza sustava je **B = 16**, a znamenke su **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**.
Koristi se za skraćeno zapisivanje binarnog sadržaja.

Primjer:

16-bitni broj	0111	1011	0011	1110
heksadekadski ekviv.	**7**	**B**	**3**	**E**

RAZLOMLJENI BINARNI BROJEVI

Razlomljeni binarni brojevi sadrže "binarnu točku", analogno decimalnom zarezu, odnosno točki u anglo-američkoj notaciji.

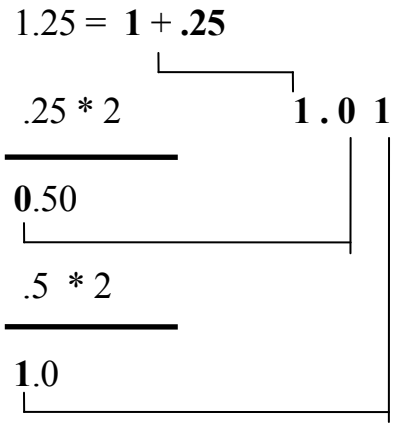
Primjer: prikaz razlomljenih brojeva

$$\mathbf{5.75}_{10} = \mathbf{5} * 10^{0} + \mathbf{7} * 10^{-1} + \mathbf{5} * 10^{-2} =$$

$$= \mathbf{1}*2^{2} + \mathbf{0}*2^{1} + \mathbf{1}*2^{0} + \mathbf{1}*2^{-1} + \mathbf{1}*2^{-2} = \mathbf{1\,0\,1\,.\,1\,1}_{2}$$

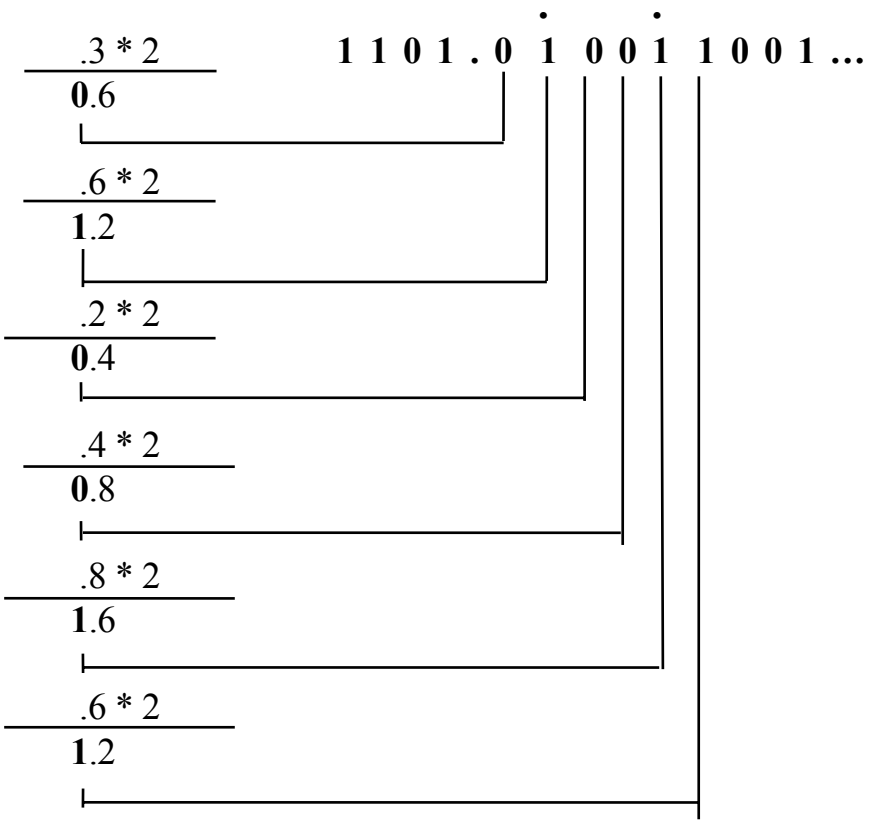
Pretvaranje decimalnog broja u binarni

Cjelobrojni dio dekadskog broja pretvara se u binarni uzastopnim dijeljenjem, a decimalni uzastopnim množenjem s 2, gdje cjelobrojni dio dobivenih produkata tvori znamenke binarnog razlomka.



Primjer: pretvaranje decimalnih brojeva koji se ne mogu prikazati konačnim brojem binarnih frakcija

13.3 = 1**3** + 0**.3**



....

Treba uočiti da se konačni decimalni razlomak prikazuje kao beskonačni periodički binarni razlomak.

Binarni broj se množi s potencijama baze 2 tako da se binarna točka pomakne odgovarajući broj mjesta desno ili lijevo, zavisno da li je predznak potencije pozitivan ili negativan.

Primjer: **1 . 1 1** * 2^2 = **1 1 1**

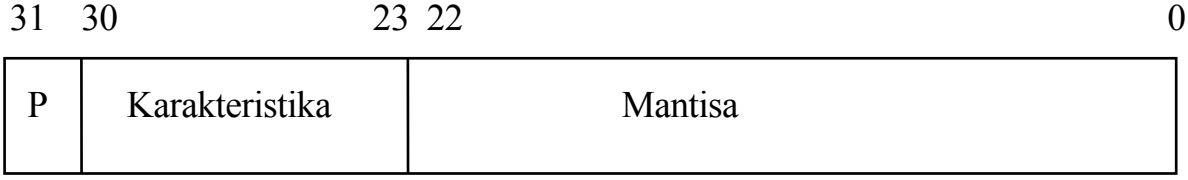
PRIKAZ REALNIH BROJEVA U RAČUNALU

Standardna točnost: 32 bita (4 byte)
Dvostruka točnost: 64 bita (8 byte)

Realni brojevi standardne točnosti

Deklaracija u programskom jeziku C: **`float`**

IEEE (Institute of Electrical and Electronics Engineers) standard 754 za prikaz realnih brojeva u standarnoj točnosti:



P	predznak (P=1 negativan, P=0 pozitivan)
Karakteristika	binarni eksponent + 127 (da se izbjegne prikaz negativnog eksponenta)
Mantisa	normalizirana (samo jedan bit ispred binarne točke).

Primjer: prikazati dekadski broj 5.75

$$\mathbf{5.75}_{10} = \mathbf{101.11}_2 * 2^0 = \mathbf{1.0111}_2 * 2^2$$

Kako se normalizacijom svakog binarnog broja (osim nule) postiže oblik 1.xxxxx, vodeća jedinica ne pohranjuje se u računalu i naziva se ***skrivenim bitom.***
Time se štedi jedan bit što povećava točnost.

Predznak = 0 (pozitivan broj)
Binarni eksponent = 2 ⟶ K = 2 + 127 = 129 = $(1000\ 0001)_2$
Mantisa (cijela) 1.0111
Mantisa (bez skrivenog bita) 0111

Rezultat: 0 10000001 01110000000000000000000
ili 0100 0000 1011 1000 0000 0000 0000 0000
4 0 B 8 0 0 0 0 (heksadekadski)

Primjeri:

2 = $10_2 * 2^0 = 1_2 * 2^1$ = 0100 0000 0000 0000 ... 0000 0000 = 4000 0000 hex
P = 0, K = 1 + 127 = 128 (10000000), M = (1.) 000 0000 ... 0000 0000

-2 = $-10_2 * 2^0 = -1_2 * 2^1$ = 1100 0000 0000 0000 ... 0000 0000 = C000 0000 hex
Jednako kao 2, ali P = 1

4 = $100_2 * 2^0 = 1_2 * 2^2$ = 0100 0000 1000 0000 ... 0000 0000 = 4080 0000 hex
Jednaka mantisa, BE = 2, K = 2 + 127 = 129 (10000001)

6 = $110_2 * 2^0 = 1.1_2 * 2^2$ = 0100 0000 1100 0000 ... 0000 0000 = 40C0 0000 hex

1 = $1_2 * 2^0$ = 0011 1111 1000 0000 ... 0000 0000 = 3F80 0000 hex
K = 0 + 127 (01111111).

.75 = $0.11_2 * 20 = 1.1_2 * 2^{-1}$ = 0011 1111 0100 0000 ... 0000 0000 = 3F40 0000 hex

Poseban slučaj - 0:

Normalizacijom broja 0 ne može se dobiti oblik 1.xxxxx

0 = 0 0000000 0000 ... tj. kao $1.0_2 * 2^{-127}$

Raspon i točnost realnih brojeva

Za slučaj realnog broja standardne točnosti karakteristika (8 bita) se može nalaziti u intervalu [0,255].

Specijalni slučajevi karakteristike su 0 i 255, pa obzirom da je BE = K - 127, BE se može kretati u intervalu [-126,127].

Kada je K = 0 i svi bitovi mantise nula radi se o broju nula.
Kada je K = 0 i postoje binarne frakcije u mantisi tada je to denormalizirani broj, tj. više ne postoji skriveni bit.
Kada je K = 255 i svi bitovi mantise nula radi se o prikazu $+\infty$ illi $-\infty$ ovisno od predznaka P
Kada je K = 255 i postoje binarne frakcije u mantisi ne radi se o prikazu broja (NaN)

Najmanji pozitivni broj $\neq 0$ koji se može prikazati je:

$$0.00000000000000000000001_2 * 2^{-126} \text{ što iznosi } 1.401298464324817*10^{-45}$$

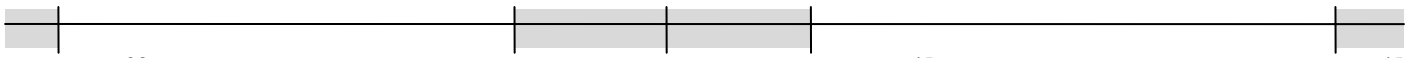
a najveći je:

$$1.11111111111111111111111_2 * 2^{127} \approx 2^{128} = 3.402823669209*10^{38}$$

Točnost: 24 binarne znamenke

$2^{24} \approx 10^x \rightarrow 24 \log 2 \approx x \log 10 \rightarrow x \approx 24 \log 2 = 7.224719895936$
tj. približno 7 prvih važećih točnih znamenki.

Prikaz pomoću brojevnog pravca:



$-3.4*10^{38}$ $-1.4*10^{-45}$ 0 $1.4*10^{-45}$ $3.4*10^{38}$

Numeričke pogreške

Nemogućnost korištenja svih bita kod računanja

Primjer: $0.0001_{10} + 0.9900_{10}$
0.0001_{10} : $(1.)10100011011011100010111_2 * 2^{-14}$
0.9900_{10} : $(1.)11111010111000010100011_2 * 2^{-1}$

Kod zbrajanje, binarne točke moraju biti poravnate:

```
  .00000000000011010001101 * 2^0   Samo 11 od 24 bita!
+.11111101011100001010011 * 2^0
-------------------------------
  .11111101011101110011000 * 2^0 =   0,9900999069214(10)
```

Realni brojevi dvostruke točnosti

Deklaracija u programskom jeziku C: **`double`**

63 62 52 51 0

P	Karakteristika (11 bita)	Mantisa

P predznak (P=1 negativan, P=0 pozitivan)
Karakteristika binarni eksponent + 1023 (11 bita)
Mantisa normalizirana (52+1 bit).

Raspon:

$K \in [0,2047]$.
$BE = K - 1023$
$BE \in [-1022,1023]$

Specijalni slučajevi: K=0 i K=2047

Kada je K = 0 i svi bitovi mantise nula radi se o broju nula
Kada je K = 0 i postoje binarne frakcije u mantisi tada je to denormalizirani broj, tj. više ne postoji skriveni bit
Kada je K = 2047 i svi bitovi mantise nula radi se o prikazu $+\infty$ ili $-\infty$ ovisno o predznaku P
Kada je K = 2047 i postoje binarne frakcije u mantisi tada se ne radi o prikazu broja (Nan)

Najmanji pozitivni broj različit od nule koji se može prikazati je:

$0.0000\ ...001_2 * 2^{-1022}$ što je $4.9406 * 10^{-324}$

a najveći je:

$1.1111.....111111_2 * 2^{1023} \approx 2^{1024} = 1.797693134862316*10^{308}$

Točnost: 53 binarne znamenke

$2^{53} \approx 10^x \rightarrow 53 \log 2 \approx x \log 10 \rightarrow x \approx 53 \log 2 = 15.95458977019$
tj. približno 16 prvih važećih točnih znamenki.

Postoji još i long double, međutim, njegova veličina ovisi o platformi, pa se tako mogu naći implementacije u kojima je njegova veličina 64,80,96 ili 128 bita. ANSI standard propisuje da nije manji od double broja. Primjer raspodjele ako je njegova veličina 80 bita:

long double 80 bita

Karakteristika: 15 bita
Binarni eksponent: Karakteristika – 16383

Realne konstante

1. 2.34 9e-8 8.345e+25	**double**
2f 2.34F -1.34e5f	**float**
1.L 2.34L -2.5e-37L	**long double**

2. Auditorne vježbe

CJELOBROJNI TIP PODATAKA U C–u

Cjelobrojni se tip podatka (`integer`), s obzirom na preciznost, može deklarirati kao `short` ili `long`.
U programskom jeziku C ne postoji ograničenje na duljinu `short`-a ili `long`-a, ali vrijede sljedeća pravila:

- `short` ne može biti dulji od `int`
- `int` ne može biti dulji od `long`

odnosno za duljinu može se napisati: `short ≤ int ≤ long`
U cjelobrojni tip podatka još pripada: `char` (kada predstavlja brojevnu, a ne znakovnu vrijednost).

VELIČINA OSNOVNIH TIPOVA PODATAKA

Veličina osnovnih tipova podataka, odnosno koliko jedna varijabla osnovnog tipa zauzima prostora u memoriji nije fiksna i ovisi o konkretnoj implementaciji prevodioca. Primjer razilaženja u interpretaciji veličine osnovnih tipova podataka je činjenica da su prevodioci tvrtke Borland koristili 16 bita za pohranu varijable tipa `int`, dok Microsoft koristi 32 bita za pohranu varijable tipa `int`.

Veličina osnovnih tipa podataka (prema MSDN Library za MS Visual C++ 6.0)

Tip	Veličina
`char, unsigned char, signed char`	1 oktet
`short, unsigned short`	2 okteta
`int, unsigned int`	4 okteta
`long, unsigned long`	4 okteta
`float`	4 okteta
`double`	8 okteta
`long double`[1]	8 (10) okteta

Dimenzije tipova nisu strogo propisane u C prevodiocima, pa je programeru na raspolaganju operator `sizeof`, koji vraća veličinu u oktetima navedenog parametra.
Operator `sizeof` se može koristiti i za određivanje veličine tipa podatka, ali i za određivanje veličine tipa pojedine varijable.

[1] Veličina long double tipa ovisna je o platformi i pojavljuju se izvedbe gdje je long double velik 8,10,12 ili 16 okteta.

Primjer:

```
int var;
printf("%d",sizeof(char)); // ispisuje 1
printf("%d",sizeof(var));  // ispisuje 4 (u MSVisualC++ 6.0)
```

PRIKAZ CIJELIH BROJEVA S PREDZNAKOM I BEZ PREDZNAKA

Kvalifikator `signed` (s predznakom) označava da se u varijabli može pohraniti pozitivna i negativna vrijednost.
Varijabla deklarirana kao `unsigned` (bez predznaka) može pohraniti samo pozitivne vrijednosti, što udvostručuje maksimalnu pozitivnu vrijednost koja u njoj može biti pohranjena u odnosu na `signed`.

Primjer:
Prikazati heksadecimalno sadržaj varijable tipa `short` u koju je pohranjena vrijednost -7.

```
7₁₀ = 111₂              0000000000000111
      (komplement)      1111111111111000
              +1                       1
                        1111111111111001
                        F    F    F    9
```

Primjer:
Broj **5** prikazan u 16-bitnom registru:

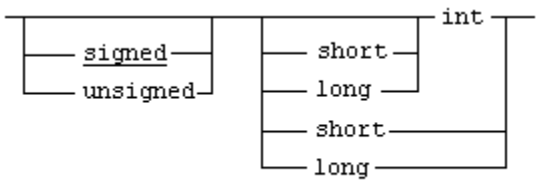
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Broj **–5** prikazan metodom dvojnog komplementa u 16-bitnom registru, gdje se vodeća jedinica uvjetno može smatrati predznakom broja (1→negativan broj):

1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1

Ako se isti niz binarnih znamenki pohrani u varijablu cjelobrojnog tipa **bez** predznaka, tada je vodeća jedinica **dio** podatka (**nije** predznak), te je na taj način predstavljen broj `65531`:

DEKLARACIJA CJELOBROJNOG (`int`) TIPA:



PRETVORBA TIPOVA PODATAKA

Tip rezultata aritmetičkih izraza određen je tipovima operanada u njemu. Kada se u izrazu nađe više različitih tipova podataka, tip rezultata ovisi o definiranim pravilima za pretvorbu. Pravila pretvorbe različitih tipova podataka u C-u orijentirana su prema višem tipu podataka. Dvije su vrste pretvorbi podataka u C-u:

- **automatska** (implicitna)
- **zadana** (eksplicitna)

Za **automatsku pretvorbu** općenito vrijedi da se varijabla (operand) podatkovnog tipa koji je manjeg raspona od podatkovnog tipa druge varijable (operanda) u izrazu interno pretvara u varijablu podatkovnog tipa većeg brojevnog raspona:

`short int` → `int` → `unsigned int` → `long int` → `unsigned long int` → `float` → `double` → `long double`

Tablica prikazuje standardnu automatsku pretvorbu jednog od operanada u tip drugog operanda u binarnim izrazima (prema MSDN Library za MS Visual C++ 6.0):

Tip operanda	**Automatska pretvorba**
Jedan od operanada je tipa **long double**.	Drugi operand se pretvara u **long double**.
Prethodni uvjet ne vrijedi, a jedan od operanada je tipa **double**.	Drugi operand se pretvara u **double**.
Prethodni uvjeti ne vrijede, a jedan od operanada je tipa **float**.	Drugi operand se pretvara u **float**.
Prethodni uvjeti ne vrijede, tj. niti jedan od operanada nije niti jednog realnog tipa.	Cjelobrojna pretvorba se obavlja prema sljedećim pravilima: • Ako je jedan od operanada tipa **unsigned long**, tada se i drugi operand pretvara u **unsigned long**. • Ako prethodni uvjet ne vrijedi, a jedan od operanada je tipa **long**, a drugi tipa **unsigned int**, tada se oba operanda

	pretvaraju u tip **unsigned long**. • Ako prethodni uvjeti ne vrijede, a jedan od operanada je tipa **long**, tada se i drugi operand pretvara u tip **long**. • Ako prethodni uvjeti ne vrijede, a jedan od operanada je tipa **unsigned int**, tada se i drugi operand pretvara u tip **unsigned int**. • Ako prethodni uvjeti ne vrijede, tada se oba operanda pretvaraju u tip **int**.

Napomena: pravila za pretvorbu cjelobrojnih i realnih tipova podataka mogu se razlikovati od prevodioca do prevodioca.

Primjer:

```
int a;
unsigned long b;
float f, g;
double d;

g = a + f;    // a se pretvara u float
d = a + b;    // a i b se pretvaraju u unsigned long, zbrajanje se
              // obavi u domeni unsigned long-a i rezultat tipa
              // unsigned long se pohrani u double
```

Primjer:

```
long lm = LONG_MAX, l;          // LONG_MAX = 2147483647
unsigned long um = ULONG_MAX; // ULONG_MAX = 4294967295
float f;

l = lm + 1;  // rezultat je: -2147483648 (preljev u neg. područje)
f = lm + 1;  // rezultat je: -2147483648.000000 (zbroj se obavio u
             // domeni long-a)

l = um + 1;  // rezultat je: 0 (preljev)
f = um + 1;  // rezultat je: 0.000000 (zbroj se obavio u domeni
             // unsigned long-a)
```

Zadana pretvorba podataka ima viši prioritet od automatske. Opći oblik deklaracije zadane (eksplicitne) pretvorbe (eng. *cast operator*) glasi:

```
(tip_podatka) operand
```

Operand može biti varijabla ili izraz.

Primjer:

```
a = (int) c;
b = (double)d + c;
```

CJELOBROJNO DIJELJENJE

Potrebno je obratiti pozornost na "neželjene" rezultate zbog cjelobrojnog dijeljenja. Ukoliko se na primjer u realnu varijablu `a` želi pridružiti vrijednost ½, sljedeća naredba pridruživanja **neće** raditi prema očekivanjima, tj. rezultat neće biti 0.5:

```
a = 1 / 2;
```

U izrazu s desne strane jednakosti **oba su operanda cjelobrojnog tipa**, pa će se obaviti cjelobrojno dijeljenje. Rezultat tog izraza je vrijednost 0 (uz ostatak 1).
Za izbjegavanje takvih neželjenih rezultata potrebno je koristiti zadanu pretvorbu tipa (dovoljno samo na jednom operandu) ili zadati konstante tako da je barem jedna realna:

```
a = (float) 1 / 2; ili
a = 1. / 2;        ili
a = 1 / 2.;
```

Napomena: U prvom slučaju korištena je zadana pretvorba tipa (mogla se uporabiti i nad drugim operandom), a u drugom i trećem slučaju, dodavanjem točke konstanta je predstavljena kao realna, te se dijeljenje obavlja u realnoj domeni.

PRIORITETI OPERATORA

1. `*` `/` `%`
2. `+` `-`

Ako u izrazu ima više operatora jednakog prioriteta, izračunavaju se slijeva nadesno. Izrazi u okruglim zagradama imaju najveći prioritet.

Primjer:
Koliki je rezultat sljedećeg izraza:

```
5 + 10 / 3 * ( 8 - 6 )
5 + 10 / 3 * 2
5 + 3 * 2
5 + 6
11
```

Primjer:
Koliko iznosi:
a) `9 / 4`
Rješenje: 2 (cjelobrojno dijeljenje)
b) `9 % 4`
Rješenje: 1 (ostatak cjelobrojnog dijeljenja 9 : 4 = 2 i ostatak 1)

Primjer:
Koju će vrijednost poprimiti varijable `i, x, c`

```
int i;
double x, c, d;
```

nakon naredbi:

```
d = 6.0;
i = (int)(d + 1.73);
x = i / 2;
c = (double)i / 2;
```

Rješenje:
i = 7, x = 3, c = 3.5

OPERATORI S BITOVIMA

& AND
| OR
^ XOR
<< SHIFT LEFT
>> SHIFT RIGHT
~ NOT

Operatori &, |, ^ su **binarni** (definirani nad dva bita), a operator ~ unarni (definiran nad jednim bitom).

Djelovanje bitovnih operatora (**b1** i **b2** predstavljaju bitove):

b1	**b2**	**b1 & b2**	**b1 | b2**	**b1 ^ b2**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

b1	**~b1**
0	1
1	0

Primjer:
Izračunati izraze 3 & 5, 3 | 5, 3 ^ 5, ~3

```
  0000 0011 (3)
& 0000 0101 (5)
  ---------
  0000 0001 (1)

  0000 0011 (3)
| 0000 0101 (5)
  ---------
  0000 0111 (7)

  0000 0011 (3)
^ 0000 0101 (5)
  ---------
  0000 0110 (6)

~ 0000 0011 (3)
  ---------
  1111 1100 (252 ili -4)
```

Operatori << i >> služe za pomak svih bitova vrijednosti varijable u lijevo ili u desno. Pomak bitova u varijabli za jedno mjesto u lijevo odgovara množenju vrijednosti varijable sa 2, dok pomak za jedno mjesto u desno rezultira dijeljenjem vrijednosti varijable sa 2.
Elektronička računala u skupu svojih strojnih naredbi u pravilu imaju naredbe za pomak vrijednosti u registru i tako izvršeno množenje ili dijeljenje s višekratnikom broja 2 bitno je brže u odnosu na klasično množenje i dijeljenje.
Broj pomaka u lijevo ili desno određen je parametrom.

Primjer:
Izračunati izraze 2 << 1, 37 >> 2

```
0000 0010 (2)
```
Nakon pomaka u lijevo za jedno mjesto, rezultat je umnožak vrijednosti s 2:
```
0000 0100 (4)
```

```
0010 0101 (37)
```
Nakon pomaka u desno za dva mjesta, rezultat je cjelobrojno dijeljenje s 4:
```
0000 1001 (9)
```

Oprez:
U slučaju da je podatak spremljen u jedan oktet bez bita za predznak, izračunati 128 << 1:

```
1000 0000 (128)
```
Nakon pomaka za jedno mjesto:
```
0000 0000 (0)
```

U slučaju da je podatak spremljen u jedan oktet sa bitom za predznak, izračunati 64 << 1:

```
0100 0000 (64)
```
Nakon pomaka za jedno mjesto:
```
1000 0000 (-128)
```

PRIKAZ ZNAKOVA U PROGRAMSKOM JEZIKU C

a) Pohrana malih cijelih brojeva
b) Pohrana slova, interpunkcija, posebnih znakova

Varijabla znakovnog tipa `(char)` može poprimiti vrijednosti takozvanih alfanumeričkih podataka, a za pohranu zauzima 1 oktet (*byte*). Alfanumerički znakovi su slova, brojevi, te posebni znakovi koji se mogu unijeti putem tipkovnice. Varijable znakovnog tipa mogu biti s predznakom `(signed char)` ili bez predznaka `(unsigned char)`.

Deklaracije u programskom jeziku C, te raspon brojeva koji je pokriven tipom:

`char`	`[-128, 127]`
`unsigned char`	`[0,255]`

Međunarodni standard: 7-bitni ASCII kod (American Standard Code for Information Interchange)

U programskom jeziku C znakovi su pohranjeni kao brojevi koji predstavljaju ASCII vrijednost (kod) navedenog znaka.
Najvažnije ASCII vrijednosti:

0 – znak NULL ('\0')
32 – praznina (' ')
48 – 57 – znamenke '0'-'9'
65 – 90 – velika slova 'A' do 'Z'
97 –122 – mala slova 'a' do 'z' (97 – 65 = **32** – razlika između malog i velikog slova!)

Podebljano su brojevi koje bi trebalo znati napamet.

ZNAKOVNE KONSTANTE

Znakovne konstante zadaju se unutar jednostrukih navodnika:

```
char a;
a = 'X';
```

Primjer:
Varijabli `c` tipa char pridružiti vrijednost slova A na različite načine:

```
c = 'A';
// ASCII kod znaka 'A' je 65₁₀ =  41₁₆ = 101₈
c = 65; ili c=x41; ili c=0101;
c='\x41';    // heksadecimalne konstante počinju s \x
c='\101';    // oktalne konstante počinju s \
```

Primjer:
Varijabli `c` tipa char pridružiti vrijednost jednostrukog navodnika ('), te znaka \

```
c='\'';  // specijalni znakovi unutar navodnika moraju imati ispred
         // sebe znak \
c='\\';
```

Kada se znakovni tip koristi za pohranjivanje znamenki treba obratiti pozornost na to da se u varijabli znakovnog tipa ne pohranjuje brojčana vrijednost te znamenke, nego ASCII vrijednost te znamenke, odnosno:

```
char a;
a = '1';  // ekvivalentno izrazu: a = 49;
```

U varijabli `a` nalazi se brojčana vrijednost 49, odnosno vrijednost znak '1' iz ASCII tablice. Ukoliko se želi dobiti brojčana vrijednost znamenke, potrebno je od te vrijednosti oduzeti 48. Vrijednost 48 ustvari predstavlja ASCII vrijednost znaka '0'.

Treba uočiti da pojedine znamenke prikazane kao ASCII znakovi ne odgovaraju binarnom prikazu tih znamenki kao cijelih brojeva.

Primjer:
varijabla a tipa char sadrži znamenku '7'. Pretvoriti tu vrijednost u broj.

Binarni sadržaj varijable a iznosi 0011 0111 (ASCII vrijednost 55_{10}).

```
char a = '7';
short int broj;

broj = a - 48; ili
broj = a - '0'; ili
broj= a & 0x0f;

0011 0111 (55₁₀)
0000 1111 (0x0f)
---------
0000 0111 (7₁₀)
```

Primjer:

```
char c ='A';
```

Što će se ispisati naredbama:

```
printf ("%c", c);              // A
printf ("%d", c);              // 65
printf ("%c", c + 32);         // a
printf ("%d", 'B' - 'A'));     // 1
```

Primjer:
Zadane su dvije varijable a i b tipa char koje sadrže znamenke ('0'-'9'). Napisati izraz koji će izračunati broj koji odgovara zbroju tih znamenki (npr. za '5' i '6' rezultat treba biti 11.)

```
char a,b;
int i;

i = a - '0' + b - '0';
ili
i = a + b - 2 * '0';
ili
i = a + b - 96; // 2 * 48
```

3. Auditorne vježbe

OPERATOR PRIDRUŽIVANJA

Operator pridruživanja u C-u je znak jednakosti (=). Potrebno ga je razlikovati od usporedbenog operatora za ispitivanje jednakosti koji je predstavljen dvostrukim znakom (==).
Operator pridruživanja obavlja pridruživanje vrijednosti izraza s desne strane (*r-value*) operandu s lijeve strane (*l-value*). Razlika između *l-value* i *r-value* je u tome što *l-value* operand mora imati dobro definiranu adresu i nakon izračunavanja izraza.

Može se napisati:

```
a = b + 3;
```

ali ne može

```
b + 3 = a;
```

U C-u postoji i **višestruko pridruživanje**.
Primjer

```
a = b = c = 0;
```

Sve tri varijable inicijaliziraju se na vrijednost 0. Prioritet pridruživanja je s desna na lijevo, tj. kao da je napisano:

```
a = (b = (c = 0));
```

Prvo se varijabli c pridružuje 0 i vrijednost tog cijelog izraza (`c = 0`) poprima vrijednost 0; zatim se varijabli b pridružuje vrijednost tog izraza, zatim cijeli taj izraz poprima vrijednost 0 koja se na kraju pridružuje varijabli a.

```
a = b = c + 3;
```

Može li?

```
a = b + 3 = c = d * 3;              // NE MOŽE !!!   →  b+3 nije l-value
```

U programiranju se često nova vrijednost varijable izračunava na temelju stare vrijednosti. Zbog toga u C-u postoje **skraćeni izrazi pridruživanja**.

Neki od operatora za skraćeno pridruživanje su: `+=, -= , *= , /= , %=`

Primjer
Izraz

```
i = i + 5;
```

može se pisati kao:

```
i += 5;
```

Izraz

```
i = i / (a + b);
```

može se pisati kao:

```
i /= a + b;
```

OPERATORI POVEĆANJA I SMANJENJA

U C-u postoje dva operatora za povećanje i smanjenje vrijednosti varijabli. Razlog njihovog postojanja je efikasnije izvršavanje (postoje posebne procesorske instrukcije koje vrlo brzo obavljaju te operacije) .

- prefiksni (operator ispred varijable)
  `++a; --a;`

- postfiksni (operator iza varijable)
  `a++; a--;`

Pri korištenju operatora ++ ili -- uz jednostavne numeričke tipove (varijante *char, int* i realne tipove), uvećanje i umanjenje se obavlja za jedan. Ovi operatori su korisni pri radu s indeksima polja, pokazivačima i brojačima unutar petlje.

Preporuča se korištenje prefiksnih operatora zato jer su efikasniji (kod postfiksnih operatora se mora kreirati privremena varijable u koju se pohranjuje stara vrijednost operanda koja se koristi u cijelom izrazu)

Primjer:

```
a = 5;
b = ++a * 2;
c = b++;
```

Nakon izvođenja ovih naredbi a ima vrijednost 6, b ima vrijednost 13, a c ima vrijednost 12

Izbjegavati dvoznačnosti oblika:

```
a = fn1( i++ ) + fn2( i++);
```

jer se u tom primjeru ne može reći koja će se funkcija prva obaviti.

Primjer: `int a=5; printf("c1=|%d|\n", ++a * --a)` -> MS Visual Studio će ispisati 25, a gcc 30.

USPOREDBENI OPERATORI

Usporedbeni operatori koriste se za uspoređivanje podataka. Ti podaci mogu biti konstante, varijable ili neki izrazi. U programskom jeziku C postoje sljedeći usporedbeni operatori :

Operator	Opis
==	jednako
>	veće
<	manje
>=	veće ili jednako
<=	manje ili jednako
!=	različito

Usporedbeni operator kao rezultat daje istinu (vrijednost različita od 0) ili laž (nula).

U programskom jeziku C ne postoji logički tip podatka, već se logičke vrijednosti prikazuju pomoću numeričkih vrijednosti.

KONTROLNA NAREDBA IF

Usporedbeni operatori obično se uključuju u izrazima kontrolnih naredbi. Ovisno o rezultatu usporedbe naredba se izvodi ili se ne izvodi. U C-u postoje tri oblika kontrolne naredbe `if`.

1) jednostrana selekcija
2) dvostrana selekcija
3) višestrana selekcija

Primjer:
Što će se ispisati sljedećim blokom naredbi za zadane vrijednosti varijabli:

1.) `a = 25` i `b = 4`.
2.) `a = 0, b = 0`

```
if (a = b)
  printf ("Vrijednost varijabli je jednaka\n");
else
  printf ("Vrijednost varijabli nije jednaka\n");
```

Rješenje:

ad 1) Za `a = 25` , `b = 4` , nakon pridruživanja `a = b`, a postaje `4` i cijeli izraz poprima vrijednost 4. Budući da se blok naredbi iza `if`-a obavlja ako je uvjet istinit (odnosno različit od 0 – 4 <> 0), ispisat će se: `"Vrijednost varijabli je jednaka"`.

ad 2) Nakon pridruživanja cijeli izraz ima vrijednost 0 pa će se ispisati: `"Vrijednost varijabli nije jednaka"`.

Pitanje: Što bi se ispisalo u prvom i drugom slučaju kada bi uvjet glasio `(a == b)`?
Odgovor: Budući da se radi o logičkom operatoru usporedbe (a ne o pridruživanju), ispisuje se:

ad 1) `"Vrijednost varijabli nije jednaka"`
ad 2) `"Vrijednost varijabli je jednaka"`

Važno je napomenuti: `if (a = b) ...` je ekvivalentno: `a = b; if (a) ...`.

Primjer:
Napisati programski odsječak koji će zbrojiti int varijable a i b ako je sadržaj char varijable c jednak '+', oduzeti ako je sadržaj varijable c jednak '-', a ispisati poruku o pogrešci ako varijabla c sadrži neki treći znak. Rezultat pohraniti u varijablu r.

```
int  a,b,r;
char c;

if( c == '+' )
   r = a + b;
else if( c == '-')
   r = a - b;
else
   printf("Pogrešna operacija");
```

Primjer:

Napisati programski odsječak koji će izračunati presjek dvije dužine na pravcu realnih brojeva (pretpostavlja se da se prva dužina na pravcu nalazi lijevo, a druga dužina desno)

```
int a1, a2, b1, b2;           // dužine [a1, a2] i [b1, b2]
int r1, r2;              // rezultat [r1, r2]

if( a2 > b1 ) /* znači da se sijeku */
  if( a2 > b2 )    /* znači da je cijela dužina B unutar dužine A */
  {
    r1 = b1;
    r2 = b2;
  }
  else
  {
    r1 = b1;
    r2 = a2;
  }
```

Obratiti pažnju na to da else pripada drugoj if naredbi, a ne prvoj. Ako else dio treba pridružiti prvoj (vanjskoj) if naredbi, mora se cijeli blok iza prve if naredbe staviti u vitičaste zagrade.

UVJETNI OPERATOR

Uvjetni operator (`?:`) je ternarni operator (zahtjeva tri operanda), a koristi se u pojedinim situacijama umjesto if-else naredbi. Oblik uvjetnog operatora je :

uvjetni_izraz ? izraz1 : izraz2;

Ako je *uvjetni_izraz* istinit, izvodi se *izraz1* te cijeli izraz poprima vrijednost tog izraza. Ako je lažan izvodi se *izraz2*, a cijeli izraz poprima tu vrijednost.

Primjer

```
int    a, b, c;
...
c = a > b ? a : b; // ako je a>b "obavi" a inače b
```

PRETPROCESORSKE NAREDBE

Prije procesa prevođenja izvornog kôda u strojni jezik, pokreće se tzv. pretprocesor. Ovisno o pretprocesorskim naredbama, pretprocesor mijenja izvorni kôd te se takav kôd prevodi (promjena se obavi u privremenu datoteku, a datoteka s izvornim kôdom ostaje nepromijenjena). Neke od naredbi pretprocesora su:

#include	uključuje neku drugu datoteku u program
#define	definira simboličku konstantu ili tzv. makro

```
#include     <stdio.h>
#include     "MojeFunkcije.h"
#define    PI              3.14159
#define    IME_PROGRAMA    "Program"
```

Pomoću "`define`" naredbe preptrocesora mogu se napraviti tzv. makro-i (podsjećaju na funkcije, ali nisu funkcije)

Primjer

Koju vrijednost će poprimiti varijabla **b** sljedećim naredbama?

```
int a, b;
#define MAX(x,y) ((x) > (y)) ? (x) : (y)
a = 3;
b = MAX (3, a++);
/*  Pretprocesor ovaj redak fizički zamijeni sljedećim retkom:
b = ((3) > (a++)) ? (3) : (a++);
   Na kraju su vrijednosti
a = 5, b = 4  */

a = 3;
b = MAX (a++, 3);
/*  Pretprocesor ovaj redak fizički zamijeni sljedećim retkom:
b = ((a++) > (3)) ? (a++) : (3);
   Na kraju su vrijednosti
a = 4, b = 3       (3++ > 3) ? 3++ : 3*/
```

LOGIČKI OPERATORI

Jednostavni relacijski izrazi mogu se kombinirati u složene pomoću logičkih operatora. C uključuje 3 logička operatora:

Operator	Značenje
&&	logičko I
||	logičko ILI
!	logičko NE

Uočiti razliku između logičkih operatora i logičkih operatora nad bitovima!

Primjer:

```
if ((x>20) && (x<100)) printf("x se nalazi u otvorenom intervalu 20-100");

if ((x<5) || (x>20)) printf("x se ne nalazi u zatvorenom intervalu 5-20");

if (!(x>20)) printf("x je manji ili jednak 20");
```

Primjer:

Što će se ispisati sljedećim programskim odsječkom?

```
int a, b, c, d;
a = 0;
b = 4;
c = (a++) + b;                                  /*(a++) +b      0+4   4*/
printf ("a = %d, b = %d,c = %d " , a, b, c);    /*  1, 4, 4   */
d = c && b + 3 * a;
printf ("d = %d", d);

Budući da je prioritet aritmetičkih veći od prioriteta logičkih operatora:
d = c && b + 3 * a   /* ovo je ekvivalentno izrazu u sljedećem retku */
d = c && (b + (3 * a))
d = 4 && (4+ (3 * 1))
d = 4 && 7
d = 1
```

Složeni izrazi često se pogrešno napišu zbog "doslovnog prepisivanja" izrečenog uvjeta:
Izraz "ako je x veći od 20 i manji od 100" (uočiti da se pod "manji od 100" podrazumijeva "x manji od 100"), često se "doslovno prepiše" u:

```
if(x>20 && <100)
```

što dovodi do pogreške pri prevođenju. Osim toga česta je pogreška i umjesto operatora "I" napisati operator "," te se prethodna naredba napiše:

```
if(x>20 , x<100)
```

što dovodi do "logičke" pogreške (koju je puno teže otkriti jer ju prevodilac ne prijavi).
Izraz "`x>20 , x<100`" odgovara kao da je napisano "`x<100`".

PRIORITET I PRIDRUŽIVANJE OPERATORA

Pojam pridruživanja odnosi se na redoslijed razrješavanja (s lijeva na desno ili obratno) dva operatora jednakog prioriteta navedena jedan za drugim. Tablica prioriteta i pridruživanja do sada spomenutih operatora:

	OPERATORI	PRIDRUŽIVANJE
↑Viši prioritet	`()`	L → D
	`! ~ ++ -- sizeof` unarni `+ -`	D → L
	`(cast)`	D → L
	`* / %`	L → D
	`+ -`	L → D
	`<< >>`	L → D
	`< <= > >=`	L → D
	`== !=`	L → D
	`&`	L → D
	`^`	L → D
	`|`	L → D
	`&&`	L → D
	`||`	L → D
	`?:`	D → L
Niži prioritet→	`= *= /= %= += -= &= ^= |= <<= >>=`	D → L

Uočiti da svi operatori uspoređivanja nisu istog prioriteta.

Primjer:

Napišite program za izračunavanje rješenja (korijena) kvadratne jednadžbe $ax^2 + bx + c = 0$.

Napomene uz program:

- Prva `#include <stdio.h>` naredba "uključuje" datoteku zaglavlja (header) s opisom funkcija i konstanti (dakle, ne implementacija tj. izvorni kôd tih funkcija) koje se odnose na rad s ulazom izlazom (čitanje s tipkovnice i ispis na zaslon)
- Druga `#include <math.h>` naredba "uključuje" datoteku zaglavlja s opisom matematičkih funkcija i konstanti (dakle, ne implementacija tj. izvorni kôd tih funkcija)
- Budući da se pri prevođenju (točnije: povezivanju–linking) s MS Visual C++ 6.0 po definiciji uključuje datoteka s implementacijom matematičkih funkcija, pri prevođenju nije potrebno navesti nikakve dodatne parametre. Pri radu na nekim inačicama Unix operacijskog sustava, C prevodilac ne uključuje po definiciji tu datoteku te je potrebno dodati i opciju "`-lm`" pri pokretanju prevođenja.

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

int main()
{
  float a, b, c, x1, x2, q, d, x1r, x2r, x1i, x2i;

  printf("Zadajte koeficijente kvadratne jednadzbe a,b,c:");
  scanf("%f %f %f", &a,&b,&c);

  d=b*b -4.0*a*c; /* diskriminanta */

  if (d > 0) {
        /* Rjesenja su realna  */
        q = pow (d, 1./2);
        x1 = (-b + q)/(2*a);
        x2 = (-b - q)/(2*a);
        printf ("X1=%f   X2=%f\n", x1, x2);
  } else if (d == 0) {
        /* postoji samo jedno rješenje */
        x1 = -b/(2*a);
        printf ("X1=X2=%f\n", x1);
  } else {
        /* Rjesenja su konjugirano kompleksni broj   */
        q = pow(-d, 1./2);
        x1r = -b/(2*a) ;
        x2r = x1r;
        x1i = q/(2*a);
        x2i = -x1i;
   printf ("X1 = (%f,%f)\n", x1r, x1i);
   printf ("X2 = (%f,%f)\n", x2r, x2i);
  }
   return 0;
}
```

4. Auditorne vježbe

Prioritet operatora

Koja je vrijednost varijable `d` nakon sljedećeg programskog odsječka:

```
short int a=4, b=2, c=8, d;
d = a < 10 && 2 * b < c;
```

Rješenje:

```
d = ( a < 10 ) && ( ( 2 * b ) < c )
d = 1 && (4<8)
d = 1 && 1
d = 1
```

Što će se ispisati slijedećim programskim odsječkom?

```
   int a = 5, b = -1, c = 0;
   c = (a = c && b) ? a = b : ++c;
   printf("a = %d, b = %d, c = %d: \n", a, b, c);
```

```
Rezultat:
   a = 0 , b = -1 , c = 1
```

Dvostrana selekcija

Primjer:

Napisati program koji računa i ispisuje apsolutnu vrijednost unesenog broja

```
#include<stdio.h>
int main(){
  int broj,abs;

  printf("Unesite broj: ");
  scanf("%d",&broj);      /*može se napisati i kao */
  if (broj < 0) {         /*if (broj < 0)          */
    abs = -broj;          /*    abs = -broj;        */
  } else {                /*else                    */
    abs = broj;           /*    abs = broj;         */
  }
  printf("Apsolutna vrijednost broja %d je %d\n", broj, abs);
  return 0;
}
```

Primjer:
Za zadani radijus izračunati površinu kruga.

```
#include<stdio.h>
#define PI 3.141592
int main(){
  float radijus,povrsina;

  printf("Unesite radijus kruga: ");
  scanf("%f",&radijus);
  if (radijus <= 0) {
    printf("Unijeli ste neispravan radijus!\n");
  } else {
    povrsina=radijus*radijus*PI;
  }
  printf("Povrsina kruga je: %f\n",povrsina);
  return 0;
}
```

Što će se ispisati ako se na upit `"Unesite radijus kruga:"` upiše vrijednost -10?
`Unijeli ste neispravan radijus!`
`Povrsina kruga je: x.xxxxx` (ovisno o sadržaju memorijskih lokacija dodijeljenih varijabli `povrsina`)

1. Kako modificirati gornji program da se poruka o izračunatoj površini ispiše samo u slučaju kada se zaista i izračuna?
2. Kako izbjeći neočekivan-nepoznat rezultat?

1. Naredbu `printf("Povrsina kruga je: %f\n",povrsina);` treba potpisati pod `else` dio:

```
  if (radijus <= 0) {
    printf("Unijeli ste neispravan radijus!\n");
  }
  else {
    povrsina=radijus*radijus*PI;
    printf("Povrsina kruga je: %f\n",povrsina);
  }
```

2. Inicijalizirati varijablu `povrsina`.

```
    float radijus,povrsina = 0;
```

Primjer:
Napisati program koji učitava dva broja, ispituje je li prvi broj djeljiv s drugim bez ostatka i ispisuje odgovarajuću poruku. Drugi broj ne smije biti 0 (zašto?).

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

  printf("Unesite a i b:");
  scanf("%d %d", &a, &b);
  if (b != 0) {
    if (a % b == 0) {
      printf("%d jest djeljiv s %d\n", a, b);
    } else {
      printf("%d nije djeljiv s %d\n", a, b);
    }
  }
  return 0;
}
```

Alternativno rješenje:

```
if (b != 0) {
  printf("%d %s djeljiv s %d\n", a, a % b ? "nije" : "jest", b);
}
```

Višestrana selekcija

Primjer: Kotao ima radnu temperaturu u intervalu [50°C-80°C]. Napisati program koji provjerava je li s tipkovnice zadana vrijednost temperature kotla u dozvoljenom intervalu i ispisuje odgovarajuću poruku.

```
#include <stdio.h>
#define donja 50.0
#define gornja 80.0
int main(){
  float temp;

  printf("\n Unesite temperaturu kotla: ");
  scanf("%f",&temp);
  if (temp < donja) {
    printf("Temperatura je pala ispod dozvoljene!\n");
  } else if (temp > gornja) {
    printf("Temperatura je porasla iznad dozvoljene!\n");
  } else {
    printf("Temperatura kotla OK\n");
  }
   return 0;
}
```

Programske petlje

Programske petlje služe za obavljanje određenog programskog odsječka više puta. Broj ponavljanja može, ali i ne mora biti unaprijed poznat. Dijelimo ih na petlje:

- s ispitivanjem uvjeta ponavljanja na početku (while, for)
- s ispitivanjem uvjeta ponavljanja na kraju (do-while)

Petlja s ispitivanjem uvjeta ponavljanja na početku

Primjer:

Napisati program koji će učitavati pozitivne cijele brojeve sve dok se ne unese broj 0, i zatim ispisati koji je najmanji uneseni broj. Ignorirati negativne brojeve zadane pomoću tipkovnice.

Napomena: kod traženja najmanjeg (najvećeg) člana polja ili niza koristimo sljedeći algoritam:

1. prvi član polja proglasimo najmanjim i njegovu vrijednost pohranimo u pomoćnu varijablu koja predstavlja trenutni minimum
2. pretražujemo preostale članove polja i, ukoliko je neki od njih manji od trenutnog minimuma, ažuriramo trenutni minimum na tu vrijednost
3. Nakon što smo pretražili cijelo polje, u pomoćnoj varijabli se nalazi minimum polja

Primjer:

niz: 5, 6, 3, 9, 4, 7, 2, -1, 5.

5
min=5

6	3	9	4	7	2	-1	5
6<5 ?	3<5?	9<3?	4<3?	7<3?	2<3?	-1<2	5<-1?
	min=3				**min=2**	**min=-1**	

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

  while(x <= 0){
    printf("Unesite broj : ");
    scanf("%d", &x);
    min = x;     // pretpostavljamo da je prvi unešeni najmanji
  }
  while (x != 0) {
    printf("Unesite broj : ");
    scanf("%d", &x);
    if(x > 0 && x < min ){
     min = x;
    }
  }
  printf("Najmanji uneseni broj je %d\n", min);
  return 0;
 }
```

Alternativno rješenje, da se izbjegne dvostruko učitavanje

```
#include <stdio.h>
int main() {
  int min=0, x=1;

  // u prvom prolasku x=1
  while (x != 0) {
    printf("Unesite broj : ");
    scanf("%d", &x); //pri prvom učitavanju x postaje != 1
    if (x > 0) {
      if ((min==0) || (x < min)) min = x;
    }
  }
  printf("Najmanji uneseni broj je %d\n", min);
  return 0;
 }
```

Primjer:

Učitavati pozitivne cijele brojeve sve dok njihova suma ne premaši dozvoljeni raspon za `short int`. Ispisati posljednju valjanu sumu.

```
#include <stdio.h>
void main() {
  short int x, suma = 0, gotovo = 0;

  while (!gotovo) {
    printf("Unesite broj : ");
    scanf("%d", &x);

    if ((signed short)(suma+x) >= suma) {   //zašto treba cast?
      suma += x;
    } else {
      gotovo = 1;
    }
  }
  printf("Suma: %d\n", suma);
  }
```

Primjer testiranja programa:

```
Unesite broj : 30000
Unesite broj : 10
Unesite broj : 10000
Suma: 30010
```

Da li je u ovom primjeru unaprijed poznat broj ponavljanja petlje?

Primjer:

Učitati pozitivan cijeli broj i izračunati slijed brojeva na sljedeći način:

- ako je broj paran podijeliti ga s 2
- ako je neparan pomnožiti s 3 i dodati 1

Ponavljati postupak dok broj ne postane jednak 1. U svakom koraku ispisati trenutnu vrijednost broja. Ispisati ukupan broj operacija nad učitanim brojem.

```
#include <stdio.h>
int main() {
    int broj, brKoraka = 0;

    printf("Upisi pozitivan broj:");
    scanf("%d", &broj);
    while (broj > 1) {
        ++ brKoraka;
        if (broj % 2){
            broj = broj * 3 + 1;
        }
        else {
            broj /= 2;
        }
        printf ("U %d. koraku broj = %d \n",brKoraka, broj);
    }
    printf ("Ukupno %d koraka. \n",brKoraka);
    return 0;
  }
```

```
Upisi pozitivan broj: 9
U 1. koraku broj = 28
U 2. koraku broj = 14
U 3. koraku broj = 7
U 4. koraku broj = 22
U 5. koraku broj = 11
U 6. koraku broj = 34
U 7. koraku broj = 17
U 8. koraku broj = 52
U 9. koraku broj = 26
U 10. koraku broj = 13
U 11. koraku broj = 40
U 12. koraku broj = 20
U 13. koraku broj = 10
U 14. koraku broj = 5
U 15. koraku broj = 16
U 16. koraku broj = 8
U 17. koraku broj = 4
U 18. koraku broj = 2
U 19. koraku broj = 1
Ukupno 19 koraka.
```

5. Auditorne vježbe

Petlja s poznatim brojem ponavljanja

Sintaksa:

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

- `izraz1` je izraz koji će se izvršiti samo jednom, prije ulaska u prvu iteraciju. Najčešće se koristi za inicijalizaciju brojača. Ako je potrebno napisati više naredbi u izrazu, one se odvajaju zarezom.
- `izraz2` se izračunava kao logički izraz (0 - false, !=0 - true), te se petlja izvršava dok je `izraz2` zadovoljen (istinit). Provjera uvjeta obavlja se prije svake iteracije.
- `izraz3` se obavlja nakon svake iteracije (prolaska kroz petlju). Najčešće se koristi za povećavanje brojača. Ako je potrebno napisati više naredbi u izrazu, one se odvajaju zarezom.
- bilo koji od izraza (`izraz1`, `izraz2`, `izraz3`) se može izostaviti. Ako je izostavljen `izraz2` petlja se izvodi kao da je logička vrijednost `izraza2` istinita (`true`).

Uobičajeno korištenje:

```
for (i = poc; i <= kraj; i = i + k)  {
     .
     .
     .
}
```

Primjer `for` petlje s dva brojača:

```
for (hi = 100, lo = 0; hi >= lo; hi--, lo++) {
     .
     .
     .
}
```

Primjer 1. Napisati program koji će izračunati aritmetičku sredinu za n učitanih brojeva.

Rješenje:

```
#include <stdio.h>

int main() {
   int    i, n, suma, x;
   float  arit_sred;

   printf("Za koliko brojeva želite izračunati aritmetičku
   sredinu : ");
   scanf("%d", &n );

   suma = 0;
   for(i=0; i<n; i++) {
       printf("Unesite %d. broj : ", i);
       scanf("%d", &x);
       suma += x;
   }
   arit_sred = (float) suma / n;
   printf("Aritmetička sredina učitanih brojeva je %f\n",
   arit_sred);
   return 0;
}
```

Primjer 2. Napisati program koji će ispisati realne brojeve od 0 do n s korakom od 0.1

Rješenje:

```
#include <stdio.h>

void main() {
   int n;
   float i;

   printf("Do kojeg broja zelite ispis :");
   scanf("%d", &n );

   for( i=0; i<=n; i=i+0.1 ) {
       printf("%f\n",i);
   }
}
```

Primjer 3. Napisati program koji će ispisati brojeve djeljive sa 7, 13 ili 19, a manje od učitanog broja n. Brojeve treba ispisati od najvećeg prema najmanjem.
Rješenje:

```
#include <stdio.h>

int main() {
   int i, n;

   printf("Upisite broj n : ");
   scanf("%d", &n );

   for( i=n; i > 0; i-- ) {
     if (( i % 7 == 0 ) || (i % 13 == 0) || (i % 19 == 0)) {
       printf(" %d \n", i);
     }
   }
   return 0;
}
```

Napomena: Beskonačna petlja

```
for(;;);  //beskonačna petlja
```

ili

```
for(;;){
     //blok naredbi
}
```

Blok naredbi unutar tijela petlje izvodi se beskonačno mnogo puta ako taj blok naredbi ne sadrži naredbu za izlazak iz petlje (`break`), naredbu za povratak iz funkcije (`return`), poziv funkcije za završetak programa (`exit`) ili `goto` naredbu.

Česte pogreške:

Kod	Ispis

`for( i=1; i = 10; i++){`
`  printf("%d\n",i);`
`}`
	neprekidno ispisuje broj 10 (beskonačna petlja)

`for( i=1; i == 10; i++){`
`  printf("%d\n",i);`
`}`
	neće ništa ispisati (naredba u tijelu petlje se neće ni jednom obaviti)

`for( i=1; i<=10; i++);`
`{`
`  printf("%d\n",i);`
`}`
	jedanput ispisuje broj 11

Petlja s ispitivanjem uvjeta ponavljanja na kraju

Sintaksa:

```
do {
     .
     .
     .
} while (izraz);
```

- petlja se izvršava sve dok je zadovoljen uvjet (dok je `izraz` istinit), za razliku od npr. Pascala gdje se petlja izvršava dok se uvjet ne zadovolji (znači dok je `false`)
- petlja će se izvršiti minimalno jedanput (prva provjera uvjeta ide tek nakon prvog prolaska kroz petlju)

Primjer u C-u:

```
do {
     .
     .
     .
} while (x < y);
```

Isti primjer u Pascal-u:

```
repeat
     .
     .
     .
until ( x >= y )
```

Primjer 4. Napisati program koji učitava brojeve iz intervala [-100,100] ili [800,1000]. Program treba ispisati koliko je bilo pozitivnih brojeva, koliko negativnih i koliko vrijednosti nula.

Program treba realizirati pomoću do – while petlje.

Rješenje:

```
#include <stdio.h>

void main(){
  int broj;
  int brojpozitivnih=0, brojnegativnih=0, brojnula=0;
  printf("Unesite brojeve: ");

 do {
   scanf("%d", &broj);
   if (broj>=-100 && broj<=100 || broj>=800 && broj<=1000) {
     if (!broj) brojnula++;
     else if (broj>0) brojpozitivnih++;
     else brojnegativnih++;
   }
 }while (broj>=-100 && broj<=100 || broj>=800 && broj<=1000);

 printf("Broj pozitivnih je %d, broj negativnih je %d, broj "
        "nula je %d\n", brojpozitivnih, brojnegativnih,
         brojnula);
}   // pojasniti nastavak stringa u novi red

/* printf("Broj pozitivnih je %d, broj negativnih je %d, broj \
nula je %d\n", brojpozitivnih, brojnegativnih,
        brojnula);*/
```

Primjer 5. Sljedeći programski odsječak

```
        k=0;
        i=7;
    lab:
        k=k+i*2;
        i=i-2;
        printf ("%d\n", k);
        if (i >= -7) goto lab;
```

nadomjestiti petljom s:

1.) ispitivanjem uvjeta ponavljanja na početku
2.) ispitivanjem uvjeta ponavljanja na kraju
3.) unaprijed poznatim brojem ponavljanja

Rješenje:

1.)
```
k=0;
i=7;
while (i>=-7) {
    k=k+i*2;
    i=i-2;
    printf ("%d\n", k);
}
```

2.)
```
k=0;
i=7;
do {
    k=k+i*2;
    i=i-2;
    printf ("%d\n", k);
} while(i >= -7);
```

3.)
```
for (i = 7, k = 0; i >= -7; i-=2){
    k = k + i * 2;
    printf ("%d\n", k);
}
```

Napomena: ako je inicijalno `i < -7` pod 1. i 3. petlja se neće obaviti niti jedanput a pod 2. hoće jedanput.

Prednost `for` petlje je u tome što se početna inicijalizacija brojača, ispitivanje uvjeta i korak brojača nalaze na jednom mjestu u kodu.

Naredbe za kontrolu toka petlje `break;` i `continue;`

U C-u postoje dvije naredbe za kontrolu toka petlje

- `break` – izlazi iz najbliže vanjske petlje
- `continue` – unutar `while` i `do` petlje prebacuje izvođenje programa na ispitivanje uvjeta, a unutar `for` petlje prebacuje izvođenje programa na korak `for` petlje i potom na ispitivanje uvjeta (također se odnosi na najbližu vanjsku petlju)

Primjer 6. Napisati program koji će provjeriti da li je zadani broj prost.
Rješenje:

```
#include <stdio.h>

  int main() {
   int i, n, prost=1; //prost je true

   printf("Upisite prirodan broj :");
   scanf("%d", &n);

   for( i=2; i<= n-1; i++) {
      if( n % i == 0 ) { //  moze i if(!(n % i))
       prost=0;  //prost je false
       break;
      }
   }
   if( prost )
      printf("%d jest prost broj !\n", n);
   else
      printf("%d nije prost broj !\n", n);

   return 0;
  }
```

Moguća poboljšanja algoritma (smanjivanje broja iteracija petlje):

- Dovoljno je s petljom ići do n/2, odnosno, samo do $\sqrt{n}$ (C funkcija `sqrt(n)`)
- Ispitati djeljivost broja s 2, te ako nije djeljiv s 2, unutar petlje ispitivati djeljivost s neparnim brojevima većim od 2

Primjer 7. Napisati program koji će učitavati cijele brojeve s tipkovnice i postupati prema sljedećem pravilu: ako je učitani broj manji od nule, treba ispisati poruku o pogrešci i prestati s učitavanjem brojeva. Ako je učitani broj veći od 100, treba ga zanemariti i prijeći na sljedeći broj, a inače treba ispisati taj broj. Osim u slučaju pogreške s učitavanjem brojeva prestati kada se učita 0.

Rješenje:

```
#include <stdio.h>

int main() {
 int x;


    do {
         printf ("Upisite broj :\n");
         scanf("%d", &x );
         if (x < 0) {
             printf("Nedozvoljena vrijednost\n");
             break;
             /* Izlazak iz petlje */
         }

         if (x > 100) {
             printf("Zanemarujem vrijednost\n");
             continue;
             /* Skok na novu iteraciju */
         }

         printf ("Upisani broj je : %d", x);

    } while (x != 0);
    return 0;
}
```

Naredba uvjetnog grananja `switch - case`

Sintaksa:

```
switch( izraz ){
        case  const_izraz1: naredbe1;
        [case const_izraz2: naredbe2;]
            .
            .
            .
        [default : naredbeN;]
     }
```

- može se upotrijebiti umjesto višestrane `if` selekcije
- obratiti pažnju: ako se unutar `case` bloka ne navede ključna riječ `break;` nastavak programa je sljedeći `case` blok u listi!
- izraz unutar switch-a mora biti cjelobrojan

Primjer 8. Učitati s tipkovnice brojevnu ocjenu (od 1 do 5), a ispisati njezinu opisnu vrijednost.

```
#include <stdio.h>
int main() {
 int ocjena;
    printf ("Upisite ocjenu :");
    scanf("%d", &ocjena);

    switch (ocjena) {
        case 1:
            printf("Nedovoljan\n");break;
        case 2:
            printf("Dovoljan\n");break;
        case 3:
            printf("Dobar\n");break;
        case 4:
            printf("Vrlo dobar\n");break;
        case 5:
            printf("Odlican\n");break;
        default:
            printf("Unijeli ste nepostojecu ocjenu\n");
            break;  // može i bez naredbe break
    }
    return 0;
}
```

Napomena: Ako bi izostavili `break;` unutar svih `case` blokova, za učitanu vrijednost npr. 3 ispis bi izgledao ovako:

```
Dobar
Vrlo dobar
Odlican
Unijeli ste nepostojecu ocjenu
```

Napomena: Ako je `default` blok posljednji blok `switch` naredbe nije unutar njega potrebno pisati naredbu `break`.

Napomena: Posljedica propadanja po `case` labelama u nedostatku `break` naredbe je da isti skup naredbi može biti izvođen za više različitih `case` labela pa se sljedeći kod

```
switch (znak) {
     case 'a': brsamoglasnika++; break;
     case 'e': brsamoglasnika++; break;
     case 'i': brsamoglasnika++; break;
     case 'o': brsamoglasnika++; break;
     case 'u': brsamoglasnika++; break;
     default: brostalih++; break;
}
```

kraće može napisati

```
switch (znak) {
     case 'a':
     case 'e':
     case 'i':
     case 'o':
     case 'u': brsamoglasnika++; break;
     default: brostalih++; break;
}
```

6. Auditorne vježbe

Petlje

Primjer 1. Što će se ispisati sljedećim programskim odsječkom?

```
i = 1;
while (i < 5) {
  if (i==3) {
    printf ("\n Hello world %d.x!", i);
    continue;
  } else if (i==4) {
    printf ("\n Goodbye %d.x!", i);
    continue;
  }
  i++;
}
```

Rješenje:

Hello world 3.x!
Hello world 3.x!
...
... i tako beskonačno puta. Kada `i` dostigne vrijednost 3 izvršava se blok naredbi pod "i==3". Zbog continue `i` se ne povećava nego se program grana na uvjetni izraz (i < 5).

Primjer 2. Napisati program koji će ispisati tablicu množenja do 100.

```
/* 1*/ int main() {
/* 2*/  int i,j;
/* 3*/
/* 4*/  for (i = 1; i <= 10; ++i) {
/* 5*/    for (j = 1; j <= 10; ++j){
/* 6*/      printf("%3d",i*j);
/* 7*/     }
/* 8*/    printf("\n");
/* 9*/  }
/*10*/  return 0;
/*11*/}
```

Dodavanjem jedne linije koda postići to da se u tablici nalaze samo parni brojevi.

```
/*6*/ if ( (i%2!=0) && (j%2!=0) ) continue;  //oba neparna
```

isto se postiže i s

```
/*6*/ if ( (i%2==0) || (j%2==0) )  //barem jedan paran
```

Primjer 3. Napisati program koji će ispisivati prvih N Fibonaccijevih brojeva. N učitavati pomoću tipkovnice. Algoritam za računanje Fibonaccijevih brojeva:

Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
Fibonacci(0) = Fibonacci(1) = 1

1, 1, 2, 3, 5, 8, 13, 21,...

Rješenje:

```
#include <stdio.h>
int main () {
  int N, i, f0 = 1, f1 = 1,f = 1;
  printf ("\n Upisite broj Fibonacci-jevih brojeva : \n");
  scanf ("%d",&N);
  for (i = 0; i <= N; i++) {
    if (i > 1) {
      f = f1 + f0;
      f0 = f1;
      f1 = f;
    }
    printf ("Fibonnaci (%d) = %d \n", i , f);
  }
  return 0;
}
```

Primjer 4. Napisati program koji učitava 2 realna broja s tipkovnice i znak računske operacije (jedne od osnovne četiri) koju je potrebno obaviti nad njima. Ukoliko je unešena neka druga operacija, tražiti ponovno unošenje operacije.

Rješenje:

```
#include <stdio.h>
#include <conio.h>[1]

int main() {

  float a,b, rezultat;
  char operacija;
  int ponoviUnosOperacije = 1;

  printf("Unesi dva broja: ");
  scanf("%f, %f", &a, &b); //komentirati zarez
```

[1] Biblioteka conio.h nije dio ANSI standarda

```
  do {
    printf("Unesi operaciju: ");
    operacija = getche()[2];
    printf("\n");
    ponoviUnosOperacije = 0;

    switch(operacija) {
      case '+': rezultat = a + b; break;
      case '-': rezultat = a - b; break;
      case '*': rezultat = a * b; break;
      case '/':
        if (b == 0) {
          printf("Dijeljenje s 0 nije dopusteno.\n");
        }
        else {
          rezultat = a / b;
        }
        break;
      default:
        ponoviUnosOperacije = 1;
      }
    } while (ponoviUnosOperacije);

  printf("%f %c %f = %f\n", a, operacija, b, rezultat);
  // Što ako je b == 0?
  return 0;
}
```

Polja

Polje je podatkovna struktura gdje isto ime dijeli više podataka. Pri tom svi podaci u polju moraju biti istog tipa. Elementima polja se pristupa koristeći indeks, koji mora biti nenegativni cijeli broj (konstanta, varijabla, cjelobrojni izraz). Indeks elemenata je broj između 0 i broja elemenata minus jedan, uključivo, tj. indeks $\in$ [0, BrojElemenata – 1].

Definicija polja u C-u:
`int x[20]` – polje od 20 cijelih brojeva
`char znakovi[2]` – polje od 2 znaka
`float niz[MAX]` – MAX je konstanta

[2] Funkcija getche() nije dio ANSI standarda

Definicija s pridjeljivanjem vrijednosti elemenata:

```
int polje[] = {1, 2, 3};
polje[0] = 1
polje[1] = 2
polje[2] = 3

int polje[4] = {1, 2};
polje[0] = 1
polje[1] = 2
polje[2] = 0
polje[3] = 0
```

Pristupanje elementima polja:

`x[0]` – prvi element polja
`niz[i]` – (i+1). element polja, 0 <= i <= BrojElemenata – 1
`niz[MAX - 1]` – posljednji element polja

Neispravno pristupanje elementima polja:

```
int polje[10] = {0};
int x = polje[10];

float a = 1.;
int x = polje[a];

int a = 1, b = 0, c = 1;
int x = polje[(a && b) - c];
```

Primjer 5. Napisati program koji će tražiti unos niza brojeva, nakon čega će izračunati aritmetičku sredinu tog niza, te najprije ispisati brojeve manje od aritmetičke sredine, a zatim one koji su veći od aritmetičke sredine.

Rješenje:

```
#include <stdio.h>
#define DIMENZIJA 10

int main() {
  int i;
  float suma = 0., arit_sred = 0., niz[DIMENZIJA]={0};

  for (i = 0; i < DIMENZIJA; i++) {
    printf("Unesi broj: ");
    scanf("%f",&niz[i]);
    suma += niz[i];
  }

  arit_sred = suma / DIMENZIJA;
  printf("Aritmeticka sredina niza je %6.2f.\n", arit_sred);

  for (i = 0; i < DIMENZIJA; i++) {
    if (niz[i] < arit_sred) {
      printf("%6.2f je manji od arit. sredine.\n", niz[i]);
    }
  }

  for (i = 0; i < DIMENZIJA; i++) {
    if (niz[i] > arit_sred) {
      printf("%6.2f je veci od arit. sredine.\n", niz[i]);
    }
  }

  // Što ako je broj točno jednak arit_sred?
  return 0;
}
```

Primjer 6. Napisati program u koji će se unijeti niz brojeva. Nakon unosa treba sve članove niza podijeliti s najvećim članom niza, i iskazati ih relativno u odnosu na najveći član.

Rješenje:

```
#include <stdio.h>
#define DIMENZIJA 10

int main() {

  int i;
  float max, polje[DIMENZIJA];

  for (i = 0; i < DIMENZIJA; i++) {
    printf("polje[%d] = ", i);
    scanf("%f",&polje[i]);

    if (i == 0) {
      max = polje[i];
    }
    if (max < polje[i]) {
      max = polje[i];
    }
  }

  printf("Najveci element polja je %f.\n\n", max);

  for (i = 0; i < DIMENZIJA; i++) {
    polje[i] /= max;
    printf("polje[%d] = %f\n", i, polje[i]);
  }
  return 0;
}
```

Primjer 7.

Napisati program koji će učitavati prirodne brojeve u intervalu [10, 99] i brojati koliko puta je učitan koji broj. Učitavanje prekinuti kad se unese broj izvan zadanog intervala. Nakon učitavanja program treba ispisati koliko je puta učitan svaki broj iz zadanog intervala koji je učitan barem jednom.

Rješenje:

```
#include <stdio.h>

#define DG 10          /* donja granica intervala */
#define GG 99          /* gornja granica intervala */

int main() {
  int broj, i;
  int brojac[GG - DG + 1] = { 0 };

  do {
    printf("\nUnesite broj u intervalu [%d, %d]: ", DG, GG);
    scanf("%d", &broj);
    if (broj >= DG && broj <= GG) {
      brojac[broj - DG]++;
    }
  } while (broj >= DG && broj <= GG);

  for (i = DG; i <= GG; i++) {
    if (brojac[i - DG] > 0) {
      printf("\nBroj %d se pojavio %d puta"
           ,i ,brojac[i - DG]);
    }
  }
  return 0;
}
```

7. Auditorne vježbe

Dvodimenzionalna polja

Deklaracija 2D polja u C-u:
`int x[3][2]` – matrica 3X2 (3 retka i 2 stupca) čiji su elementi cijeli brojevi
`char znakovi[2][4]` – polje znakova 2 retka i 4 stupca
`float niz[MAXRED][MAXSTUP]` – MAXRED i MAXSTUP su konstante; matrica MAXRED X MAXSTUP

Primjer deklaracije s inicijalizacijom:

```
int polje[3][3] = {1, 2, 3, 4, 5};
polje[0][0] = 1
polje[0][1] = 2
polje[0][2] = 3
polje[1][0] = 4
polje[1][1] = 5
polje[1][2] = 0
polje[2][0] = 0
polje[2][1] = 0
polje[2][2] = 0

char fer[3] = {'F', 'E', 'R'} // polje znakova

int y[3][4] = {{1, 2, 3},
               {4, 5, 6},
               {7, 8, 9} };

y[0][0]= 1   y[0][1]= 2   y[0][2]= 3   y[0][3]= 0
y[1][0]= 4   y[1][1]= 5   y[1][2]= 6   y[1][3]= 0
y[2][0]= 7   y[2][1]= 8   y[2][2]= 9   y[2][3]= 0
```

Deklaracija višedimenzionalnog polja

```
int x[3][2][4]      3D polje cijelih brojeva
float x[3][2][4][1] 4D polje realnih brojeva
```

Primjer 1.
Napisati program koji će učitati realnu matricu dimenzija 10x10 te naći najmanji element na glavnoj i najmanji na sporednoj dijagonali.

```
#include <stdio.h>

#define BR_RED  10
#define BR_STUP 10

int main() {
  int   i, j;
  float mat[BR_RED][BR_STUP], min_gl, min_sp;

  printf("Unos elemenata matrice :");
  for (i = 0; i < BR_RED; i++) {
    for (j = 0; j < BR_STUP; j++) {
      printf("\nUnesite element [%d][%d] : ", i, j);
      scanf("%f", &mat[i][j]);
    }
  }

  min_gl = mat[0][0]; //minimun je clan mat(0,0) pa krece od 1
  for (i = 1; i < BR_RED; i++) {
    if (mat[i][i] < min_gl) {
      min_gl = mat[i][i];
    }
  }

  min_sp = mat[0][BR_STUP-1];
  for (i = 1; i < BR_RED; i++) {
    if (mat[i][BR_STUP-i-1] < min_sp) {
      min_sp = mat[i][BR_STUP-i-1];
    }
  }

  printf("\nNajmanji element na glavnoj dijagonali je : %f",
         min_gl);
  printf("\nNajmanji element na sporednoj dijagonali je : %f",
         min_sp);
  return 0;
}
```

Skraćena varijanta sa samo jednim prolaskom kroz matricu:

```
#include <stdio.h>

#define BR_RED  10
#define BR_STUP 10

int main() {
  int   i, j;
  float mat[BR_RED][BR_STUP], min_gl, min_sp;

  printf("\nUnos elemenata matrice :\n");
  for (i = 0; i < BR_RED; i++) {
    for (j = 0; j < BR_STUP; j++) {
      printf("\nUnesite element [%d][%d] : ", i, j);
      scanf("%f", &mat[i][j]);
      if (i == 0 && j == 0) {
        min_gl = mat[i][j];
      }
      if (i == 0 && j == BR_STUP - 1) {
        min_sp = mat[i][j];
      }
      if (i == j) {
        if (mat[i][j] < min_gl) {
          min_gl = mat[i][j];
        }
      }
      if (i == BR_STUP - 1 - j) {
        if (mat[i][j] < min_sp) {
          min_sp = mat[i][j];
        }
      }
    }
  }

  printf("\nNajmanji element na glavnoj dijagonali je : %f",
         min_gl);
  printf("\nNajmanji element na sporednoj dijagonali je : %f",
         min_sp);
  return 0;
}
```

Primjer 2.
Napisati program za transponiranje matrice. Potrebno je učitati dimenzije matrice i sve elemente matrice. Transponirana matrica je ona kojoj se zamijene reci i stupci.

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

#define MAX_RED  50
#define MAX_STUP 50

int main() {
  int i, j, m, n, pom, max;
  int mat[MAX_RED][MAX_STUP];

// varijabla dim postavlja se na manju vrijednost od mogucih
// maksimalnih broja redaka i stupaca

  int dim = (MAX_RED < MAX_STUP)? MAX_RED : MAX_STUP;

  // ucitavanje velicine matrice
  do {
    printf("Upisite vrijednost za broj redaka < %d:", dim);
    scanf("%d", &m);
    printf("Upisite vrijednost za broj stupaca < %d:", dim);
    scanf("%d", &n);
  }  while (m < 1 || m > dim || n < 1 || n > dim);

  // ucitavanje elemenata matrice
  printf("\nUnos elemenata matrice :\n");
  for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        printf("Unesite element [%d][%d] : ", i, j);
        scanf("%d", &mat[i][j]);
    }
  }

  // ispis prije transponiranja
  printf("\n\nIspis matrice prije transponiranja:\n");
  for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
      printf("%3d", mat[i][j]);
    }
    printf("\n");
  }
```

```
  max=m>n ? m : n;
  // transponiranje
  for ( i=0; i<max; ++i ) {
     for ( j=i+1; j<max; ++j ) {// petlja mora ici od i+1 !!
         pom = mat[i][j];
         mat[i][j] = mat[j][i];
         mat[j][i] = pom;
     }
  }

  // ispis nakon transponiranja
  // broj redaka je sada broj stupaca ...
  printf("\nIspis matrice nakon transponiranja:\n");
  for (i = 0; i < n; i++) {
     for (j = 0; j < m; j++) {
       printf("%3d", mat[i][j]);
     }
    printf("\n");
  }
  return 0;
} // main
```

Primjer izvođenja:

```
Upisite vrijednost za broj redaka < 50: 3
Upisite vrijednost za broj stupaca < 50: 2
Unos elemenata matrice :
Unesite element [0][0] : 1
Unesite element [0][1] : 2
Unesite element [1][0] : 3
Unesite element [1][1] : 4
Unesite element [2][0] : 5
Unesite element [2][1] : 6

Ispis matrice prije transponiranja :
1   2
3   4
5   6

Ispis matrice nakon transponiranja :
1  3  5
2  4  6
```

Primjer 3.

Napisati program koji će učitati realnu matricu dimenzija 10x10 te naći sume elemenata svakog stupca i produkte elemenata svakog retka. Ispisati najmanju sumu i pripadni indeks stupca te najveći produkt i pripadni indeks retka. Sume i produkte čuvati u jednodimenzionalnim poljima.

```
#include <stdio.h>

#define BR_RED  10
#define BR_STUP 10

int main() {
  int    i, j;
  int    min_sum_ind, max_prod_ind;
  float  mat[BR_RED][BR_STUP];
  float  sum[BR_STUP], prod[BR_RED];

  /* 1. varijanta unosa elemenata i racunanja
     suma i produkata */
  for (i = 0; i < BR_RED; i++) {
    for (j = 0; j < BR_STUP; j++) {
      printf("\nUnesite element [%d][%d] : ", i, j);
      scanf("%f", &mat[i][j]);
    }
  }

  for (j = 0; j < BR_STUP; j++) {
    sum[j] = 0;
    for (i = 0; i < BR_RED; i++) {
      sum[j] += mat[i][j];
    }
  }

  for (i = 0; i < BR_RED; i++) {
    prod[i] = 1;
    for (j = 0; j < BR_STUP; j++) {
      prod[i] *= mat[i][j];
    }
  }
  /* kraj unosa elemenata i racunanja suma i produkata */
```

```
  /* naci indeks stupca za najmanju sumu */
  min_sum_ind = 0;
  for (j = 1; j < BR_STUP; j++) {
    if (sum[j] < sum[min_sum_ind]) {
      min_sum_ind = j;
    }
  }

  /* naci indeks retka za najveci produkt */
  max_prod_ind = 0;
  for (i = 1; i < BR_RED; i++) {
    if (prod[i] > prod[max_prod_ind]) {
      max_prod_ind = i;
    }
  }

  printf("\nNajmanja suma je %f , a pripadni indeks je %d\n",
         sum[min_sum_ind], min_sum_ind);
  printf("\nNajveci produkt je %f , a pripadni indeks je
         %d\n", prod[max_prod_ind], max_prod_ind);
  return 0;
}
```

Odsječak skraćene varijanta učitavanja elemenata matrice i računanja suma i produkata:

```
  /* 2. varijanta unosa elemenata i racunanja
     suma i produkata */
  for (i = 0; i < BR_RED; i++) {
    prod[i] = 1;
    for (j = 0; j < BR_STUP; j++) {
      printf("\nUnesite element [%d][%d] : ", i, j);
      scanf("%f", &mat[i][j]);
      prod[i] *= mat[i][j];
      if (i == 0) {
        sum[j] = 0;
      }
      sum[j] += mat[i][j];
    }
  }
  /* kraj unosa elemenata i racunanja suma i produkata */
```

Primjer 4.
Napisati funkciju koja izračunava aritmetičku sredinu za tri zadana realna broja i glavni program koji učitava tri broja i korištenjem napisane funkcije izračunava njihovu aritmetičku sredinu.

```
#include <stdio.h>

float arit_sred( float a, float b, float c )
{
  float ar;
  ar = (a + b + c) / 3;
  return ar; // Koliko se vrijednosti može vratiti sa return
}
int main() {
  float x, y, z, sred;
  printf("\nUčitaj tri realna broja : ");
  scanf("%f %f %f", &x, &y, &z );
  sred = arit_sred(x,y,z);
  printf("\nAritmetička sredina unesenih brojeva je : %f",
         sred);
  return 0;
}
```

Primjer 5. Što će se ispisati nakon obavljanja programa koji poziva funkciju ?

```
void uduplaj1(int x)
{
  printf ("\nF:Ulazni argument prije izmjene je %d ", x);
  x *= 2;
  printf ("\nF:Ulazni argument nakon izmjene je %d ", x);
}
int main() {
  int broj=10;
  printf ("\nG:Broj prije poziva funkcije je %d ", broj);
  uduplaj1 (broj);
  printf("\nG:Broj nakon poziva funkcije je %d ", broj);
  return 0;
}
```

Rezultat ispisa je:

```
G:Broj prije poziva funkcije je 10
F:Ulazni argument prije izmjene je 10
F:Ulazni argument nakon izmjene je 20
G:Broj nakon poziva funkcije je 10
```

Promjena unutar funkcije nije zapamćena nakon povratka u glavni program ! Zašto ?

```
int uduplaj2(int x)
{
  printf ("\nF:Ulazni argument prije izmjene je %d ", x);
  x *= 2;
  printf ("\nF:Ulazni argument nakon izmjene je %d ", x);
  return x;
}

int main() {
  int broj=10;
  printf ("\nG:Broj prije poziva funkcije je %d ", broj);
  broj = uduplaj2 (broj);
  printf("\nG:Broj nakon poziva funkcije je %d ", broj);
  return 0;
}
```

```
G:Broj prije poziva funkcije je 10
F:Ulazni argument prije izmjene je 10
F:Ulazni argument nakon izmjene je 20
G:Broj nakon poziva funkcije je 20
```

Primjer 6.

Napisati funkciju koja će za zadani argument u radijanima izračunati vrijednost funkcije sinus kao sumu *n* članova reda. Funkcija sinus je definirana redom:

$$\sin(x)=\sum_{i=1}^{\infty}(-1)^{i+1}\frac{x^{2i-1}}{(2i-1)!}=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+...$$

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

// realizacija bez korištenja pomoćnih funkcija
float sinus(float x, int n){
     int    i, predznak;
     float  sum, clan, fakt, xpot;
     sum = 0.0;
     xpot = x;
     fakt = 1.0;
     predznak = 1;
     for( i=1; i<=n; i++ ) {
       clan = predznak * xpot / fakt;
       predznak *= -1;
       xpot *= x*x;
       fakt *= (2*i) * (2*i+1);
       sum += clan;
     }
     return sum;
}
// realizacija sa korištenjem pomoćnih funkcija
long fakt( int n )
{
     int   i;
     long  f=1;
     for( i=1; i<=n; i++ )
       f *= i;
     return f;
}

float sinus(float x, int n)
{
     int i, predznak;
     float  sum, clan;
     sum = 0.0;
     predznak = 1;
     for( i=1; i<=n; i++ ) {
       clan = predznak * pow(x,2*i-1) / fakt(2*i-1);
       predznak *= -1;
       sum += clan;
     }
     return sum;
 }
```

8. Auditorne vježbe

Pokazivači

Napomena:
Svi primjeri su prevedeni korištenjem *Microsoft Visual C++* prevodioca.

Primjer 1.
Što će se ispisati izvođenjem sljedećeg programskog odsječka?

```
int a = 5;
```

a	1245052
5	

```
int *b;
```

a	1245052	b	1245048
5		?	

```
b = &a;
```

a	1245052	b	1245048
5		1245052	

```
*b = 6;
```

a	1245052	b	1245048
6		1245052	

```
printf("a = %d *b = %d\n", a, *b);
printf("a = %d b = %ld\n", a, b);
printf("&a = %ld &b = %ld\n", &a, &b);
```

Rješenje:
```
a = 6 *b = 6
a = 6 b = 1245052
&a = 1245052 &b = 1245048
```

Važno: vrijednost na koju *pokazuje* varijabla `b` nalazi se na istoj adresi (`b = &a`) na kojoj se nalazi i vrijednost koju *sadrži* varijabla `a`.

Što bi se dogodilo da izostavimo liniju `b = &a;` ?

Rješenje:
Pokazivač `b` prije pridruživanja vrijednosti pokazuje na nedefiniranu adresu, pa može doći do pokušaja upisivanja vrijednosti `6` na adresu koja je rezervirana za pohranu drugih varijabli ili koda. To može izazvati neočekivano ponašanje ili izazvati pogrešku pri izvođenju programa zbog neovlaštenog pristupa dijelu memorije.

Tipične pogreške:

```
scanf("%d", n);
printf("%d", &n);
```

Primjer 2. Povezanost pokazivača i polja

Ime svakog deklariranog polja se može koristiti kao pokazivač, a svaki pokazivač se može koristiti kao polje.

Što će se ispisati izvođenjem sljedećeg programskog odsječka?

```
int x[] = {0, 1, 2};
```

x[0]	1245044	x[1]	1245048	x[2]	1245052
0		1		2	

```
printf("&x[0] = %ld x[0] = %d\n", &x[0], x[0]);     // 1
printf("&x[1] = %ld x[1] = %d\n", &x[1], x[1]);     // 2
printf("&x[2] = %ld x[2] = %d\n", &x[2], x[2]);     // 3
```

Rješenje:

```
&x[0] = 1245044 x[0] = 0
&x[1] = 1245048 x[1] = 1
&x[2] = 1245052 x[2] = 2
```

Napomena:

Ispisi u gornjem odsječku mogli su biti zamijenjeni sljedećim naredbama:

```
printf("x = %ld *x = %d\n", x, *x);                       // 1
printf("(x + 1) = %ld *(x+1) = %d\n", x + 1, *(x+1));     // 2
printf("(x + 2) = %ld *(x+2) = %d\n", x + 2, *(x+2));     // 3
```

zato što su sljedeći izrazi ekvivalentni:

*x ⇔ x[0] x ⇔ &x[0]

*(x+1) ⇔ x[1] x+1 ⇔ &x[1]

*(x+2) ⇔ x[2] x+2 ⇔ &x[2]

Ime polja (u gornjem primjeru: `x`) predstavlja pokazivač na nulti član polja.
Notacija `x + 1` predstavlja pokazivač na prvi član polja, koji je od nultog udaljen za `1 * sizeof(int)`. Analogno, `x + 2` predstavlja pokazivač na drugi član polja, koji je od nultog člana polja udaljen za `2 * sizeof(int)`.

Primjer 3. Ime polja kao pokazivačka konstanta
Pri prevođenju programskog odsječka:

```
int a[10], b[10];
a = b;
```

dojavit će se pogreška. Ime polja (`a`) predstavlja pokazivačku konstantu, pa se vrijednost tog pokazivača ne može mijenjati.

Primjer 4. Povezivanje pokazivača i polja
Pridjeljivanje adrese polja pokazivaču (što je ekvivalentno pridjeljivanju adrese prvog elementa polja pokazivaču) može se izvesti na dva načina:

```
int *p, a[10];
p = a;       // 1. način
p = &a[0];   // 2. način
```

Primjer 5. Aritmetika pokazivača
Što će se ispisati izvođenjem sljedećeg programskog odsječka?

```
int x[] = {1, 2, 3, 4};
```

x[0]	1245040	x[1]	1245044	x[2]	1245048	x[3]	1245052
1		2		3		4	

```
int *p = &x[2];
```

p	1245036
1245048	

```
int *q = &x[1];
```

q	1245032
1245044	

```
int *r = ++q;
```

r	1245028	q	1245032
1245048		1245048	

```
printf("(p + 1) = %d *(p + 1) = %d\n", (p + 1), *(p + 1));
printf("(p - 1) = %d *(p - 1) = %d\n", (p - 1), *(p - 1));
printf("q = %d *q = %d\n", q, *q);
printf("r = %d *r = %d\n", r, *r);
```

Rješenje:

```
(p + 1) = 1245052 *(p + 1) = 4
(p - 1) = 1245044 *(p - 1) = 2
q = 1245048 *q = 3
r = 1245048 *r = 3
```

Primjer 6. Prijenos adresa argumenata (*call by reference*) u funkciju

Napisati funkciju koja pretvara polarne koordinate u kartezijeve, te prikazati poziv funkcije iz glavnog programa.

```
#include <math.h>
void kart(float r, float fi, float *x, float *y) {

     *x = r * cos(fi);

     *y = r * sin(fi);

}
```

x	1244956	*x	1245052
1245052		1.755165	

y	1244960	*y	1245048
1245048		0.958851	

Odsječak glavnog programa:

```
float x, y;
```

x	1245052	y	1245048
?		?	

```
float fi = 0.5, r = 2;
```

fi	1245044	r	1245040
0.5		2	

```
kart(r, fi, &x, &y);
```

```
printf("x=%f y=%f", x, y);
```

x	1245052	x	1245048
1.755165		0.958851	

Pitanje: Što bi se dogodilo kada bi funkcija izgledala ovako:

```
void kart(float r, float fi, float *x, float *y) {
     x = 10000;
     *x = r * sin(fi);
     *y = r * cos(fi);
}
```

Važno: Funkcija može vratiti samo jednu vrijednost pomoću naredbe `return`. Ukoliko funkcija treba vratiti više vrijednosti (u ovom slučaju `x` i `y`), tada se te vrijednosti vraćaju preko argumenata (*Call by Reference*).

Primjer 7.
Napisati funkciju koja za zadane brojnike i nazivnike dvaju razlomaka vraća brojnik i nazivnik njihovog zbroja. Ako je moguće, potrebno je skratiti brojnik i nazivnik zbroja. Brojnici i nazivnici pribrojnika su prirodni brojevi.

```
void zbrojiRazlomke(int br1, int naz1, int br2, int naz2, int
     *brZbroj, int *nazZbroj) {

     int min, i;

     *brZbroj = br1 * naz2 + naz1 * br2;
     *nazZbroj = naz1 * naz2;

     min = (*brZbroj < *nazZbroj) ? *brZbroj : *nazZbroj;
     // ispituje se da li su brojnik i nazivnik
     // jedan drugome višekratnici
     if ((*brZbroj % min == 0) && (*nazZbroj % min == 0)) {
          *brZbroj /= min;
          *nazZbroj /= min;
     } else {
          // ispituje se da li su brojnik i nazivnik djeljivi
          // istim brojem
          i = min / 2;
          while ( i >= 2) {
               if ((*brZbroj % i == 0) && (*nazZbroj % i == 0)){
                    *brZbroj /= i;
                    *nazZbroj /= i;
               }
               else --i;
          }
     }
}
```

Odsječak glavnog programa (poziv funkcije `zbrojiRazlomke`):

```
int br1, naz1, br2, naz2, brZbroj, nazZbroj;
...
zbrojiRazlomke(br1, naz1, br2, naz2, &brZbroj, &nazZbroj);
```

Primjer 8. Funkcija koja pronalazi najveći element u polju realnih brojeva.
Zašto se kao argument funkcije prenosi duljina niza `duljina`?

```
float max(int duljina, float *p) {
// ILI: float max(int duljina, float p[]) {
    float rez; int i;

    rez = p[0];
    for (i = 1; i < duljina; i++)
        if (p[i] > rez)
            rez = p[i];
    return rez;
}
```

Isti problem riješen korištenjem aritmetike pokazivača:

```
float max(int duljina, float *p) {
    float rez; int i;

    rez = *p;
    p++;
    for (i = 1; i < duljina; i++, p++)
        if (*p > rez)
            rez = *p;
    return rez;
}
```

Napomena:
Pokazivač `p` sadrži adresu prvog člana polja, ali pomicanjem pokazivača `p` u funkciji adresa početnog člana polja neće se promijeniti, niti će imati utjecaja na pozivajući program.

Primjer 9.
Rezultati ispita prenose se u funkciju kao polje cijelih brojeva (`niz`) koji predstavljaju ocjene. Broj članova polja je `brOcjena`. Ocjene su predstavljene brojevima 1, 2, 3, 4 i 5. Napisati funkciju koja će vratiti ocjenu koju je dobilo najviše studenata.

```
int najcescaOcjena(int *niz, int brOcjena) {
    int ocjena[5] = {0}, najcescaOcjena = 0, i;

    for(i = 0; i < brOcjena; i++)
        ocjena[niz[i] -1]++;
```

```
    for(i = 1; i < 5; i++)
       if(ocjena[i] > ocjena[najcescaOcjena])
          najcescaOcjena = i;

    return najcescaOcjena + 1;
}
```

Primjer 10.
Napisati funkciju koja će sve elemente zadanog cjelobrojnog polja `p` od `N` elemenata posmaknuti za određeni broj mjesta `posmak` (`posmak < N`). Ukoliko je posmak pozitivan, elementi polja posmiču se udesno, a ukoliko je posmak negativan, elementi polja posmiču se ulijevo. Ispražnjeni elementi pune se nulama.

Primjer posmaka za `posmak` = 2 i polje [`10 40 50 60 12`]:

Polje prije posmaka: `[10 40 50 60 12]`
Polje nakon posmaka: `[ 0  0 10 40 50]`

Rješenje:

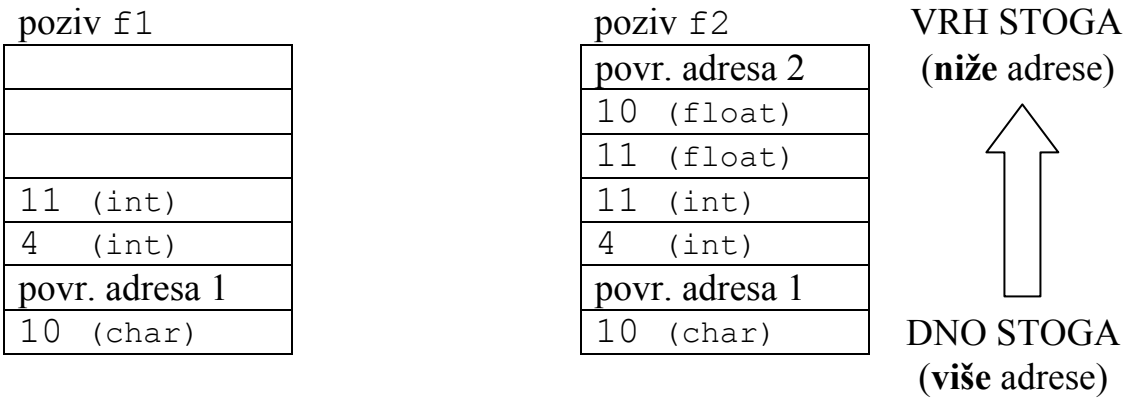
```
void posmakniPolje(int *p, int N, int posmak) {
     int i;
     // ako je posmak udesno
     if (posmak > 0 && posmak < N) {
          for (i = N - 1 - posmak; i >= 0; i --)
               p[i + posmak] = p[i];
          for (i = posmak - 1; i >= 0; i --)
               p[i] = 0;
     }



     // ako je posmak ulijevo
     else if (posmak < 0 && -posmak < N) {
          posmak = -posmak;
          for (i = 0; i <= N - 1 - posmak; i++)
               p[i] = p[i + posmak];
          for (i = N - posmak; i < N; i++)
               p[i] = 0;
     }
}
```

Primjer 11. Pohrana podataka na stog kod poziva funkcije
Svaki novi podatak koji se pohranjuje na stog, pohranjuje se na vrh stoga. Međutim, kod mnogih je prevodioca vrh stoga uvijek na adresi **nižoj** od dna stoga. To znači da se podaci na stog pohranjuju odozgo prema dolje (tj. od dna stoga prema **nižim** memorijskim adresama).

Na stog se pohranjuju sljedeći podaci (od viših adresa prema nižim):
- argumenti funkcije (najveću adresu ima krajnje desni argument)
- povratna adresa
- lokalne varijable (najveću adresu ima prva deklarirana varijabla)

Napomena:
Kod prijenosa podataka na stog neće se uzimati u obzir okvir stoga (registri procesora), koji se kod nekih prevodioca također pohranjuju na stog.

Za koliko byte-ova maksimalno naraste stog tijekom izvođenja odsječka:
```
...
y =  f1(10);
...
```
ako su definirane funkcije
```
long f2(float a, float b) {
    return a + b;
}
int f1(char x) {
    int i = 4, y;
    y = x + 1;
    return i * f2(x, y);
}
```



Povratna adresa je podatak tipa `long`. Pozivom funkcije `f2` stog je narastao na svoju najveću veličinu:
`sizeof(char) + sizeof`(povr. adresa 1)` + 2 * sizeof(int) +`
`2 * sizeof(float) + sizeof`(povr. adresa 2) = 25 bajta

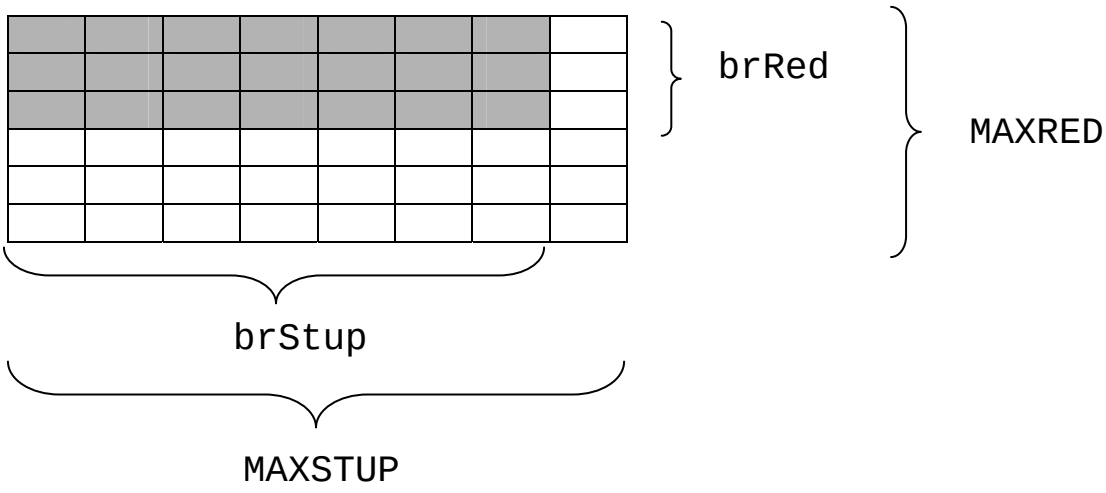
9. Auditorne vježbe

1. Definiran je maksimalni broj redaka i stupaca koji matrica može imati. Napisati glavni program u kojem se zadaje stvarni broj redaka i stupaca matrice, te unose elementi matrice. U glavnom programu ispisati:
   a. sumu elemenata matrice (pozivati funkciju za računanje sume elemenata matrice)
   b. maksimalne vrijednosti redaka matrice (pozivati funkciju za računanje najvećeg elementa u nizu)

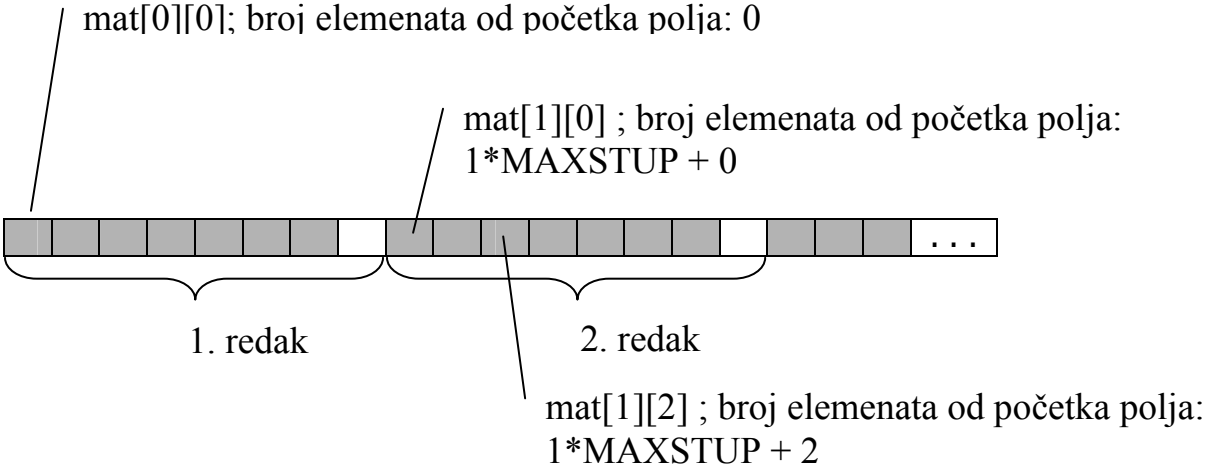
Deklarirana je matrica `mat` kao dvodimenzionalno polje na slijedeći način :

```
float mat[MAXRED][MAXSTUP];
```

U varijable `brRed` i `brStup` spremaju se dimenzije matrice koje zadaje korisnik.



U memoriji računala alocirano je `MAXRED * MAXSTUP * sizeof(float)` bajta. Retci matrice slijede jedan iza drugoga tako da se drugi redak smješta odmah nakon prvog retka itd.



Općenito za **`mat[i][j]`** – broj elemenata od početka polja je : **`i * MAXSTUP + j`**

```
#include <stdio.h>
#define MAXRED    100
#define MAXSTUP   100

float max( int len, float *p ) {
// ILI: float max( int len, float p[] )
  float  rez;
  int i;
  printf("\nU funkciji max:");
  printf("\nAdresa pocetka retka matrice u mem. : %ld", p);
  rez = p[0];
  for( i=1; i<len; i++ )
    if( p[i] > rez )
      rez = p[i];
  return rez;
}

float suma( int brRed, int brStup, int maxStup, float *mat ) {
  int i, j;
  float sum;

  printf("\nU funkciji suma:");
  printf("\nAdresa pocetka matrice u memoriji : %ld", mat);
  printf("\nAdr. poc. drugog retka:%ld", &mat[maxStup]);
  printf("\nAdresa poc. treceg retka:%ld", &mat[2* maxStup]);

  sum = 0.0;
  for( i=0; i<brRed; i++ )
    for( j=0; j<brStup; j++ )
      sum += mat[i*maxStup + j];
                  // ili sum += *(mat + i*maxStup + j)
  return sum;
}

void main() {
  int red, stup, i, j;
  float zbroj, maksRed, mat[MAXRED][MAXSTUP];

  printf("\nUnesi broj redaka i stupaca :");
  scanf("%d %d", &red, &stup );

  printf("\nU funkciji main");
  printf("\nAdresa pocetka matrice u memoriji : %ld", mat);
  printf("\nAdresa pocetka drugog retka:%ld\n\n", &mat[1][0]);
```

```
  printf("\nAdresa pocetka treceg retka:%ld\n\n", &mat[2][0]);
  for( i=0; i<red; i++ )
    for( j=0; j<stup; j++ ) {
      printf("Unesi element [%d,%d] ", i, j );
      scanf("%f", &mat[i][j] );
  }

  // izracunaju sumu elemenata matrice
  zbroj = suma(red, stup, MAXSTUP,(float *) mat);
  printf("\n\nSuma elemenata matrice je %f", zbroj );

  // ispisi maksimalne vrijednosti redaka matrice
  for(i=0;i<red;i++) {
    maksRed = max(stup, &mat[i][0]);
                  // ili … max(stup, mat+i*MAXSTUP)
    printf("\nNajveci element u retku %d je %f \n",i,maksRed);
  }
}
```

Primjer izvođenja programa:

```
Unesi broj redaka i stupaca :3 2

U funkciji main
Adresa pocetka matrice u memoriji : 1205032
Adresa pocetka drugog retka: 1205432
Adresa pocetka treceg retka: 1205832

Unesi element [0,0] 2
Unesi element [0,1] 3
Unesi element [1,0] 4
Unesi element [1,1] 3
Unesi element [2,0] 1
Unesi element [2,1] 5

U funkciji suma:
Adresa pocetka matrice u memoriji : 1205032
Adresa pocetka drugog retka matrice u memoriji : 1205432
Adresa pocetka treceg retka matrice u memoriji : 1205832

Suma elemenata matrice je 18.000000
U funkciji max:
Adresa pocetka retka matrice u memoriji : 1205032
Najveci element u retku 0 je 3.000000
```

1205432	= 125032 + 1 * MAXSTUP * sizeof(float)
	= 125032 + 400;
1205832	= 125032 + 2 * MAXSTUP * sizeof(float)
	= 125032 + 800;

```
U funkciji max:
Adresa pocetka retka matrice u memoriji : 1205432
Najveci element u retku 1 je 4.000000

U funkciji max:
Adresa pocetka retka matrice u memoriji : 1205832
Najveci element u retku 2 je 5.000000
```

2. Napisati funkciju koja vraća **niz** maksimalnih vrijednosti pojedinih redaka matrice.

```
void maxNiz(float *mat, int brRed, int brStup,
             int maxStup, float *niz) {
// ILI: void maxNiz(float mat[], int brRed, int brStup,
                     int maxStup, float niz[])
    int i, j;
    for (i = 0; i < brRed; i++) {
        niz[i] = mat[i * maxStup];
        for(j = 1; j < brStup; j++)
            if (mat[i * maxStup + j] > niz[i])
                niz[i] = mat[i * maxStup + j];
    }
}
```

3. Napisati funkciju koja vraća duljinu nekog stringa (znakovni niz koji završava null znakom). (*Duljina znakovnog niza jednaka je broju znakova od početka niza do prvog pojavljivanja znaka '\0').*

```
int mojStrlen( char *p ) {
  int i=0;
  while( p[i] != '\0' )
    i++;
  return i;
}
```

Poziv funkcije:

```
...
  char niz[]="Programiranje"; //Dodaje '\0' na kraj niza!!
  printf("Duljina niza je %d", mojStrlen(niz));
...
```

4. Napisati funkciju koja spaja dva znakovna niza u jedan tako da drugi znakovni niz doda na kraj prvoga.

```
char *mojstrcat( char *dest, char *src ) {
  int  i=0, j=0;
  while( dest[i] != '\0' )   // tražimo kraj stringa
    i++;
  while( src[j] != '\0' )
    dest[i++] = src[j++];
  dest[i] = src[j];   // dodajemo '\0' na kraj stringa
  return dest;
}
```

skraćena verzija (samo uz korištenje pokazivača):

```
char *mojstrcat( char *dest, char *src ) {
  char *p = dest;
  while (*dest)
     dest++;

  while(*dest = *src)
     dest++, src++;

  return p;
}
```

5. Napisati funkciju koja kopira sadržaj jednog stringa u drugi.

```
char *mojstrcpy( char *dest, char *src ) {
  char *p = dest;
  while( *src != '\0' ) {
    *dest = *src;
    dest++;
    src++;
  }
  *dest = '\0';
  return p;
}
```

6. Napisati funkciju koja traži zadani znak unutar znakovnog niza i vraća pokazivač na nađeni znak, odnosno NULL ukoliko tog znaka nema.

```
char *mojstrchr( char *s, char c ) {
  while( *s != '\0' ) {
    if( *s == c )
    return s;
    s++;
  }
  return NULL;
}
```

7. Napisati funkciju koja izračunava sumu znamenaka (0-9) u zadanom nizu znakova. Ukoliko u nizu nema niti jedne znamenke, funkcija vraća 0. Npr. za niz znakova "a4dft2" funkcija vraća 6.

```
#include <ctype.h>
unsigned int sumaznam (char *niz) {
    unsigned int suma=0;
    for(;*niz;niz++)
        if (isdigit(*niz)) suma+=*niz-'0';
    return suma;
}
```

8. Napisati funkciju `strncpy` bez korištenja funkcija iz datoteke `<string.h>`

```
char *strncpy(char *dest, const char *src, int n) {
  char *p=dest;
  for (; n > 0 && *src; --n) {
    *dest = *src;
    ++dest;
    ++src;
  }
  if (n) *dest = '\0';
  return p;
}
```

9. Napisati funkciju koja za zadani datum formata **dd.mm.gggg** vraća dan (`dd`), mjesec (`mm`) i godinu (`gggg`), te datum (`datum2`) u formatu **gggg/mm/dd**.
Npr. za datum **12.02.1912** potrebno je vratiti: `dd = 12, mm = 2`, `gggg = 1912`, `datum2` = `"1912/12/02".`

Rješenje:
```
void danMjesecGodina(char *datum, int *dd, int *mm, int *gggg,
     char *datum2){
     sscanf(datum, "%2d.%2d.%4d", dd, mm, gggg);
     sprintf(datum2, "%d/%02d/%02d", *gggg, *mm, *dd);
}
```

Primjer glavnog programa koji poziva funkciju `danMjesecGodina`:
```
void main(){
   char datum[11], datum2[11];
   int d, m, g;

   scanf("%s", datum);

   danMjesecGodina(datum, &d, &m, &g, datum2);
   printf("d = %d, m = %d, g= %d\n", d, m, g);
   printf("datum2 = %s\n", datum2);
}
```

10. Napisati funkciju koja će u nekom znakovnom nizu engleske abecede sva mala slova pretvoriti u velika, a velika u mala. Ako znak nije slovo, treba ga zamijeniti s prazninom.

```
#include <ctype.h>
void mijenjaj( char *s ) {
  for(; *s; s++ ) {
    if( !isalpha(*s) )    *s = ' ';
    else if( islower(*s) ) *s = toupper(*s);
    else if( isupper(*s) ) *s = tolower(*s);
  }
}
```

11.Što ne valja u sljedećoj makro naredbi?

```
#define duplo(x) (x) + (x)
```

Makro naredbom se prije prevođenja programskog kôda zamjenjuje tekst `duplo(x)` s tekstom `(x) + (x)` svugdje gdje se tekst `duplo(x)` pojavljuje u programskom kôdu. To znači da će se npr. izraz:

```
a = duplo(x);
```

zamijeniti s

```
a = (x) + (x);
```

Izraz:

```
a = duplo(x) * 3;
```

će se zamijeniti:

```
a = (x) + (x) * 3;
```

Zbog prioriteta operatora prvo se izračuna `(x) * 3`, pa će konačni rezultat biti pogrešan. Da bi se to izbjeglo potrebno je definirati makro naredbu na sljedeći način:

```
#define duplo(x) ((x) + (x))
```

12.Napisati funkcije `mojfmod`, `mojceil` i `mojfloor` koje obavljaju isti posao kao i ugrađene funkcije **fmod**, **ceil** i **floor**.

x	floor(x)	ceil(x)
5.7	5	6
-5.7	-6	-5
5	5	5
-5	-5	-5

```
double mojfmod(double x,double y) {
  return x - (int)(x / y) * y;
}

double mojfloor(double x) { // vraca cijeli broj <= x
    double floor_x;
    if ((int) x == x)  { // x je cijeli broj
        floor_x = x;
    } else if (x > 0) { // x je pozitivan realan broj
        floor_x = (double)((int)x); // cijeli broj < od x
    } else { // x je negativan realan broj
        floor_x = (double)((int)x - 1); // cijeli broj < od x
    }
    return floor_x;
}
```

```
double mojceil (double x) { // vraca cijeli broj >= x
     double ceil_x;
     if ((int) x == x)  { // x je cijeli broj
          ceil_x = x;
     } else if (x > 0) { // x je pozitivan realan broj
          ceil_x = (double)((int)x + 1); // cijeli broj > od x
     } else { // x je negativan realan broj
          ceil_x = (double)((int)x); // cijeli broj > od x
     }
     return ceil_x;
}
```

Rješenje na drugi način:

```
double mojfloor(double x) {
  return ((int)x == x) ? x :
     x > 0 ? (double)((int)x) : (double)((int)(x - 1));}

double mojceil(double x) {
   return ((int)x == x) ? x :
     x > 0 ? (double)((int)x + 1) : (double)((int)(x));}
```

10. Auditorne vježbe

Primjer 1.
Napisati funkciju koja će transformirati niz znakova tako da se zadani podniz u nizu zamjenjuje (jednim) uskličnikom. Funkcija ima prototip

```
int zamijeni(char *niz, const char *podniz)
```

Funkcija vraća broj obavljenih zamjena.

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

int zamijeni(char *niz, const char *podniz) {

    int br=0, duz;

    duz=strlen(podniz);

    while( niz = strstr(niz,podniz) ) {    // isto što i
//  while( (niz = strstr(niz,podniz)) != NULL ) {
        *niz++ = '!';
        strcpy(niz, niz + duz - 1);
        ++br;
    }
    return br;
}
```

Za ulazni niz

```
Paško pa tak dobio zada tak.

// strstr vrati pokazivač niz na prvi podniz tak

//*niz++='!'; t-->!           Paško pa!ak dobio zadatak.

// strcpy(niz,niz+duz-1);     Paško pa! dobio zadatak.
// ostatak podniza ak se "proguta"
...
```

Nakon poziva funkcije obavljene su 2 zamjene podniza *tak* u !
Konačni niz je

```
Paško pa! dobio zada!.
```

Primjer 2.
Napisati funkciju koja će transformirati niz znakova tako da se zadani podniz u nizu zamjenjuje drugim nizom. Funkcija ima prototip

```
int zamijeni(char *niz, const char *podniz, const char
*noviniz)
```

Funkcija vraća broj obavljenih zamjena.

```
int zamijeni(char *niz, const char *podniz, const char
*noviniz) {
    char pom[512];
    int br=0, duzPod, duzNovi;

    duzPod=strlen(podniz);
    duzNovi=strlen(noviniz);

    while( niz = strstr(niz, podniz) ) {   // traži podniz
        // sačuvaj sve iza nađenog podniza
        strcpy(pom, niz + duzPod);
        // kopiraj novi niz preko podniza
        strcpy(niz, noviniz);
        // vrati spremljeni dio na kraj
        strcat(niz, pom);
        // pomakni pokazivač
        niz += duzNovi;
        ++br;
    }
    return br;
}
```

Primjer glavnog programa:

```
void main() {
    char niz1[40]="Paško Patak dobio zadatak";
    int br=0;
    printf("%s\n", niz1);
    br = zamijeni3(niz1, "tak", "pak");
    printf("%s %d\n", niz1, br);
 }
```

Rezultat izvođenja:
```
Paško pa<b>tak</b> dobio zada<b>tak.</b>
Paško pa<b>pak</b> dobio zada<b>pak.</b> 2
```

Pitanje: zašto je korišteno pomoćno polje?
Odgovor: kad se pomoćno polje ne bi koristilo, nego bi se primijenio algoritam sličan onome u 1. primjeru, problem bi nastao ako je novi niz dulji od podniza (u ostalim bi slučajevima radilo). Napišite program koristeći algoritam iz 1. primjera i analizirajte taj slučaj!

Primjer 3.
Napisati funkciju prema zadanom prototipu

```
char *spoji (char *ime, char *prezime);
```

koja će spajanjem imena (npr. *Ivo*) i prezimena (npr. *Ivić*) osobe načiniti niz znakova koji sadrži prezime i ime osobe odvojene zarezom i prazninom (npr. *Ivić, Ivo*). Funkcija vraća pokazivač na rezultat. Ime i prezime moraju ostaju nepromijenjeni, a pretpostavlja se da imaju pratećih praznina. Rezultat ne smije imati pratećih praznina.

```
char *spoji (char *ime, char *prezime) {
  static char s[512];
  int zadnjiznak, i, j;
  for (i = 0; prezime[i]; i++) {
    s[i] = prezime[i];
    if (s[i] != ' ') {
       zadnjiznak = i;
    }
  }
  s[zadnjiznak+1] = ',';  s[zadnjiznak+2] = ' ';
  for (i = 0, j = zadnjiznak+3; ime[i]; i++, j++) {
    s[j] = ime[i];
    if (s[j] != ' ') {
       zadnjiznak = j;
    }
  }
  s[zadnjiznak+1] = '\0';
  return s;
}
```

Pitanje: da li bi funkcija dobro radila kada varijabla s ne bi bila deklarirana kao static?
Odgovor: ako se iz funkcije vraća pokazivač na lokalno polje iz funkcije, ispravno bi to polje trebalo deklarirati kao static kako bi se vrijednost polja sačuvala i nakon izlaska iz funkcije.

a) glavni program

```
main () {
  char osoba1[40]; char osoba2[40];
// dodjeljivane vrijednosti stringu znakom = NE MOŽE
// NIJE DOBRO  osoba1 = spoji("ime", "prezime");
  strcpy (osoba1, spoji("Ana Marija", "Matic   "));
  strcpy (osoba2, spoji("Ivana ", "Brlic Mazuranic"));
  printf ("\nPrva osoba=%s", osoba1);
  printf ("\nDruga osoba=%s", osoba2);
  printf ("\nOpet prva osoba=%s ", osoba1);
  return 0;
}
```

Rezultat izvođenja programa pod a):

```
    Prva osoba je Matic, Ana Marija
    Druga osoba je Brlic Mazuranic, Ivana
    Opet prva osoba je Matic, Ana Marija
```

b) glavni program

```
main () {
  char osoba1[40], osoba2[40];
  char *p1, *p2;

  p1 = spoji("Ana Marija", "Matic  ");
  printf ("\nPrva osoba=%s", p1);

  p2 = spoji("Ivana ", "Brlic Mazuranic");
  printf ("\nDruga osoba=%s", p2);
  printf ("\nOpet prva osoba=%s", p1);
  return 0;

}
```

Rezultat izvođenja programa pod b):

```
    Prva osoba je Matic, Ana Marija
    Druga osoba je Brlic Mazuranic, Ivana
    Opet prva osoba je Brlic Mazuranic, Ivana
```

Pitanje: zašto se u b) slučaju ispisuje krivi rezultat?
Odgovor: budući da se u funkciji rezultat sprema uvijek u **ISTU** statičku varijablu, uzastopno pozivanje funkcije ima za posljedicu prepisivanje starog sadržaja statičke varijable sa svakim pozivom funkcije. Ako se želi sačuvati rezultat svakog poziva funkcije, potrebno je rezultat kopirati u novi niz znakova kao u slučaju a).

Primjer 4.
Potrebno je napisati funkciju koja vraća i generira lozinku kao niz slučajno odabranih brojeva duljine 6 znamenaka i glavni program za ispis 10 novo generiranih lozinki.

Općenito za generiranje cijelog broja iz intervala [n1, n2] treba pozvati sljedeće:

```
  //"inicijalizira" generator pseudo slučajnih brojeva
  srand ((unsigned) time(NULL));
  //time(NULL) vraća  broj sekundi od 1.1.1970

     (int) ((float)rand()/(RAND_MAX+1)*(n2-n1+1) + n1);
  ili
     n1 + rand()%(n2-n1+1)

  srand (10);  i zatim rand...
  generirao bi se uvijek isti niz "slučajnih" brojeva
```

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

char* lozinka() {
  static char niz[6+1];
  int sl_broj,i;
  // ako bi se srand pozvao na ovom mjestu,
  // vraćala bi se uvijek ista lozinka!
  for(i=0; i<6; i++) {
     sl_broj = (int)((float)rand()/(RAND_MAX+1)*10 + 0);
         //int rand (void);
     niz[i] = '0' + sl_broj;
  }
  niz[i] = '\0';
  return niz;
}
```

```
void main() {
  int i;
  // srand pozvati jednom na početku
  srand ((unsigned) time(NULL));
  for (i=1; i<=10 ; i++)
     printf("\n %2d. generirana lozinka je %s"
              ,i, lozinka() );
}
```

Primjer 5.
Što se ispisuje sljedećom naredbom ?

```
float f=1.234567;
printf("%10s%6.2f\n","niz10",f);
```

Rješenje:

					n	i	z	1	0			1	.	2	3

Primjer 6.
Formatiranje teksta u HTML jeziku postiže se naredbama oblika

<NAREDBA PARAMETAR_1=XX ... PARAMETAR_N=YY> tekst koji želimo formatirati </NAREDBA>.

Dozvoljeno je gniježđenje naredbi tako da npr.

<FONT COLOR=\"RED\"><STRONG>Hello, world</STRONG></FONT>

označava da se tekst "Hello, world" mora ispisati crvenom bojom i podebljano. Tekst koji se formatira u sebi ne smije sadržavati znakove '<' i '>'.

Napisati funkciju koja će primati ulazni niz znakova u HTML formatu i, zanemarujući sve naredbe za formatiranje, ispisati samo čisti tekst koji se želi formatirati. U gore navedenom primjeru treba ispisati "Hello, world".

```
void ispis(char *html) {
   int br = 0;
   while (*html) {
      if (*html=='<') {
         br++;
      }
      if (!br) {
         putchar(*html);
      }
      if (*html=='>') {
         br--;
      }
      html++;
   }
}
```

Primjer 7.
Napisati funkciju za generiranje skupa slučajnih brojeva i ispis tablice brojeva prema zadanim ulaznim parametrima.

```
void ispis(int br_redaka, int br_stupaca, int izvuci)
```

Funkcija generira `izvuci` slučajnih brojeva od [1, br_redaka * br_stupaca].
Izvučeni brojevi generiraju se kao slučajni broj unutar funkcije.
Ispis mora biti u obliku tablice brojeva redom od 1 do br_redaka*br_stupaca, a izvučeni brojevi trebaju biti označeni znakom *x*.

Npr. za tablicu dimenzija 13x3, i 7 brojeva za izvlačenje poziva se funkcija
`ispis(13, 3, 7)`
a ukoliko su npr. izvučeni brojevi 5,14,3,31,19,30,39 ispis treba izgledati ovako:

```
+-----+-----+-----+
|   1 |   2 |   x |
|   4 |   x |   6 |
|   7 |   8 |   9 |
|  10 |  11 |  12 |
|  13 |   x |  15 |
|  16 |  17 |  18 |
|   x |  20 |  21 |
|  22 |  23 |  24 |
|  25 |  26 |  27 |
|  28 |  29 |   x |
|   x |  32 |  33 |
|  34 |  35 |  36 |
|  37 |  38 |   x |
+-----+-----+-----+
```

Svaki broj ispisuje se u formatu |_###_|

```
/*Rješenje pretpostavlja da je br_redaka * br_stupaca < 100. Rješenje
bez tog ograničenja uključivalo bi dinamičku alokaciju memorije! */

void ispis (int br_redaka,int br_stupaca, int izvuci) {
     int i,j, n, k, izv_broj;
     int p[100] = {0};

     srand ((unsigned) time(NULL));
     n = br_redaka*br_stupaca;
     i = 0;
     while (i<izvuci)  {
       izv_broj = (int)((float)rand()/(RAND_MAX+1)*n + 1);

       // ako je ponovno generiran isti broj preskoci ga
       if (p[izv_broj-1]) continue;

       p[izv_broj-1] = izv_broj;
       i++;
     }

      // zaglavlje tablice
     printf("\n");
     for(j=0; j<br_stupaca; ++j) printf("+-----");
     printf("+\n");

     for(k = 0, i = 0; i<br_redaka; ++i) {
          for(j = 0; j<br_stupaca; ++j, k++) {
               if (p[k])
                    printf("|   x ");
               else
                    printf("| %3d ",k+1);
          }
          printf("|\n");
     }
     //  kraj tablice
     for(j=0; j<br_stupaca; ++j) printf("+-----");
     printf("+\n");

}
```

11. Auditorne vježbe

1. U trenutnom direktoriju nalaze se datoteke čija imena se sastoje od 5 slova engleske abecede. Napisati program koji ispisuje imena svih datoteka iz tog direktorija koja počinju sa slovima PROG.

```
#include <stdio.h>
void main() {
  char naziv[]= "PROGx", zadnjeslovo;
  FILE *f;

  for (zadnjeslovo ='A'; zadnjeslovo <= 'Z';
       zadnjeslovo ++){
    naziv[4]= zadnjeslovo;
    if ((f = fopen(naziv, "r")) != NULL) {
      printf("%s\n", naziv);
      fclose(f);
    }
  }
}
```

2. Napisati funkciju `prebroji` koja će prebrojati broj znamenki u formatiranoj datoteci. Ime datoteke predaje se kao argument.

```
#include <stdlib.h> //zbog exit

int prebroji(char *ime) {
  FILE *f;
  char c;
  int br = 0;

  if ((f = fopen(ime, "r")) == NULL) {
    printf("Pogreska kod otvaranja datoteke %s!\n", ime);
    exit(1);
  }

  while ((c = fgetc(f)) != EOF) {
    if(isdigit(c)) br++;
  }

  fclose(f);
  return br;
}
```

3. Napisati program koji će čitajući slijednu **formatiranu** datoteku u kojoj je zapis o studentu oblika:
   mmmmNNpppopppopppopppo…

   gdje je:
   `mmmm` – matični broj studenta
   `NN` - broj položenih ispita
   `ppp` – šifra predmeta
   `o` – ocjena (parova `pppo` ima `NN`)

   ispisati za svakog studenta matični broj i prosječnu ocjenu.

```
#include <stdio.h>
void main () {
  FILE *fi;
  int i, n, mbr, sif_pred, ocjena;
  float suma;

  fi = fopen ("ispiti", "r");
  /* provjera otvaranja */

  while (fscanf(fi, "%4d%2d", &mbr, &n) == 2) {
    suma = 0;
    for (i = 0; i < n; i++) {
      fscanf (fi, "%3d%1d", &sif_pred, &ocjena);
      suma += ocjena;
    }
    printf ("Prosjek od %d je %f\n", mbr, suma/n);
  }

  fclose (fi);
}
```

4. Načiniti funkciju koja će u slijednoj formatiranoj datoteci čije se ime zadaje kao argument funkcije:
   a) prebrojati koliko ima praznih redaka
   b) prebrojati koliko ima redaka u kojima se nalazi samo jedan znak

Funkcija mora biti tipa `void` i rezultate vratiti preko argumenata. Možemo pretpostaviti da redak nije duži od 4095 znaka uključujući znak za novi red.

```
#include <stdio.h>
void broji(char *ime, long *brpraz, long *brred1znak) {
  FILE *f;
  char buf[4096];
  *brpraz = 0;
  *brred1znak = 0;

  f = fopen (ime, "r");
  /* provjera otvaranja */

  while (fgets(buf, 4096, f)) {
    if (strlen (buf) == 1) ++(*brpraz);   /* u liniji je \n */
    if (strlen (buf) == 2) ++(*brred1znak);
  }
  fclose(f);
}
```

5. Napisati funkciju :
   `int traziDat(FILE *dat, char *podniz)`
   koja će za zadanu datoteku ispisati redne brojeve redaka u kojima se pojavljuje zadani podniz i broj pojavljivanja podniza u tom retku. Funkcija vraća ukupan broj pojavljivanja zadanog podniza u datoteci.

```
int traziDat(FILE *dat, char *podniz) {
  int rbrRetka = 0, brPojav, ukPojav = 0;
  char redak[MAX+1], *p;

  // fgets cita najvise MAX znakova ili dok ne dodje do \n,
  // a na kraj ulaznog niza dodaje \0
  while(fgets(redak, MAX + 1, dat)) {
    ++ rbrRetka;
    brPojav = 0;
    p = redak;

    // broji pojavu podniza u procitanom retku
    while(p = strstr(p, podniz)) {
      ++ brPojav;
      p += strlen(podniz);
    }

    if (brPojav > 0) { // ako se podniz pojavio u retku
      printf("%3d.%5d\n", rbrRetka, brPojav);
      ukPojav += brPojav;
    }
  }
  return ukPojav;
}
```

6. Napisati funkciju čiji je prototip:
   `int najcescaZnamenka(char *imeDatoteke, char *z)`
   koja u formatiranoj datoteci zadanog imena `imeDatoteke` pronalazi dekadsku znamenku koja se najčešće pojavljuje. Ako se dvije ili više znamenki pojavljuju jednak broj puta, potrebno je vratiti onu znamenku koja ima najmanju vrijednost ASCII koda.
   Ako se datoteka ne može otvoriti funkcija treba vratiti –1, ako se u datoteci ne nalazi niti jedna znamenka funkcija treba vratiti 0, a inače funkcija treba vratiti 1.

```
int najcescaZnamenka(char *imeDatoteke, char *z) {
  FILE *fin;
  char c;
  int znamenke[10] = {0};
  int imaNesto = 0;
  int max, i;

  fin = fopen(imeDatoteke, "r");
  if (fin == NULL) return -1;

  while ((c = fgetc(fin)) != EOF) {
    if ((c >= '0') && (c <= '9')) {
      imaNesto = 1;
      znamenke[c - '0']++;
    }
  }
  fclose(fin);

  if (imaNesto) {
    max = 0;
    // Ako se dvije znamenke pojavljuju jednak broj puta,
    // vratiti će onu s manjim ASCII kodom, jer će max prije
    // postaviti na nju
    for (i = 1; i < 10; ++ i) {
      if (znamenke[i] > znamenke[max]) max = i;
    }

    // znamenka koja se najvise puta pojavila
    *z = (char)('0' + max);
    return 1;
  }

  return 0;
}
```

7. Za studenta se vodi evidencija o sljedećim podacima: ime (najviše 20 znakova), prezime (najviše 25 znakova), prosjek ocjena (realan broj). Napisati funkcije koje će omogućiti:
   - dodavanje novog zapisa o studentu u slijednu formatiranu datoteku `student.txt`
   - ispis podataka o studentu na temelju zadanog prezimena (ukoliko ima više studenata sa zadanim prezimenom, potrebno ih je sve ispisati)

Ako datoteka postoji, zapisi se dodaju na kraj postojeće datoteku. Ukoliko datoteka `student.txt` ne postoji, potrebno ju je stvoriti.

ime	**\n**	**prezime**	**\n**	**prosjek**	**\n**	**ime**	**\n**	**prezime**	**\n**	**prosjek**	**\n**	**...**

```
int dodaj(char *ime, char *prezime, float prosjek) {
  FILE *fin;
  // Pozicioniram se na kraj, ako datoteke nema
  // otvara se
  fin = fopen("student.txt", "a");
  if (fin == NULL) return -1;

  fprintf(fin, "%s\n%s\n%f\n", ime, prezime, prosjek);
  fclose(fin);
  return 0;
}

int ispisi(char *prezime) {
  FILE *fout;
  char prez2[25+1];
  char ime2[20+1];
  float prosjek2;

  fout = fopen("student.txt", "r");
  if (fout == NULL) return -1;

  while(fscanf(fout,"%20[^\n]%*c%25[^\n]%*c%f%*c",
                      ime2, prez2, &prosjek2) > 0) {
    if (strcmp(prez2, prezime) == 0) {
      printf("\n%s, %s:%5.2f", prez2, ime2, prosjek2);
    }
  }
  fclose(fout);

  return 0;
}
```

12. Auditorne vježbe

1. U slijednoj formatiranoj datoteci **`podaci.txt`** nalaze se zapisi o uspjehu studenta na kontrolnoj zadaći: jmbag (10 znamenki, nije veći od 2000000000), ime (najviše 20 znakova), prezime (najviše 25 znakova), te bodovi po zadacima. Podaci su unutar zapisa odvojeni delimiterom (#).

Izgled datoteke:

```
0036363636#Ana#Anic#40#50#100#90#10#
0035353555#Marko s dugim imenom#Matic s dugim prezimenom#0#0#0#50#50#
```

Načiniti program koji će sadržaj datoteke prepisati na zaslon u obliku:

```
        1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
JMBAG     •Prezime, Ime                    •  1•  2•  3•  4•  5••Ukupno
0036363636•Anic, Ana                       • 40• 50•100• 90• 10••   290
0035353555•Matic s dugim prezimenom, Mark•  0•  0•  0• 50• 50••   100
…
```

Znak • označava obaveznu prazninu u ispisu. Osjenčane brojke ne treba ispisivati, one su ovdje da naglase strukturu zapisa.

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

main() {
     long jmbag;
     char ime[20+1], prezime[25+1], prezIime[47+1];
     int o[5];
     FILE * fin;

     if ( (fin=fopen("podaci.txt", "r")) == NULL) {
          printf("Ne mogu otvoriti datoteku 'podaci'!\n");
          exit(1);
     }
     printf("%-10s %-30s   1   2   3   4   5  Ukupno\n",
     "JMBAG", "Prezime, ime");
     while ( fscanf(fin, "%ld#%[^#]#%[^#]#%d#%d#%d#%d#%d#",
     &jmbag, ime, prezime, o, o+1, o+2, o+3, o+4) == 8 ) {
          strcat(strcat(strcpy(prezIime, prezime), ", "), ime);
```

```
        // ili dulje:
        //strcpy(prezIime, prezime);
        //strcat(prezIime, ", ");
        //strcat(prezIime, ime);

        printf("%010d %-30.30s %3d %3d %3d %3d %3d  %6d\n",
                jmbag, prezIime, o[0], o[1], o[2], o[3],
                o[4], o[0]+o[1]+o[2]+o[3]+o[4]);
    }
    fclose(fin);
    return 0;
}
```

2. U slijednoj neformatiziranoj datoteci "cijene" nalaze se podaci o cijenama određenih proizvoda. Prvi podatak u datoteci je broj podataka koji se nalazi u datoteci (long) a zatim slijedi toliki broj parova šifre (int) i cijene proizvoda (float). Napisati program koji će izračunati prosječnu cijenu proizvoda.

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

void main( void ) {
    FILE  *fp;
    long broj, i, sifra;
    float cijena, suma=0;

    if( (fp = fopen("cijene", "rb")) == NULL ) {
        printf("Ne moze otvoriti datoteka cijene");
        exit(1);

    }
    fread(&broj, sizeof(long), 1, fp);
    for( i=0; i<broj; i++ ) {
        fread(&sifra, sizeof(long), 1, fp);
        fread(&cijena, sizeof(float), 1, fp);
        suma += cijena;
    }
    printf("\nProsječna cijena je %f", suma/broj);
    fclose(fp);
}
```

3. Načiniti funkciju koja će kao rezultat vratiti broj popunjenih zapisa direktne neformatizirane datoteke u kojoj su zapisi oblika:

matični broj (int)
prezime i ime (40 +1 znak)

Redni broj zapisa odgovara matičnom broju.

```
long brpunih (FILE *f) {
     struct {
       int mbr;
       char prime[40+1];
     } zapis;
     long i = 0, n = 0;
     while (fread (&zapis, sizeof (zapis), 1, f) > 0) {
       if (zapis.mbr == ++i) ++n;
     }
     return n;
}
```

4. Na magnetskom disku nalazi se direktna neformatizirana datoteka kojoj su zapisi oblika

matični broj `char[8+1]`
iznos plaće `double`

Redni broj zapisa odgovara numeričkoj vrijednosti matičnog broja. Načiniti funkciju koja će povećati plaću zadanoj osobi za zadani postotak. Pretpostaviti da je datoteka otvorena u pozivajućem programu. U pozivajućem programu su učitani i matični broj i postotak. Funkcija ne smije koristiti globalne varijable.

```
void povecaj(char *mbr, int posto, FILE *fp) {
  struct {
    char mbr[8+1];
    double placa;
  } zapis;
  fseek (fp, (atoi(mbr) - 1) * sizeof (zapis), SEEK_SET);
  fread (&zapis, sizeof (zapis), 1, fp);
  zapis.placa += zapis.placa * posto/100;
  fseek (fp, -sizeof (zapis), SEEK_CUR);
  fwrite (&zapis, sizeof (zapis), 1, fp);
}
```

5. Neformatizirana datoteka “knjige” ima zapise oblika

šifra knjige	long
ISBN	15 znakova
autor knjige	40 znakova
godina izdanja	int
naslov	50 znakova
kritika	1-obavezno preskočiti, 2-preskočiti, ... , 5-obavezno pročitati

Prazan zapis prepoznaje se po šifri knjige 0. Napisati funkciju koja će ispisati naslove knjiga s natprosječnom kritikom. Ispisati maksimalno 100 naslova.

```
struct zapis {
     long sifra;
     char ISBN[15+1];
     char autor[40+1];
     int  god_izd;
     char naslov[50+1];
     int  kritika;
};

void natprosj( void ) {
     struct zapis z;
     FILE   *fp;
     float  prosj;
     long  suma_krit;
     int    br, br_ispis;

     if( ( fp = fopen("Knjige", "rb") ) == NULL ) {
          puts( "Greška u otvaranju datoteke ");
          return;
     }

     suma_krit = br = 0;
     while( fread(&z, sizeof(z), 1, fp) == 1 ) {
          if( z.sifra != 0 ) {
               suma_krit += z.kritika;
               br++;
          }
     }
     prosj = (float) suma_krit / br;

     rewind(fp);
     br_ispis = 0;
```

```
        while( fread(&z, sizeof(z), 1, fp) == 1 ) {

          if( z.kritika>prosj && br_ispis<100 && z.sifra!=0 )
          {
            printf("\n%s", z.naslov );
            br_ispis++;

          }
        }
    }
```

6.Što će ispisati sljedeći programi i zašto?

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

main() {
    FILE *fB, *fT;
    char c = '\n';

    if ( (fB=fopen("Binary.dat", "wb")) == NULL) {
        printf("Ne moze otvoriti 'Binary.dat'\n");
        exit(1);
    }
    fputc('\n', fB);
    printf("Nakon upisivanja \\n znaka u Binary.dat
            ftell=%d\n", ftell(fB));
    if ( (fT=fopen("Text.txt", "w")) == NULL) {
    // ili if ( (fT=fopen("Text.txt", "wt")) == NULL) {

        printf("Ne moze otvoriti 'Text.txt'\n");
        fclose(fB);
        exit(1);
    }
    fputc('\n', fT);
    printf("Nakon upisivanja \\n znaka u Text.txt ftell=%d\n",
        ftell(fT));
    fclose(fT);
    fclose(fB);
}
```

Program će ispisati :

```
Nakon upisivanja \n znaka u Binary.dat ftell=1
Nakon upisivanja \n znaka u Text.txt ftell=2
```

Obrazloženje: Kod "b" moda , \n znak se upiše kao 1 byte vrijednost: `0x0A` (10)
Ako se eksplicitno ne upiše mod kod otvaranja datoteke,
podrazumijeva se tekst-t mod.
Kod "t" moda upisivanja \n znak se upiše kao 2 byte-a (vrijednost: `0x0D 0x0A` (13 10)).

"t" mod pri pisanju u datoteku radi pretvorbu byte `0A` u `0D 0A` tj. ukoliko je vrijednost jednog byte-a neke varijable `0A`, upisat će se jedan byte više
npr. `short s=10;` će u datoteci otvorenoj "t" modom, nakon upisivanja `fwrite` funkcijom zauzeti 3 byte-a.
Vrijedi i obrat kod čitanja: kad se u datoteci otvorenoj "t" modom naiđe na dva byte-a vrijednosti `0D 0A` pročitat će se kao 1 byte vrijednosti `0A`.

"b" mod pri pisanju i čitanju NE obavlja takvu (niti bilo kakvu drugu) pretvorbu.

Direktne datoteke zbog gore navedenih razloga treba otvarati u "b" modu jer inače može doći do prepisivanja sadržaja što je pokazano sljedećim primjerom:

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

main() {
     FILE *fD;
     int i,j;

     if ( (fD=fopen("direktna.dat", "w+")) == NULL) {
          printf("Ne moze otvoriti 'direktna.dat'\n");
          exit(1);
     }
     for (i=1; i<12; i++) {
          // pomak na pocetak i-tog zapisa:
          fseek(fD, (i-1)*sizeof(int), SEEK_SET);
          fwrite(&i, sizeof(int), 1, fD);
          printf("zapis: %2d vrijednost %2d; pozicija nakon
                    zapisivanja: %3d\n", i, i, ftell(fD));
     }
     for (i=1; i<12; i++) {
          // pomak na pocetak i-tog zapisa:
          fseek(fD, (i-1)*sizeof(int), SEEK_SET);
          fread(&j, sizeof(int), 1, fD);
          printf("U %2d. zapisu vrijednost %2d\n", i, j);
     }
     fclose(fD);
}
```

Program će ispisati sljedeće:

```
zapis:  1 vrijednost  1; pozicija nakon zapisivanja:   4
zapis:  2 vrijednost  2; pozicija nakon zapisivanja:   8
zapis:  3 vrijednost  3; pozicija nakon zapisivanja:  12
zapis:  4 vrijednost  4; pozicija nakon zapisivanja:  16
zapis:  5 vrijednost  5; pozicija nakon zapisivanja:  20
zapis:  6 vrijednost  6; pozicija nakon zapisivanja:  24
zapis:  7 vrijednost  7; pozicija nakon zapisivanja:  28
zapis:  8 vrijednost  8; pozicija nakon zapisivanja:  32
zapis:  9 vrijednost  9; pozicija nakon zapisivanja:  36
zapis: 10 vrijednost 10; pozicija nakon zapisivanja:  41
zapis: 11 vrijednost 11; pozicija nakon zapisivanja:  44
U  1. zapisu vrijednost 1
U  2. zapisu vrijednost 2
U  3. zapisu vrijednost 3
U  4. zapisu vrijednost 4
U  5. zapisu vrijednost 5
U  6. zapisu vrijednost 6
U  7. zapisu vrijednost 7
U  8. zapisu vrijednost 8
U  9. zapisu vrijednost 9
U 10. zapisu vrijednost 184549386
U 11. zapisu vrijednost 11
```

Zašto je 10. zapisu vrijednost=184549386?

Obrazloženje: prilikom upisivanja 10. zapisa u datoteku je zapisano `0D 0A 00 00 00` (5 byte umjesto 4) pa je ovaj posljednji byte (`00`) prepisan prilikom upisivanje 11 zapisa sa vrijednosti `0B`)

Znači u datoteci od pozicije 36 byte-a su podaci: `0D 0A 00 00 0B`. Prilikom čitanja tih byte-ova, a zbog "t" moda pročita se tih 5 byte-ova i dobije se `0A 00 00 0B` što predstavlja broj `0x0B00000A` (184549386)

13. Auditorne vježbe

1. Potrebno je napisati funkciju `mojstrdup` koja će biti implementacija standardne `strdup` funkcije. (Objašnjenje `strdup` funkcije:
`char * **strdup** *(const char *s)*`–funkcija kopira niz znakova *s* u novi niz znakova koji se alocira pomoću funkcije `malloc`. Niz znakova *s* završen je s null znakom (`'\0'`). Funkcija `strdup` vraća pokazivač na novi niz znakova, odnosno `NULL` pokazivač ukoliko funkcija `malloc` nije uspjela rezervirati dovoljno memorije ili je kao ulazni podatak bio proslijeđen `NULL` pokazivač.

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

char * mojstrdup(const char *s) {
    char *retVal = NULL;
    if (s)
        if (retVal=(char *) malloc(strlen(s)+1))
            strcpy(retVal, s);
    return retVal;
}
```

Napomena: funkcija `strdup` (i `mojstrdup`) dinamički alocira, a ne oslobađa memoriju. Zbog toga je potrebno u pozivajućoj funkciji (kada duplikat niza znakova više nije potreban) osloboditi blok memorije funkcijom `free` koji je nastao kao rezultat funkcije `strdup` (i `mojstrdup`). Mogući izgled glavnog programa:

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

int main() {
    char s[] = "Prvi string", *p;
    int i;

    for (i=0; i<20; i++) {
        p = mojstrdup(s);
        printf("%2d. Poziv adresa je %d a string %s\n",
            i, p, p);
        free(p);
    }
}
```

2. Napisati općeniti potprogram (koji radi za polja proizvoljnih dimenzija) za pronalaženje najvećeg člana dvodimenzionalnog cjelobrojnog polja.

 datoteka `najveci.c`

```
int najveci(int *mat, int brRed, int brStup, int maxStup) {
    int i, j, retVal = *mat;
    for (i = 0; i < brRed; i++)
        for(j = 0; j < brStup; j++)
            if (mat[i * maxStup + j] > retVal)
                retVal = mat[i * maxStup + j];
    return retVal;
}
```

a) Napisati glavni program u kojem se zadaje broj redaka i stupaca matrice, te unose elementi matrice. Nakon toga pomoću funkcije `najveci` pronaći i ispisati najveći element matrice.

 datoteka `glavni1.c`

```
#include <stdio.h>
#define MAXRED 10
#define MAXSTUP 20

// prototip = deklaracija funkcije
int najveci(int *, int, int , int);

int main() {
    int m[MAXRED][MAXSTUP];
    int brRed, brStup, i, j;
    printf("Unesite broj redaka i stupaca: ");
    scanf("%d %d", &brRed, &brStup);
    for (i=0; i<brRed; i++)
        for (j=0; j<brStup; j++) {
            printf("m[%2d][%2d] = ", i, j);
            scanf ("%d", &m[i][j]);
        }
    printf("Najveci element polja je: %d\n",
            najveci(&m[0][0], brRed, brStup, MAXSTUP));
    return 0;
}
```

b) Napisati glavni program u kojem se zadaje broj redaka i stupaca matrice, te unose elementi matrice uz učitavanje podataka u središnju memoriju uz minimalni utrošak središnje memorije. Nakon toga pomoću funkcije `najveci` pronaći i ispisati najveći element matrice.

datoteka `glavni2.c`

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

int najveci(int *, int, int , int);

int main() {
    int *m;
    int brRed, brStup, i, j;
    printf("Unesite broj redaka i stupaca: ");
    scanf("%d %d", &brRed, &brStup);
    m = (int *) malloc(brRed * brStup * sizeof(int));
    if (m) {
        for (i=0; i<brRed; i++)
            for (j=0; j<brStup; j++) {
                printf("m[%2d][%2d] = ", i, j);
                scanf ("%d", &m[i* brStup + j]);
            }
        printf("Najveci element polja je: %d\n",
                najveci(&m[0], brRed, brStup, brStup));
        free(m);
    } else printf("Nema dovoljno memorije!\n");
    return 0;
}
```

NAPOMENA: uočiti da se kod dinamičke alokacije polja maksimalni broj stupaca (i redaka) "podudara" s stvarnim (korisnički zadanim) brojem stupaca (i redaka).

3. Zadana je slijedna formatirana datoteka "`mjesta.txt`" koja sadrži zapise sljedećeg oblika:

```
Ada 31214 Laslovo
Adamovec 10363 Belovar
Adzamovci 35422 Zapolje
Alaginci 34000 Pozega
Alan 53271 Krivi Put
Bankovci 34000 Pozega
...
```

Prvi dio zapisa je naziv mjesta, zatim slijedi poštanski broj i na kraju je naziv poštanskog ureda (NAPOMENA: više mjesta može pripadati jednom poštanskom uredu, u nazivu mjesta ne pojavljuju se brojke, najmanji poštanski broj je 10000 a najveći je 53296, niti jedan naziv nije dulji od 64 slova).

Potrebno je napisati program kojim će se poštanski brojevi i nazivi poštanskih ureda prepisati u direktnu neformatiranu datoteku "`mjesta.dat`". Redni broj zapisa u datoteci odgovara poštanskom broju umanjenom za 10000 (zbog uštede prostora jer ne postoje manji poštanski brojevi).

 datoteka `strukt.h`

```
typedef struct {
     int pbr;
     char naziv[64];
} sMjesto;
```

Koliko je velika struktura? (68 By)

 datoteka `mjesto.c`

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

int main() {
     FILE * fUlaz, * fIzlaz;
     sMjesto mjesto, mjesto2;

     if ((fUlaz = fopen("mjesta.txt", "r")) == NULL)
          exit(-1);
     if ((fIzlaz = fopen("mjesta.dat", "wb")) == NULL)
          exit(-1);
     while (fscanf(fUlaz, "%*[^0-9]%d %[^\n]%*c",
               &mjesto.pbr, mjesto.naziv) == 2) {
          fseek(fIzlaz, (mjesto.pbr - 10000L) * sizeof(mjesto),
                                                  SEEK_SET);
          fwrite(&mjesto, sizeof(mjesto), 1, fIzlaz);
     }
     fclose (fUlaz);
     fclose (fIzlaz);
}
```

Koliko je velika datoteka? 2.944.196 By = 68 * (53296 – 10000 + 1)

Što znači format "`%*[^0-9]%d %[^\n]%*c`" ?

4. Napisati program koji će u datoteci "`mjesta.dat`" iz prethodnog zadatka za zadani poštanski broj ispisati naziv poštanskog ureda. Poštanski broj potrebno je zadati kao argument komandne linije. Ukoliko ne postoji poštanski ured s zadanim brojem ispisati odgovarajuću poruku.
(NAPOMENA: koristi se i datoteka "`strukt.h`" iz prethodnog zadatka)

datoteka `IspisMjesta.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include "strukt.h"

int main(int brojA, char *poljeA[]) {
     // u engl. literaturi koriste se argc i argv
     // brojA == argc == argument count
     // poljeA == argv == argument vector
     FILE * fUlaz;
     sMjesto mjesto;
     int pbr;

     if (brojA != 2) {
          printf("Uporaba:%s PBR\n", poljeA[0]);
          exit(-1);
     }
     pbr = atoi(poljeA[1]);
     if ((fUlaz = fopen("mjesta.dat", "rb")) == NULL)
          exit(-1);
     fseek(fUlaz, (pbr - 10000L) * sizeof(mjesto), SEEK_SET);
     if (fread(&mjesto, sizeof(mjesto), 1, fUlaz) == 1
          && pbr == mjesto.pbr)
          printf("Pbr:%d Naziv:%s\n", mjesto.pbr, mjesto.naziv);
     else
          printf("Ne postoji mjesto s Pbr:%d\n", pbr);
     fclose (fUlaz);
}
```

Primjer izvođenja programa:

```
F:\>ispismjesta
Uporaba:ispismjesta PBR
F:\>ispismjesta 10000
Pbr:10000 Naziv:Zagreb
F:\>ispismjesta 21000
Pbr:21000 Naziv:Split
F:\>ispismjesta 21001
Ne postoji mjesto s Pbr:21001
F:\>
```

4. Napisati program koji će u direktnoj neformatiranoj datoteci "`mjesta.dat`" iz prethodnog zadatka ažurirati naziv poštanskog ureda. Poštanski broj i naziv poštanskog ureda potrebno je zadati kao argumente komandne linije. Ukoliko ne postoji poštanski ured sa zadanim brojem, dodati novi zapis i ispisati odgovarajuću poruku.
(NAPOMENA: koristi se i datoteka "`strukt.h`" iz prethodnog zadatka)

datoteka `azuriraj.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "strukt.h"

int main(int brojA, char *poljeA[]) {
     FILE * fUlazIzlaz;
     sMjesto mjesto;
     int pbr;

     if (brojA != 3) {
          printf("Uporaba:%s PBR Naziv\n", poljeA[0]);
          exit(-1);
     }
     pbr = atoi(poljeA[1]);
     if ((fUlazIzlaz = fopen("mjesta.dat", "r+b")) == NULL)
          exit(-1);
     fseek(fUlazIzlaz, (pbr - 10000L) * sizeof(mjesto), SEEK_SET);
     if (fread(&mjesto, sizeof(mjesto), 1, fUlazIzlaz) == 1
          && pbr == mjesto.pbr) // vec postoji
          printf("Stari naziv:%s\n", mjesto.naziv);
     else
          printf("Ne postoji mjesto s Pbr:%d\n", pbr);
     // vrati se na prethodni zapis
     fseek(fUlazIzlaz, -sizeof(mjesto), SEEK_CUR);
     mjesto.pbr = pbr;
     strcpy(mjesto.naziv, poljeA[2]);
     fwrite(&mjesto, sizeof(mjesto), 1, fUlazIzlaz);
     fclose (fUlazIzlaz);
}
```

Što će se desiti ako se zada:

```
azuriraj 11000 Neko Mjesto
```

P: Kako riješiti nazive s više riječi?
O: Staviti naziv u dvostruke navodnike (ne može jednostruke):

```
azuriraj 11000 "Neko Mjesto"
```

5. Napisati program koji će popunjene zapise iz direktne neformatirane datoteke "`mjesta.dat` " iz prethodnog zadatka prepisati u slijednu neformatiranu datoteku "`mjestas.dat`". Nakon toga izmjeriti vrijeme potrebno za 1000 puta pročitati zapis s poštanskim brojem 40000.

```
#include <stdio.h>
#include <stdlib.h>
#include "strukt.h"
#include <time.h>

int main() {
	FILE * fUlaz, * fIzlaz;
	sMjesto mjesto;
	int i;
	time_t poc, kraj;

	if ((fUlaz = fopen("mjesta.dat", "rb")) == NULL)
		exit(-1);
	if ((fIzlaz = fopen("mjestas.dat", "wb")) == NULL)
		exit(-1);
	i = 0;
	while (fread(&mjesto, sizeof(mjesto), 1, fUlaz) == 1) {
		if (i == mjesto.pbr - 10000)
			fwrite(&mjesto, sizeof(mjesto), 1, fIzlaz);
		i++;
	}
	fclose(fUlaz);
	fclose(fIzlaz);
	poc = clock();
	for (i=0; i<1000; i++) {
		if ((fUlaz = fopen("mjesta.dat", "rb")) == NULL)
			exit(-1);
		fseek(fUlaz, 30000L*sizeof(mjesto), SEEK_SET);
		fread(&mjesto, sizeof(mjesto), 1, fUlaz);
		fclose(fUlaz);
	}
	kraj = clock();
	printf("Za direktnu datoteku proslo je %8.3lf sekundi\n",
				(double)(kraj-poc)/CLOCKS_PER_SEC);
	poc = clock();
	for (i=0; i<1000; i++) {
		if ((fUlaz = fopen("mjesta.dat", "rb")) == NULL)
			exit(-1);
		while (fread(&mjesto, sizeof(mjesto), 1, fUlaz) == 1)
			if (mjesto.pbr == 40000) break;
		fclose(fUlaz);
	}
	kraj = clock();
	printf("Za slijednu datoteku proslo je %8.3lf sekundi\n",
				(double)(kraj-poc)/CLOCKS_PER_SEC);
}
```

Primjer izvođenja programa:

```
Za direktnu datoteku proslo je    0.219 sekundi
Za slijednu datoteku proslo je    7.922 sekundi
```

P: Koja datoteka je veća?

O: Direktna neformatirana jer ima praznih zapisa.

6. Potrebno je napisati program koji će spojiti dvije tekstualne datoteke (slijedne formatirane) tako da ispiše odgovarajuće retke datoteka (nisu duži od 80 znakova) odvojene prazninom:

```
prvi_redak_lijeve praznina prvi_redak_desne
drugi_redak_lijeve praznina drugi_redak_desne
```

…

Nazive datoteka zadati kao argumente komandne linije.

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

void fatal (char *msg) {
    printf("Fatal error:%s\n", msg);
    exit(-1);
}
int main (int brojA, char *poljeA[]) {
    FILE *fLijeva, *fDesna;
    char redLijeva[80+1] = {0}, redDesna[80+1] = {0};
    char *pLijeva, *pDesna;
    if (brojA != 3)
        fatal("Uporaba: paste lijeva desna");
    if( (fLijeva=fopen(poljeA[1], "r")) == NULL)
        fatal("Ne mogu otvoriti datoteku!");
    if( (fDesna=fopen(poljeA[2], "r")) == NULL){
        fclose(fLijeva);
        fatal("Ne mogu otvoriti datoteku!");
    }
    do {
        pLijeva = fgets(redLijeva, 81, fLijeva);
        pDesna = fgets(redDesna, 81, fDesna);
        if (pLijeva){
            //ukloni znak za novi red (ako postoji)
            if (redLijeva[strlen(redLijeva)-1]=='\n')
                redLijeva[strlen(redLijeva)-1] = '\0';
            printf("%s",redLijeva);
        }
        printf(" ");
        if (pDesna){
            //ukloni znak za novi red (ako postoji)
            if (redDesna[strlen(redDesna)-1]=='\n')
                redDesna[strlen(redDesna)-1] = '\0';
            printf("%s",redDesna);
        }
        printf("\n");
    } while (pLijeva || pDesna);
}
```

7. Potrebno je napisati program koji će zbrojit dva broja zadana kao argumente komandne linije.

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

void fatal (char *msg) {
     printf("Fatal error:%s\n", msg);
     exit(-1);
}

int main (int brojA, char *poljeA[]) {
     int prvi, drugi;

     if (brojA != 3)
          fatal("Uporaba: zbroji n1 n2");
     prvi = atoi(poljeA[1]);
     drugi = atoi(poljeA[2]);
     printf("%s + %s = %d\n", poljeA[1], poljeA[2],
                                          prvi+drugi);
     return 0;
}
```

Primjer izvođenja programa

```
C:\>zbroji 1 2
1 + 2 = 3

C:\>zbroji 1a 2a
1a + 2a = 3

C:\>zbroji 1s s2
1s + s2 = 1
```

Zašto je rezultat zbrajanja `1s` i `s2` jednak `1`?

14. Auditorne vježbe

Priprema za pismeni ispit

1. Prijenos varijabli u funkciju BY REFERENCE:

U svim zadacima gdje se traži da funkcija vrati više od jedne vrijednosti potrebno je koristiti prijenos varijabli u funkciju po referenci, odnosno funkciji treba poslati (adrese) varijable koje će funkcija "napuniti" sa vrijednostima i na taj način vratiti više od jedne vrijednosti.

1. *(30 bodova) (10. studeni 2000.)*

Mjeri se broj sekundi od početka nekog događaja. Napisati funkciju koja će za zadani broj sekundi (long) vratiti koliko je prošlo sati, minuta i sekundi od početka tog događaja.

```
void sek2SatMinSek(long brSekUkupno, int *brSat, int* brMin,
int *brSek){

   *brSat = brSekUkupno / (60 * 60);

   *brMin = (brSekUkupno - (*brSat * 60 * 60)) / 60;

   *brSek = brSekUkupno % 60;

}
```

Tipične pogreške:
Kod raznih sumiranja i prebrojavanja zaboravlja se postaviti varijablu na 0 (ili 1 kod produkta) ili se u glavnom programu postavlja varijabla na 0. To nije dobro, funkcija se ne bi smjela oslanjati na pozivajući program da joj inicijalizira varijable.
Evo primjera gdje takva funkcija ne radi dobro:

```
void main(){
int brSam=0;
...
prebrojiSamoglasnike("babe", &brSam);
printf("brSam = %d", brSam ); // ispisuje 2
prebrojiSamoglasnike("babe", &brSam);
printf("brSam = %d", brSam ); // ispisuje 4
```

Grube pogreške:	

`return brSat, brMin, brSek;`	`return brSat;`
	`return brMin;`
	`return brSek;`

2. Prijenos polja u funkciju.

To su zadaci u kojima je potrebno napisati funkcije za rad sa znakovnim nizovima ili brojevnim jednodimenzionalnim ili dvodimenzionalnim poljima (matrice).
Vrijedi sljedeća tablica:

TIP POLJA	POTREBNO JE PRENIJETI U FUNKCIJU
1D brojevno polje	1. pokazivač na početak polja 2. duljina polja
2D brojevno polje (matrica)	1. pokazivač na početak polja 2. aktualni broj redaka koji se koristi 3. aktualni broj stupaca koji se koristi 4. maksimalna dimenzija stupca(na koju je deklarirana matrica)
Znakovni niz	1. pokazivač na početak polja (jer znakovni nizovi završavaju sa znakom '\0' pa mu se može u funkciji lako odrediti dužina)

2. *(35 bodova) (27. rujna 2000.)*
Napisati funkciju koja će iz zadanog niza izbaciti sve razmake.

```
char *izbaci_razmake(char *niz){
     int idx_novi, idx_stari;
     idx_novi=idx_stari=0;

     while(niz[idx_stari] != '\0'){
          if (niz[idx_stari]==' '){
               idx_stari++;
          }else{
               niz[idx_novi]=niz[idx_stari];
               idx_stari++;
               idx_novi++;
          }
     }
     niz[idx_novi]=0; // ili '\0' ili niz[idx_stari]
     return niz;
}
```

Tipične pogreške:

- Kod matrica se ne prenosi MAXSTU ili se direktno koristi konstanta definirana sa pretprocesorskom naredbom. To nije dobro jer funkcija mora raditi za matrice raznih dimenzija, npr:
-

```
#define MAXRED 17
#define MAXSTU 10
void main(){
   int mat[MAXRED][MAXSTU];
   int mat2[5][55];
...
   transp(&mat[0][0], 2, 2, MAXSTU);
   transp(&mat2[0][0], 2, 2, 55);
...
```

- Funkcija koja mijenja/stvara znakovni niz mora ispravno postaviti oznaku kraja niza inače se znakovni niz ne može smatrati valjanim.

Grube pogreške:

- zaboravlja se predati duljina polja (1D brojevna polja)
- u funkcija duljina polja pokušava odrediti sa `sizeof(niz)`

3. (Pseudo)Slučajni brojevi

Funkcija srand() služi za inicijalizaciju generatora slučajnih brojeva i treba ju pozvati samo jednom prije generiranja slučajnih brojeva. Toj funkciji predajemo slučajnu vrijednost na osnovu koje se inicijalizira generator slučajnih brojeva i to je tipično broj sekundi od 1.1.1970 do trenutka izvršavanja programa:

```
srand ((unsigned) time(NULL));
```

Funkcija rand() generira slučajne cijele brojeve u intervalu [0, RAND_MAX]. Gotovo uvijek u zadacim je potrebno generirati broj u nekom drugom intervalu te je potrebno dobivene slučajne vrijednosti iz intervala [0, RAND_MAX] preslikati u interval [dg, gg]. To se može napraviti na jedan od ova dva načina:

```
1.) (int) ((float)rand()/(RAND_MAX+1)*(gg-dg+1) + dg);
2.) rand()%(gg -dg + 1) + dg;
```

---

1. *(30 bodova) (8. studenog 2002)*
Napisati funkciju koja zamijenjuje sadržaj dvaju slučajno odabranih redaka u zadanoj matrici proizvoljnih dimenzija.

```
void zamRedak(int *m, int brRed, int brStup, int maxStu){
   // treba generirati slučajan broj u intervalu [0, brRed-1]
   // pretpostavlja se da matrica ima više od jednog retka ili
   // se može dodati: if (brRed > 1){
   int r1, r2, j, pom;
   r1 = (int)((float)rand()/(RAND_MAX+1)*(brRed-1-0+1)+0);
   do{
      r2 = rand()%(brRed-1-0+1)+0;
   }while(r1 == r2);
   for(j = 0; j < brStup; j++){
      pom = m[ r1*maxStu + j];
      m[ r1*maxStu + j] = m[ r2*maxStu + j];
      m[ r2*maxStu + j] = pom;
   }
}
```

Tipične pogreške:

- `srand()` se poziva više puta ili u petlji
- kod izračunavanje preslikavanja zaboravlja se `(float)` cast

4. Formatirane datoteke

Najčešće se jedan zadatak na pismenom ispitu odnosi na (slijedne) formatirane datoteke. Datoteku treba otvoriti (sa pravilnom oznakom "r" – čitanje, "w" – pisanje,...) i nakon toga provjeriti da li je uspješno otvorena. Na kraju funkcije datoteku treba zatvoriti. Slijedi tablica koja govori koje funkcije se koriste za formatirane odnosno neformatirane datoteke:

FORMATIRANE		NEFORMATIRANE
	`fopen()` `fclose()`	

`fscanf()`
`fprintf()`
`fgets()`
`fputs()`
`fgetc()`
`...`
		`fread()`
		`fwrite()`

	`fseek()`	

Formatirane datoteke se najčešće čitaju sa funkcijom fscanf(), pri čemu je važno znati formate čitanja koji se predaju kao argumenti funkciji fscanf(). Evo nekih najčešće korištenih:

FORMAT	ZNAČENJE
`"%s"`	Učitava znakovni niz; Pažnja: kao i kod scanf učitavanje prestaje kad naiđe na razmak, oznaku novog reda('\n') i tabulator ('\t') !
`"%10[^\n]"`	Učitava znakove dok ne naiđe na znak '\n' ili dok ne učita 10 znakova(ako nema '\n' među prvih 10). Ovime je moguće učitati znakovni niz koji sadrži razmake (ili koristeći fgets()).
`"%*c"`	Preskače se jedan znak. Ovaj element se ne broji u broj uspješno učitanih vrijednosti koje fscanf() vraća.
`"%*[^#]"`	Preskaču se svi znakovi dok se ne naiđe na znak #.
`"%4d"`	Učitava se cijeli broj na 4 znamenke

---

2. *(40 bodova) (8. studeni 2002.)*
Datoteka sadrži zapise oblika:

```
12345678901234567890123456789012345678901234567890
ime                  prezime              godina_rođenja
```

Napisati program koji će, za zadano ime datoteke iz komandne linije, provjeriti da li je datoteka sortirana po prezimenu uzlazno. Ukoliko datoteka nije sortirana ispisati broj i sadržaj retka koji narušava sortiranost.

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

void main(int argc, char *argv[]) {
   FILE *fIn;
   char prezime[20+1], staro_prezime[20+1];
   int sort = 1, brRetka = 1;
   if (argc != 2){
      printf("Uporaba: %s ime_datoteke", argv[0]);
      exit(1);
   }
   fIn=fopen(argv[1], "r");
   if (fIn != NULL){
      while(fscanf(fIn, "%*20c%20[^\n]%*[^\n]%*c", prezime) == 1){
         if ( (brRetka>1) && strcmp(prezime, staro_prezime)<0){
            sort = 0;
            break;
         }
         brRetka++;
         strcpy(staro_prezime, prezime);
      }
      if (sort == 1){
         printf("Datoteka je sortirana");
      }else{
         printf("Datoteka NIJE sortirana, redak:%d", brRetka);
      }
   }else{
      printf("Ne mogu otvoriti datoteku:%s", argv[1]);
   }
}
```

Tipične pogreške:

- `staro_prezime = prezime;`
- zaboravlja se provjeriti broj ulaznih argumenata
- zaboravlje se da je prvi ulazni argument uvijek ime programa, npr. za ovakvo pokretanje programa iz komandne linije:

```
c:\temp\mojProg.exe ulaz1 ulaz2
```

vrijedi:

```
argc = 3
argv[0] = "c:\temp\mojProg.exe"
argv[1] = "ulaz1"
argv[2] = "ulaz2"
```

5. Neformatirane datoteke

U zadacima se najčešće pojavljuje direktna neformatirana datoteka (u zadatku je zadano kako se direktno pozicionirati na neki zapis, npr. matični broj odgovara rednom broju zapisa...) u kombinaciji sa nekom drugom datotekom. U direktnoj datoteci se tipično pogledavaju vrijednosti na osnovu ključa.

---

3. *(45 bodova) (8. studeni 2002.)*
Neformatirana datoteka "ispiti.dat" sadrži podatke o ispitima:

```
šifra studenta      long
šifra predmeta      long
šifra nastavnika    long
ocjena              int
```

Direktna neformatirana datoteka "predmeti.dat" sadrži podatke o predmetima:

```
šifra predmeta      long
naziv               char(100+1)
```

gdje šifra predmeta odgovara rednom broju zapisa u datoteci.
Napisati funkciju koja će vratiti prosječnu ocjenu svih ispita iz predmeta koji u nazivu sadržavaju zadani niz (npr. za zadani niz "Matematika" predmeti "Matematika I" i "Matematika II" sadržavaju zadani niz). Ukoliko nema niti jednog ispita koji odgovara zadanim kriterijima funkcija vraća 0. Ukoliko se neka od datoteka ne može otvoriti funkcija vraća -1. Napomena: među ocjenama se nalaze i negativne ocjene koje ne ulaze u prosjek ocjena.

U ovom zadatku ipak nije očigledno da li je bolje:

a) proći kroz sve zapis o ocjenama samo jednom i za svaku pozitivnu ocjenu direktno pogledati da li je to ocjena iz traženog predmeta
b) proći kroz zapise o predmetima slijedno i samo za tražene predmete slijedno proći datoteku s ispitima i sumirati ocjene

pa bi se oba rješenja u potpunosti priznala.
Prikazano je rješenje a).

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

typedef struct{
   long sifStud;
   long sifPred;
   long sifNast;
   int ocjena;
} strIspit;

typedef struct{
   long sifPred;
   char nazPred[100+1];
} strPredmet;


float prOcjena(char *nazPred){
  FILE *fIspit, *fPredmet;
  strIspit zIspit;
  strPredmet zPredmet;
  int suma=0, br = 0;
  fIspit = fopen("ispiti.dat", "rb");
  if (fIspit == NULL) return -1;
  fPredmet = fopen("predmeti.dat", "rb");
  if (fPredmet == NULL){
    fclose(fIspit);
    return -1;
  }
  while(fread(&zIspit, sizeof(zIspit), 1, fIspit) == 1){
    if (zIspit.ocjena > 1){
      fseek(fPredmet, (zIspit.sifPred-1L)*sizeof(zPredmet), SEEK_SET );
      fread(&zPredmet, sizeof(zPredmet), 1, fPredmet);
      if (strstr(zPredmet.nazPred, nazPred) != NULL){
        suma += zIspit.ocjena;
        br++;
      }
    }
  }
  if (br)
    return (float) suma/br;
  else
    return 0;

}
```

Tipične pogreške:

- kod otvaranje neformatirane datoteke zaboravlja se "b"
- direktnoj datoteci se prisupa slijedno kada to nema smisla, npr. zadatak:

---

4. *(40 bodova) (14. rujna 2000.)*
Direktna neformatirana datoteka "namirnice" sadrži podatke o namirnicama:

šifra namirnice	`long`
naziv namirnice	30+1 znakova
jedinična cijena	`float`

Šifra namirnice odgovara rednom broju zapisa u datoteci. Napisati funkciju:
`void kosarica(FILE *fo, long *sifre, float *kolicine, int n)`
koji će za zadanih `n šifri` namirnica i njihovih `n količina` na zaslon ispisati račun u sljedećem obliku:

```
01234567890123456789012345678901234567890123456789
--------------------------------------------------
Namirnica                      Kolicina  Cijena
--------------------------------------------------
Mlijeko                        001.000   006.730
Banane                         002.500   013.500
...
--------------------------------------------------
                               Ukupno:   XXX.XXX
```

Ovdje se gube bodovi ukoliko se datoteku "namirnice" pretražuje slijedno.