

Univerzitet u Beogradu  
Matematički fakultet

Milena, Jelena, Ana, Mirko, Anđelka, Nina

## Zbirka programa

Beograd, 2015.



# Predgovor

U okviru kursa *Programiranje 2* na Matematičkom fakultetu vežbaju se zadaci koji imaju za cilj da studente nauče rekurzivnom pristupu rešavanju problema, ispravan rad sa pokazivačima i dinamički alociranom memorijom, osnovne algoritme pretraživanja i sortiranja, kao i rad sa dinamičkim strukturama podataka, poput listi i stabala. Zadaci koji se nalaze u ovoj zbirci predstavljaju objedinjen skup zadataka sa vežbi i praktikuma ovog kursa, kao i primere zadataka sa kolokvijuma i ispita. Elektronska verzija zbirke, dostupna je u okviru strane kursa [www.programiranje2.matf.bg.ac.rs](http://www.programiranje2.matf.bg.ac.rs), a tu je dostupan i radni repozitorijum elektronskih verzija rešenja zadataka.

Autori velikog broja zadataka ove zbirke su ujedno i autori same zbirke, ali postoje i zadaci za koje se ne može tačno utvrditi ko je originalni autor jer su zadacima davali svoje doprinose različiti asistenti koji su držali vežbe iz ovog kursa u prethodnih desetak godina, pomenimo tu, pre svega, Milana Bankovića i doc dr Filipa Marića. Zbog toga smatramo da je naš osnovni doprinos što smo objedinili, precizno formulisali i rešili sve najvažnije zadatke koji su potrebni za uspešno savlađivanje koncepata koji se obrađuju u okviru kursa.

Zahvaljujemo se recenzentima na ..., kao i studentima koji su svojim aktivnim učešćem u nastavi pomogli i doprineli u obličavanju ovog materijala.

*Autori*

---

# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>3</b>
1.1	Podela koda po datotekama . . . . .	3
1.2	Algoritmi za rad sa bitovima . . . . .	6
1.3	Rekurzija . . . . .	11
1.3.1	Rekurzivne funkcije nad brojevima . . . . .	11
1.3.2	Rekurzivne funkcije za rad sa nizovima . . . . .	13
1.3.3	Rekurzivne funkcije - razni zadaci . . . . .	15
1.3.4	Rekurzivne funkcije za rad sa bitovima . . . . .	17
1.4	Rešenja . . . . .	18
<b>2</b>	<b>Pokazivači</b>	<b>61</b>
2.1	Pokazivačka aritmetika . . . . .	61
2.2	Višedimenzioni nizovi . . . . .	64
2.3	Dinamička alokacija memorije . . . . .	68
2.4	Pokazivači na funkcije . . . . .	71
2.5	Rešenja . . . . .	73
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>109</b>
3.1	Pretraživanje . . . . .	109
3.2	Sortiranje . . . . .	113
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	123
3.4	Rešenja . . . . .	128
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>199</b>
4.1	Liste . . . . .	199
4.2	Stabla . . . . .	212
4.3	Rešenja . . . . .	221
<b>5</b>	<b>Ispitni rokovi</b>	<b>255</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	255
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	256
5.3	Rešenja . . . . .	258
	<b>Literatura</b>	<b>268</b>



# Glava 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

**Zadatak 1.1** Napisati program za rad sa kompleksnim brojevima.

- (a) Definirati strukturu `KompleksanBroj` koja predstavlja kompleksan broj i sadrži realan i imaginarni deo kompleksnog broja.
- (b) Napisati funkciju `ucitaj_kompleksan_broj` koja učitava kompleksan broj sa standardnog ulaza.
- (c) Napisati funkciju `ispisi_kompleksan_broj` koja ispisuje kompleksan broj na standardni izlaz u formatu (npr. broj čiji je realan deo 2 a imaginarni  $-3$  ispisati kao  $(2 - 3i)$  na standardni izlaz).
- (d) Napisati funkciju `realan_deo` koja računa vrednosti realnog dela broja.
- (e) Napisati funkciju `imaginarni_deo` koja računa vrednosti imaginarnog dela broja.
- (f) Napisati funkciju `moduo` koja računa moduo kompleksnog broja.
- (g) Napisati funkciju `konjugovan` koja računa konjugovano-kompleksni broj svog argumenta.
- (h) Napisati funkciju `saberi` koja sabira dva kompleksna broja.
- (i) Napisati funkciju `oduzmi` koja oduzima dva kompleksna broja.
- (j) Napisati funkciju `mnozi` koja množi dva kompleksna broja.
- (k) Napisati funkciju `argument` koja računa argument kompleksnog broja.

Napisati program koji testira prethodno napisane funkcije za dva kompleksna broja  $z_1$  i  $z_2$  koja se unose sa standardnog ulaza i ispisuje:

- (a) realni, imaginarni deo i moduo kompleksnog broja  $z_1$ ,

## 1 Uvodni zadaci

---

- (b) konjugovano kompleksan broj i argument broja  $z_2$ ,
- (c) zbir, razliku i proizvod brojeva  $z_1$  i  $z_2$ .

### Test 1

```
Ulaz:  Unesite realan i imaginaran deo kompleksnog broja: 1 -3
       Unesite realan i imaginaran deo kompleksnog broja: -1 4
Izlaz: (1.00 -3.00 i )
       realan_deo: 1
       imaginaran_deo: -3.000000
       moduo 3.162278

       (-1.00 + 4.00 i )
       Njegov konjugovano kompleksan broj: (-1.00 -4.00 i )
       Argument kompleksnog broja 1.815775

(1.00 -3.00 i ) + (-1.00 + 4.00 i ) = ( 1.00 i )
(1.00 -3.00 i ) - (-1.00 + 4.00 i ) = (2.00 -7.00 i )
(1.00 -3.00 i ) * (-1.00 + 4.00 i ) = (11.00 + 7.00 i )
```

[Rešenje 1.1]

**Zadatak 1.2** Uraditi prethodni zadatak tako da su sve napisane funkcije za rad sa kompleksnim brojevima zajedno sa definicijom strukture `KompleksanBroj` izdvojene u posebnu biblioteku, dok test program koristi tu biblioteku da za kompleksan broj unet sa standardnog ulaza ispiše polarni oblik unetog broja.

### Test 1

```
Ulaz:  Unesite realan i imaginaran deo kompleksnog broja: -5 2
Izlaz: Polarni oblik kompleksnog broj je 5.39 * e^i * 2.76
```

[Rešenje 1.2]

**Zadatak 1.3** Napisati malu biblioteku za rad sa polinomima.

- (a) Definirati strukturu `Polinom` koja predstavlja polinom (stepena najviše 20). Struktura sadrži stepen i niz koeficijenata. (Diskutovati redosled navođenja koeficijenata u nizu).
- (b) Napisati funkciju koja ispisuje polinom na standardni izlaz u što lepšem obliku.
- (c) Napisati funkciju koja učitava polinom sa standardnog ulaza.
- (d) Napisati funkciju za izračunavanje vrednosti polinoma u datoj tački (diskutovati Hornerov algoritam i njegove prednosti).
- (e) Napisati funkciju koja sabira dva polinoma.
- (f) Napisati funkciju koja množi dva polinoma.



Sve vreme, paralelno sa razvojem funkcija, pisati i glavni program koji ih testira.

Nakon toga, izdvojiti funkcije u zasebnu datoteku `polinom.c`, a program u `test-polinom.c`. Prikazati probleme u kompilaciji, pošto ne postoji više definicija strukture kao ni prototipovi funkcija u `test-polinom.c`. Diskutovati da dupliranje ovoga u obe C datoteke dovodi do velikih problema prilikom održavanja programa. Uvesti `polinom.h` kao rešenje i uključiti ga u obe datoteke. Prikazati kreiranje objektna datoteke `polinom.o`.

```
gcc -o test-polinom polinom.c test-polinom.c
```

i

```
gcc -c polinom.c
```

```
gcc -c test-polinom.c
```

```
gcc -o test-polinom polinom.o test-polinom.o
```

### Upotreba programa 1

```
Unesite polinom (prvo stepen, pa zatim koeficijente od najveceg
stepena do nultog):
3 1 2 3 4
1.00x^3+2.00x^2+3.00x+4.00
Unesite tacku u kojoj racunate vrednost polinoma
5
Vrednost polinoma u tacki je 194.00
Unesite drugi polinom (prvo stepen, pa zatim koeficijente od najveceg
stepena do nultog):
2 1 0 1
Zbir polinoma 1.00x^3+2.00x^2+3.00x+4.00 i 1.00x^2+1.00 polinoma je
: 1.00x^3+3.00x^2+3.00x+5.00
Zbir polinoma 1.00x^3+2.00x^2+3.00x+4.00 i 1.00x^2+1.00 polinoma je
: 1.00x^5+2.00x^4+4.00x^3+6.00x^2+3.00x+4
Unosite izvod polinoma koji zelite:
2
2. izvod polinoma 1.00x^3+2.00x^2+3.00x+4.00 je : 6.00x+4.00
```

[Rešenje 1.3]

**Zadatak 1.4** Napraviti biblioteku za rad sa razlomcima.

- (a) Definirati strukturu za reprezentovanje razlomaka.
- (b) Napisati funkcije za učitavanje i ispis razlomaka.
- (c) Napisati funkcije koje vraćaju brojilac i imenilac.
- (d) Napisati funkciju koja vraća vrednost razlomka kao `double` vrednost.
- (e) Napisati funkciju koja izračunava recipročnu vrednost razlomka.
- (f) Napisati funkciju koja skraćuje dati razlomak.
- (g) Napisati funkcije koje sabiraju, oduzimaju, množe i dele dva razlomka.

Napisati program koji testira prethodne funkcije tako što se sa standardnog ulaza unose dva razlomka `r1` i `r2` i na standardni izlaz se ispisuju skraćeni vrednosti razlomaka koji predstavljaju zbir, razliku, proizvod i količnik razlomka `r1` i recipročne vrednosti razlomka `r2`.

*Test 1*

```
|| Ulaz:   Unesite imenilac i brojilac: 1 2
||         Unesite imenilac i brojilac: 3 1
|| Izlaz:   Zbir je 5/6
||         Razlika je 1/6
||         Proizvod je 1/6
||         Kolicnik je 3/2
```

## 1.2 Algoritmi za rad sa bitovima

**Zadatak 1.5** Napisati funkciju `print_bits` koja štampa bitove u binarnom zapisu celog broja `x`. Napisati program koja testira funkciju `print_bits` za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

*Test 1*

```
|| Ulaz:   0x7F
|| Izlaz:   0000 0000 0000 0000 0000 0000 0111 1111
```

*Test 2*

```
|| Ulaz:   0x80
|| Izlaz:   0000 0000 0000 0000 0000 0000 1000 0000
```

*Test 3*

```
|| Ulaz:   0x00FF00FF
|| Izlaz:   0000 0000 1111 1111 0000 0000 1111 1111
```

*Test 4*

```
|| Ulaz:   0xFFFFFFFF
|| Izlaz:   1111 1111 1111 1111 1111 1111 1111 1111
```

*Test*

```
|| Ulaz:   0xABCDE123
|| Izlaz:   1010 1011 1100 1101 1110 0001 0010 0011
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkciju koja broji bitove postavljene na 1 u zapisu broja `x`. Napisati program koji testira tu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

<p><i>Test 1</i></p> <pre>    Ulaz:  0x7F    Izlaz:      Broj bitova u      zapisu je 7         </pre>	<p><i>Test 2</i></p> <pre>    Ulaz:  0x80    Izlaz:      Broj bitova u      zapisu je 1         </pre>	<p><i>Test 3</i></p> <pre>    Ulaz:  0x00FF00FF    Izlaz:      Broj bitova u      zapisu je 16         </pre>
<p><i>Test 4</i></p> <pre>    Ulaz:  0xFFFFFFFF    Izlaz:      Broj bitova u      zapisu je 32         </pre>	<p><i>Test 4</i></p> <pre>    Ulaz:  0ABCDE123    Izlaz:      Broj bitova u      zapisu je 17         </pre>	

[Rešenje 1.6]

**Zadatak 1.7** Napisati funkciju **najveci** koja određuje najveći broj koji se može zapisati istim binarnim ciframa kao dati broj i funkciju **najmanji** koja određuje najmanji broj koji se može zapisati istim binarnim ciframa kao dati broj.

Napisati program koji testira prethodno napisane funkcije tako što prikazuje binarnu reprezentaciju brojeva koji se dobijaju nakon poziva funkcije **najveci**, odnosno **najmanji** za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

*Test 1*

```

|| Ulaz:  0x7F
|| Izlaz:
|| Najveci:
|| 1111 1110 0000 0000 0000 0000 0000 0000
|| Najmanji:
|| 0000 0000 0000 0000 0000 0000 0111 1110
        
```

*Test 2*

```

|| Ulaz:  0x80
|| Izlaz:
|| Najveci:
|| 1000 0000 0000 0000 0000 0000 0000 0000
|| Najmanji:
|| 0000 0000 0000 0000 0000 0000 0000 0001
        
```

*Test 3*

```

|| Ulaz:  0x00FF00FF
|| Izlaz:
|| Najveci:
|| 1111 1111 1111 1111 0000 0000 0000 0000
|| Najmanji:
|| 0000 0000 0000 0000 1111 1111 1111 1111
        
```

Test 4

```
Ulaz:  0xFFFFFFFF
Izlaz:
Najveci:
1111 1111 1111 1111 1111 1111 1111 1111
Najmanji:
1111 1111 1111 1111 1111 1111 1111 1111
```

Test 4

```
Ulaz:  0xABCDE123
Izlaz:
Najveci:
1111 1111 1111 1111 1000 0000 0000 0000
Najmanji:
0000 0000 0000 0001 1111 1111 1111 1111
```

[Rešenje 1.7]

**Zadatak 1.8** Napisati program za rad sa bitovima.

- (a) Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja, počevši od pozicije  $p$  postave na 0.
- (b) Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja, počevši od pozicije  $p$  postave na 1.
- (c) Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja, počevši od pozicije  $p$  i vraća ih kao bitove najmanje težine rezultata.
- (d) Napisati funkciju koja vraća broj koji se dobija upisivanjem poslednjih  $n$  bitova broja  $y$  u broj  $x$ , počevši od pozicije  $p$ .
- (e) Napisati funkciju koja vraća broj koji se dobija invertovanjem  $n$  bitova broja  $x$  počevši od pozicije  $p$ .
- (f) Napisati program koji testira prethodno napisane funkcije.

Program treba da testira prethodno napisane funkcije nad neoznačenim celim brojem koji se unosi sa standardnog ulaza. *Napomena: Pozicije se broje počev od pozicije najnižeg bita, pri čemu se broji od nule .*

### Test 1

[illegible]

[Rešenje 1.8]

**Zadatak 1.9** Rotiranje ulevo podrazumeva pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najviše težine pomera na poziciju najmanje težine. Analogno, rotiranje udesno podrazumeva pomeranje svih bitova za jednu poziciju udesno, s tim što se bit sa pozicije najmanje težine pomera na poziciju najviše težine.

- Napisati funkciju `rotate_left` koja određuje broj koji se dobija rotiranjem  $k$  puta u levo datog celog broja  $x$ .
- Napisati funkciju `rotate_right` koja određuje broj koji se dobija rotiranjem  $k$  puta u desno datog celog neoznačenog broja  $x$ .
- Napisati funkciju `rotate_right_signed` koja određuje broj koji se dobija rotiranjem  $k$  puta u desno datog celog broja  $x$ .

Napisati program koji testira prethodno napisane funkcije za broj **x** i broj **k** koji se sa standardnog ulaza unose u heksadekasnom formatu.

### Test 1

```

Ulaz:      B10011A7 5
Izlaz:
    x
    101100010000000000001000110100111
rotate_left(2969571751,      5)
    001000000000000100011010011110110
rotate_right(2969571751,      5)
    001111011000100000000000010001101
rotate_right_signed(2969571751,      5) =
    001111011000100000000000010001101

```

[Rešenje 1.9]

[Rešenje 1.10]

*Test 3*

[Rešenje 1.11]

### Test 3

[Rešenje 1.12]

rešavanju nije dozvoljeno koristiti pomoćni niz niti aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $%$ .

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Poziv: ./a.out 1 2</code>	<code>Poziv: ./a.out 1 2</code>	<code>Poziv: ./a.out 12 12</code>
<code>Ulaz: 11</code>	<code>Ulaz: 1024</code>	<code>Ulaz: 12345</code>
<code>Izlaz: 13</code>	<code>Izlaz: 1024</code>	<code>Izlaz: 12345</code>

**Zadatak 1.14** Napisati funkciju koja na osnovu neoznačenog broja  $x$  formira nisku  $s$  koja sadrži heksadekadni zapis broja  $x$ , koristeći algoritam za brzo prevođenje binarnog u heksadekadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksadekadnom cifrom). Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Ulaz: 11</code>	<code>Ulaz: 1024</code>	<code>Ulaz: 12345</code>
<code>Izlaz: 0000000B</code>	<code>Izlaz: 00000400</code>	<code>Izlaz: 00003039</code>

[Rešenje 1.14]

**Zadatak 1.15** ++ Napisati funkciju koja za dva data neoznačena broja  $x$  i  $y$  invertuje u podatku  $x$  one bitove koji se poklapaju sa odgovarajućim bitovima u broju  $y$ . Ostali bitovi ostaju nepromenjeni. Napisati program koji tu funkciju testira za brojeve koji se zadaju sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Ulaz: 123 10</code>	<code>Ulaz: 3251 0</code>	<code>Ulaz: 12541 1024</code>
<code>Izlaz: 4294967285</code>	<code>Izlaz: 4294967295</code>	<code>Izlaz: 4294966271</code>

**Zadatak 1.16** ++ Napisati funkciju koja računa koliko petica bi imao ceo neoznačen broj  $x$  u oktalnom zapisu. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Ulaz: 123</code>	<code>Ulaz: 3245</code>	<code>Ulaz: 100328</code>
<code>Izlaz: 0</code>	<code>Izlaz: 2</code>	<code>Izlaz: 1</code>

## 1.3 Rekurzija

### 1.3.1 Rekurzivne funkcije nad brojevima

**Zadatak 1.17** Napisati rekurzivnu funkciju koja izračunava  $x^k$ , za dati ceo broj  $x$  i prirodan broj  $k$ .

Napisati program koji testira napisanu funkciju za vrednosti koje se unose sa standardnog ulaza.

[Rešenje 1.17]

*Test 1*

```
|| Ulaz:  2 10
|| Izlaz: 1024
```

**Zadatak 1.18** Koristeći uzajamnu (posrednu) rekurziju napisati naredne dve funkcije:

- funkciju **paran** koja proverava da li je broj cifara nekog broja paran i vraća 1 ako jeste, a 0 inače;
- i funkciju **neparan** koja vraća 1 ukoliko je broj cifara nekog broja neparan, a 0 inače.

Napisati program koji testira napisanu funkciju tako što se za heksadekadnu vrednost koja se unosi sa standardnog ulaza ispisuje da li je paran ili neparan.

*Test 1*

```
|| Ulaz:  11
|| Izlaz: Uneti broj ima paran broj
||         cifara
```

*Test 2*

```
|| Ulaz:  123
|| Izlaz: Uneti broj ima neparan
||         broj cifara
```

[Rešenje 1.18]

**Zadatak 1.19** Napisati repno-rekurzivnu funkciju koja izračunava faktorijel broja  $n$ . Napisati program koji testira napisanu funkciju za poizvoljan broj  $n$  ( $n \leq 12$ ) unet sa standardnog ulaza.

*Test 1*

```
|| Ulaz:  5
|| Izlaz: 5! = 120
```

[Rešenje 1.19]

**Zadatak 1.20** Elementi funkcije  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = a * F(n - 1) + b * F(n - 2)$$

Napisati rekurzivnu funkciju koja računa  $n$ -ti element u nizu  $F$  ali tako da se problemi manje dimenzije rešavaju samo jedan put. Napisati program koji testira napisane funkcije za poizvoljan broj  $n$  ( $n \in \mathbb{N}$ ) unet sa standardnog ulaza.



*Test 1*

```
|| Ulaz:  2 3 5
|| Izlaz: 61
```

[Rešenje 1.20]

**Zadatak 1.21** Napisati rekurzivnu funkciju koja sabira dekadne cifre datog celog broja  $x$ . Napisati program koji testira ovu funkciju, za broj koji se unosi sa standardnog ulaza.

*Test 1*

```
|| Ulaz:  123
|| Izlaz:  6
```

*Test 2*

```
|| Ulaz:  23156
|| Izlaz:  17
```

*Test 3*

```
|| Ulaz:  1432
|| Izlaz:  10
```

*Test 4*

```
|| Ulaz:  1
|| Izlaz:  1
```

*Test 5*

```
|| Ulaz:  0
|| Izlaz:  0
```

[Rešenje 1.21]

### 1.3.2 Rekurzivne funkcije za rad sa nizovima

**Zadatak 1.22** Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva. Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojnog niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom.

*Test 1*

```
|| Ulaz:  5 1 2 3 4 5
|| Izlaz: Suma elemenata je 15
```

[Rešenje 1.22]

**Zadatak 1.23** Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju, za niz koji se unosi sa standardnog ulaza. Niz neće imati više od 256 elemenata, i njegovi elementi se unose sve do kraja ulaza.

*Test 1*

```
|| Ulaz:  3 2 1 4 21
|| Izlaz:
|| Suma elemenata je 21
```

*Test 2*

```
|| Ulaz:  2 -1 0 -5 -10
|| Izlaz:
|| Suma elemenata je 2
```

<i>Test 3</i>	<i>Test 4</i>
<pre>   Ulaz:  1 11 3 5 8 1    Izlaz:         Suma elemenata je 11</pre>	<pre>   Ulaz:  5    Izlaz:         Suma elemenata je 5</pre>

[Rešenje 1.23]

**Zadatak 1.24** Napisati rekurzivnu funkciju `skalarno` koja izračunava skalarni proizvod dva data vektora. Napisati program koji testira ovu funkciju, za nizove koji se unose sa standardnog ulaza. Nizovi neće imati više od 256 elemenata. Prvo se unosi dimenzija nizova, a zatim i sami njihovi elementi.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  3 1 2 3 1 2 3    Izlaz:  14</pre>	<pre>   Ulaz:  2 3 5 2 6    Izlaz:  36</pre>

*Test 3*

```
|| Ulaz:  0
|| Izlaz:  0
```

[Rešenje 1.24]

**Zadatak 1.25** Napisati rekurzivnu funkciju `br_pojave` koja računa broj pojavljivanja elementa  $x$  u nizu  $a$  dužine  $n$ . Napisati program koji testira ovu funkciju, za  $x$  i niz koji se unose sa standardnog ulaza. Niz neće imati više od 256 elemenata. Prvo se unosi  $x$ , a zatim elementi niza sve do kraja ulaza.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  4 1 2 3 4    Izlaz:  1</pre>	<pre>   Ulaz:  11 3 2 11 14 11 43 1    Izlaz:  2</pre>

*Test 3*

```
|| Ulaz:  1 3 21 5 6
|| Izlaz:  0
```

[Rešenje 1.25]

**Zadatak 1.26** Napisati rekurzivnu funkciju `tri_uzastopna_clana` kojom se proverava da li su tri zadata broja uzastopni članovi niza. Potom, napisati program koji je testira. Sa standardnog ulaza se unose najpre tri tražena broja, a zatim elementi niza, sve do kraja ulaza. Pretpostaviti da neće biti uneto više od 256 brojeva.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  1 2 3 4 1 2 3 4 5    Izlaz:  da</pre>	<pre>   Ulaz:  1 2 3 11 1 2 4 3 6    Izlaz:  ne</pre>

*Test 3*

```

|| Ulaz:      1 2 3 1 2
|| Izlaz:     ne

```

[Rešenje 1.26]

### 1.3.3 Rekurzivne funkcije - razni zadaci

**Zadatak 1.27** Napisati rekurzivnu funkciju palindrom koja ispituje da li je data niska palindrom. Napisati program koji testira ovu funkciju. Pretpostaviti da niska neće imati više od 31 karaktera, i da se unosi sa standardnog ulaza.

*Test 1*

```

|| Ulaz:      programiranje
|| Izlaz:     ne

```

*Test 2*

```

|| Ulaz:      anavolimilovana
|| Izlaz:     da

```

*Test 3*

```

|| Ulaz:      a
|| Izlaz:     da

```

*Test 4*

```

|| Ulaz:      aba
|| Izlaz:     da

```

*Test*

```

|| Ulaz:      aa
|| Izlaz:     da

```

[Rešenje 1.27]

\* **Zadatak 1.28** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za poizvoljan prirodan broj  $n$  ( $n \leq 50$ ) unet sa standardnog ulaza.

*Test 1*

```

|| Ulaz:      3
|| Izlaz:     1 2 3
||           1 3 2
||           2 1 3
||           2 3 1
||           3 1 2
||           3 2 1

```

[Rešenje 1.28]

\* **Zadatak 1.29** Paskalov trougao se dobija tako što mu je svako polje (izuzev jedinica po krajevima) zbir jednog polja levo i jednog polja iznad.

```

      1
    1 1
  1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

```

## 1 Uvodni zadaci

---

- (a) Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta  $\binom{n}{k}$ , tj. vrednost polja  $(n, k)$ , gde je  $n$  redni broj hipotenuze, a  $k$  redni broj elementa u tom redu (na toj hipotenuzi). Brojanje počinje od nule. Na primer vrednost polja  $(4, 2)$  je 6.
- (b) Napisati rekurzivnu funkciju koja izračunava  $d_n$  kao sumu elemenata  $n$ -te hipotenuze Paskalovog trougla.

Napisati program koji testira prethodno napisane funkcije tako što najpre iscrta Paskalov trougao

### Test 1

```
Ulaz:  5 3
Izlaz:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1

8
```

### Test 2

```
Ulaz:  6 5
Izlaz:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1

32
```

[Rešenje 1.29]

**Zadatak 1.30** ++ Napisati rekurzivnu funkciju koja prikazuje sve varijacije sa ponavljanjem dužine  $n$  skupa  $\{a, b\}$ , i program koji je testira, za  $n$  koje se unosi sa standardnog ulaza.

*Test 1*

```

|| Ulaz:      3
|| Izlaz:     a a a
||            a a b
||            a b a
||            a b b
||            b a a
||            b a b
||            b b a
||            b b b

```

**Zadatak 1.31** ++ *Hanojske kule*: Data su tri vertikalna štapa, na jednom se nalazi  $n$  diskova poluprečnika 1,2,3,... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, a preostali štap se koristi kao pomoćni štap prilikom premeštanja.

Napisati program koji za proizvoljnu vrednost  $n$ , koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

**Zadatak 1.32** ++ *Modifikacija Hanojskih kula*: Data su četiri vertikalna štapa, na jednom se nalazi  $n$  diskova poluprečnika 1,2,3,... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala tri štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, premestajući jedan po jedan disk, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, pri čemu se preostala dva štapa koriste kao pomoćni štapovi prilikom premeštanja.

Napisati program koji za proizvoljnu vrednost  $n$ , koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

### 1.3.4 Rekurzivne funkcije za rad sa bitovima

Milena: Naredne zadatke prebaciti da budu nakon rekurzije. Nina: Da li je u redu da budu ovde?

**Zadatak 1.33** Napisati rekurzivnu funkciju vraća broj bitova koji su postavljeni na 1, u binarnoj reprezentaciji njenog celobrojnog argumenta. Napisati program koji testira napisanu funkciju za brojeve koji se učitavaju sa standardnog ulaza zadati u heksadekadnom formatu.

*Test 1*

```

|| Ulaz:  0x7F
|| Izlaz: 7

```

*Test 2*

```

|| Ulaz:  0x80
|| Izlaz: 1

```

*Test 3*

```

|| Ulaz:  0x00FF00FF
|| Izlaz: 16

```

*Test 4*

```

|| Ulaz:  0xFFFFFFFF
|| Izlaz: 32

```

[Rešenje 1.33]

**Zadatak 1.34 ++** Napisati rekurzivnu funkciju koja štampa bitovsku reprezentaciju neoznačenog celog broja, i program koji je testira za ulaz koji se zadaje sa standardnog ulaza.

### Test 1

```
Ulaz:      10
Izlaz:     000000000000000000000000000000001010
```

**Zadatak 1.35** Napisati rekurzivnu funkciju za određivanje najveće cifre u oktalnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. Uputstvo: binarne cifre grupisati u podgrupe od po tri cifre, počev od bitova najmanje težine.

### Test 1

```
Ulaz: 5
Izlaz: 5
```

### Test 2

```
Ulaz: 125
Izlaz: 7
```

### Test 3

```
Ulaz: 8
Izlaz: 1
```

*Test 4*

```
Ulaz: 10
Izlaz: 2
```

[Rešenje 1.35]

**Zadatak 1.36** Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. Uputstvo: binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.

*Test 1*

```
Ulaz: 5
Izlaz: 5
```

### Test 2

```
Ulaz: 16
Izlaz: 1
```

### Test 3

```
Ulaz: 18
Izlaz: 2
```

### Test 4

```
Ulaz: 165
Izlaz: 10
```

[Rešenje 1.36]

## 1.4 Rešenja

### Rešenje 1.1

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

```

4  /* Struktura kojom predstavljamo kompleksan broj, cuvajuci njegov
   realan i imaginaran deo */
5  typedef struct {
6      float real;
7      float imag;
8  } KompleksanBroj;

10 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
   kompleksnog broja i smesta ih u strukturu cija adresa je
   argument funkcije */
11 void ucitaj_kompleksan_broj(KompleksanBroj* z) {
12     printf("Unesite realan i imaginaran deo kompleksnog broja: ");
13     scanf("%f", &z->real);
14     scanf("%f", &z->imag);
15 }

16 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
   se salje kao argument u obliku (x + y i)
17 Ovoj funkciji se kompleksan broj prenosi po vrednosti, jer za
   ispis nam nije neophodno da imamo adresu
   */
20 void ispisi_kompleksan_broj(KompleksanBroj z) {
21     printf("(");
22     if( z.real != 0 ) {
23         printf("%.2f",z.real);
24         if(z.imag > 0 )
25             printf(" +");
26     }
27     if(z.imag !=0 )
28         printf(" %.2f i ",z.imag);
29     if(z.imag ==0 && z.real ==0 )
30         printf("0 ");
31     printf(")");
32 }

36 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
38 float realan_deo(KompleksanBroj z) {
39     return z.real;
40 }

42 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
44 float imaginaran_deo(KompleksanBroj z) {
45     return z.imag;
46 }

48 /* Funkcija vraca vrednost modula kompleksnog broja koji joj se
   salje kao argument */
50 float moduo(KompleksanBroj z) {
51     return sqrt( z.real* z.real + z.imag* z.imag);
52 }

54 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
   odgovara kompleksnom broju poslatom kao argument */

```

## 1 Uvodni zadaci

---

```
KompleksanBroj konjugovan(KompleksanBroj z) {
54     KompleksanBroj z1 = z;

56     z1.imag *= -1;

58     return z1;
}

60 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
    argumenata funkcije */
62 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z = z1;

64     z.real += z2.real;
66     z.imag += z2.imag;

68     return z;
}

70 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
    argumenata funkcije */
72 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z = z1;

74     z.real -= z2.real;
76     z.imag -= z2.imag;

78     return z;
}

80 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
    argumenata funkcije */
82 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z;

84     z.real = z1.real * z2.real - z1.imag * z2.imag;
86     z.imag = z1.real * z2.imag + z1.imag * z2.real;

88     return z;
}

90 /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
    kao argument */
92 float argument(KompleksanBroj z) {
    return atan2(z.imag, z.real);
94 }

96 /* U main() funckiji testiramo sve funckije koje smo definisali */
98 int main() {
    /* deklariseimo promenljive tipa KompleksanBroj */
100     KompleksanBroj z1, z2;

102     /* Ucitavamo prvi kompleksan broj */
    ucitaj_kompleksan_broj(&z1);
104 }
```



```

106     /* Ucitavamo i drugi kompleksan broj */
    ucitaj_kompleksan_broj(&z2);

108     /* Ispisujemo prvi kompleksan broj, a zatim i njegov realan i
    imaginaran deo, kao i moduo kompleksnog broja z1 */
    ispisi_kompleksan_broj(z1);
110     printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
    realan_deo(z1), imaginaran_deo(z1), moduo(z1));
    printf("\n");

112     /* Ispisuje drugi kompleksan broj, a zatim i racunamo i
    ispisujemo konjugovano kompleksan broj od z2 */
114     ispisi_kompleksan_broj(z2);
    printf("\nNjegov konjugovano kompleksan broj: ");
116     ispisi_kompleksan_broj( konjugovan(z2) );

118     /* Testiramo funkciju koja racuna argument kompleksnih brojeva
    */
    printf("\nArgument kompleksnog broja %.f\n", argument(z2) );
120     printf("\n");

122     /* Testiramo sabiranje kompleksnih brojeva */
    printf("\n");
124     ispisi_kompleksan_broj(z1);
    printf(" + ");
126     ispisi_kompleksan_broj(z2);
    printf(" = ");
128     ispisi_kompleksan_broj(saberi(z1, z2));
    printf("\n");

130     /* Testiramo oduzimanje kompleksnih brojeva */
132     printf("\n");
    ispisi_kompleksan_broj(z1);
134     printf(" - ");
    ispisi_kompleksan_broj(z2);
136     printf(" = ");
    ispisi_kompleksan_broj(oduzmi(z1, z2));
138     printf("\n");

140     /* Testiramo mnozenje kompleksnih brojeva */
    printf("\n");
142     ispisi_kompleksan_broj(z1);
    printf(" * ");
144     ispisi_kompleksan_broj(z2);
    printf(" = ");
146     ispisi_kompleksan_broj(mnozi(z1, z2));

148     /* program se zavrшава uspesno, tj, bez greske*/
    return 0;
150 }

```

## Rešenje 1.2

```

/* Uključujemo zaglavlje neophodno za rad sa kompleksnim brojevima
2 * Ovde je to neophodno jer nam je neophodno da bude poznata

```

```
    definicija tipa KompleksanBroj
    * i da budu ukljucena zaglavlja standardne biblioteke, neophodna za
      definicije, a
4   * njih smo vec naveli u complex.h
    */
6   #include "complex.h"

8   /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
      kompleksnog broja i smesta ih u strukturu cija adresa je
      argument funkcije */
    void ucitaj_kompleksan_broj(KompleksanBroj* z) {
10      printf("Unesite realan i imaginaran deo kompleksnog broja: ");
      scanf("%f", &z->real);
12      scanf("%f", &z->imag);
    }

14   /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
      se salje kao argument u obliku (x + i y)
16   Ovoj funkciji se kompleksan broj prenosi po vrednosti, jer za
      ispis nam nije neophodno da imamo adresu
      */
18   void ispisi_kompleksan_broj(KompleksanBroj z) {
      printf("(");
20      if( z.real != 0 ) {
          printf("%.2f",z.real);
22          if(z.imag > 0 )
              printf(" +");
24      }
26      if(z.imag !=0 )
          printf(" %.2f i ",z.imag);
28
      if(z.imag ==0 && z.real ==0 )
30          printf("0 ");

32      printf(")");
    }

34   /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
36   float realan_deo(KompleksanBroj z) {
      return z.real;
38   }

40   /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
42   float imaginaran_deo(KompleksanBroj z) {
      return z.imag;
    }

44   /* Funkcija vraca vrednost modula kompleksnog broja koji joj se
      salje kao argument */
46   float moduo(KompleksanBroj z) {
      return sqrt( z.real* z.real + z.imag* z.imag);
48   }

50   /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
      odgovara kompleksnom broju poslatom kao argument */
```

```

KompleksanBroj konjugovan(KompleksanBroj z) {
52     KompleksanBroj z1 = z;

54     z1.imag *= -1;

56     return z1;
}

58
/* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
   argumenata funkcije */
60 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z = z1;

62     z.real += z2.real;
64     z.imag += z2.imag;

66     return z;
}

68
/* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
   argumenata funkcije */
70 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z = z1;

72     z.real -= z2.real;
74     z.imag -= z2.imag;

76     return z;
}

78
/* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
   argumenata funkcije */
80 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z;

82     z.real = z1.real * z2.real - z1.imag * z2.imag;
84     z.imag = z1.real * z2.imag + z1.imag * z2.real;

86     return z;
}

88
/* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
   kao argument */
90 float argument(KompleksanBroj z) {
    return atan2(z.imag, z.real);
92 }

```

```

/*
2  * Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
   deklaracije funkcija za rad sa kompleksnim brojevima.
   * Zaglavlje nikada ne treba da sadrzi definicije funkcija.
4  * Bilo koji program koji bi htelo da koristi ove brojeve i funkcije
   iz ove biblioteke, neophodno je da ukljuci ovo zaglavlje
   */

6
/* Ovim preprocesorskim direktivama zakljucavamo zaglavlje i time

```

## 1 Uvodni zadaci

---

```

    onemogucujemo
8  * da se sadrzaj zaglavlja vise puta ukljuci, ukoliko se u kodu
    vise puta ukljuci isto zaglavlje
    *
10 * Niska posle kljucne reci ifndef je proizvoljna ali treba da se
    ponovi u narednoj preprocesorskoj define direktivi
    */
12 #ifndef _COMPLEX_H
13 #define _COMPLEX_H
14
15 /* Zaglavlja standardne biblioteke koje sadrze deklaracije funkcija
    koje se koriste u definicijama funkcija koje smo naveli u
    complex.c */
16 #include <stdio.h>
17 #include <math.h>
18
19 /* struktura kojom predstavljamo kompleksan broj, cuvajuci njegov
    realan i imaginaran deo */
20
21 typedef struct {
22     float real;
23     float imag;
24 } KompleksanBroj;
25
26 /* Deklaracije funkcija za rad sa kompleksnim brojevima.
    * Sve one su definisane u complex.c */
27 void ucitaj_kompleksan_broj(KompleksanBroj* z) ;
28
29 void ispisi_kompleksan_broj(KompleksanBroj z) ;
30
31 float realan_deo(KompleksanBroj z) ;
32
33 float imaginaran_deo(KompleksanBroj z);
34
35 float moduo(KompleksanBroj z);
36
37 KompleksanBroj konjugovan(KompleksanBroj z) ;
38
39 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) ;
40
41 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) ;
42
43 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) ;
44
45 float argument(KompleksanBroj z) ;
46
47 /* Kraj zakljucanog dela */
48 #endif

```

```

1 /*
2  * Koristimo korektno definisanu biblioteku kompleksnih brojeva.
3  * U zaglavlju complex.h nalazi se definicija kompleksnog broja i
    popis deklaracija podrzanih funkcija
4  * a u complex.c se nalaze njihove definicije.
5  *
6  * Ovde pisemo i main() funkciju drugaciju od prethodnog zadatka.

```

```

7  *
  * I dalje kompilacija programa se najjednostavnije postize naredbom
9  * gcc -Wall -lm -o izvrsni complex.c main.c

11 Kompilaciju možemo uraditi i na sledeci nacin:
gcc -Wall -c -o complex.o complex.c
13 gcc -Wall -c -o main.o main.c
gcc -lm -o complex complex.o main.o

15
Objasnjenje naredbi
17 gcc -Wall -c -o complex.o complex.c
Poziva prevodilac za kod compex.c sa opcijama:

19
-Wall (Stampaj upozorenja prevodioca),
21 -lm (za linkovanje sa math.h bibliotekom),
-o (fajl koji prevodilac generise imenuj sa complex.o)
23 -c (ne vrši prevodjenje do izvršnog programa, već samo
do objektnog koda).
25 Rezultat ovoga je objektni fajl complex.o, koji sadrži
program na mašinskom jeziku. Još uvek nije izvršni
27 program, jer nije uradjeno uvezivanje (linkovanje)
biblioteka koje su u njemu korišćene, i ostalih
29 objektnih fajlova koji se koriste sa njim.
Druge naredbe radi analogno za main.c.

31
Ova dva objektna fajla treba ulinkovati medjusobno, i sa
33 objektnim kodom standardne biblioteke. To se radi naredbom.
Prevodilac gcc prepoznaje da su njegovi
35 argumenti objektni fajlovi i da ne treba da ih prevodi, već
samo da ih ulinkuje ispravno.
37 gcc -lm -o complex complex.o main.o
*/

39

41 #include <stdio.h>
/* Uključujemo zaglavlje neophodno za rad sa kompleksnim brojevima
*/
43 #include "complex.h"

45 /* U main funkciji za uneti kompleksan broj ispisujemo njegov
polarni oblik */
int main() {
47     KompleksanBroj z;

49     /* Učitavamo kompleksan broj */
ucitaj_kompleksan_broj(&z);

51
ispisi_kompleksan_broj(z);

53
printf("\n");

55
printf("Polarni oblik kompleksnog broj je %.2f * e^i * %.2f\n",
moduo(z), argument(z));

57
return 0;
59 }

```

### Rešenje 1.3

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "polinom.h"
4
5
6 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku
7    Polinom prenosimo po adresi, da bi uštedeli kopiranje cele
8    strukture,
9    vec samo prenosimo adresu na kojoj se nalazi polinom kog
10   ispisujemo */
11 void ispisi(const Polinom * p)
12 {
13     int i;
14     for (i = p->stepen; i >= 0; i--) {
15         if (p->koef[i]) {
16             if (p->koef[i] >= 0)
17                 putchar('+');
18             if (i > 1)
19                 printf("%.2fx^%d", p->koef[i], i);
20             else if (i == 1)
21                 printf("%.2fx", p->koef[i]);
22             else
23                 printf("%.2f", p->koef[i]);
24         }
25     }
26     putchar('\n');
27 }
28
29 /* Funkcija koja ucitava polinom sa tastature */
30 Polinom ucitaj()
31 {
32     int i;
33     Polinom p;
34     /* Ucitavamo stepen polinoma */
35     scanf("%d", &p.stepen);
36     /* Ponavljamo ucitavanje stepena sve dok ne unesemo stepen iz
37        dozvoljenog opsega */
38     while (p.stepen > MAX_STEPEN || p.stepen < 0) {
39         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
40         scanf("%d", &p.stepen);
41     }
42     /* Unosimo koeficijente polinoma */
43     for (i = p.stepen; i >= 0; i--)
44         scanf("%lf", &p.koef[i]);
45     return p;
46 }
47
48 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
49    algoritmom */
50 /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x + 2)x + 3)x + 2)x + 1$  */
51 double izracunaj(const Polinom * p, double x)
```

```

{
49     double rezultat = 0;
    int i = p->stepen;
51     for (; i >= 0; i--)
        rezultat = rezultat * x + p->koef[i];
53     return rezultat;
}

55
/* Funkcija koja sabira dva polinoma */
57 Polinom saberi(const Polinom * p, const Polinom * q)
{
    Polinom rez;
    int i;

    rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;

    for (i = 0; i <= rez.stepen; i++)
65 rez.koef[i] =
    (i > p->stepen ? 0 : p->koef[i]) + (i >
67     q->stepen ? 0 : q->
        koef[i]);
69     return rez;
71 }

73 /* Funkcija mnozi dva polinoma p i q */
Polinom pomnozi(const Polinom * p, const Polinom * q)
75 {
    int i, j;
    Polinom r;

    r.stepen = p->stepen + q->stepen;
    if (r.stepen > MAX_STEPEN) {
81 fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
        exit(EXIT_FAILURE);
83     }

    for (i = 0; i <= r.stepen; i++)
85 r.koef[i] = 0;

    for (i = 0; i <= p->stepen; i++)
87     for (j = 0; j <= q->stepen; j++)
89         r.koef[i + j] += p->koef[i] * q->koef[j];

    return r;
93 }

95 /* Funkcija racuna izvod polinoma p */
Polinom izvod(const Polinom * p)
97 {
    int i;
    Polinom r;

    if (p->stepen > 0) {
101     r.stepen = p->stepen - 1;
103

```

## 1 Uvodni zadaci

```
105     for (i = 0; i <= r.stepen; i++)
        r.koef[i] = (i + 1) * p->koef[i + 1];
107     } else
        r.koef[0] = r.stepen = 0;

109     return r;
}

111
/* Funkcija racuna n-ti izvod polinoma p */
113 Polinom nIzvod(const Polinom * p, int n)
{
115     int i;
    Polinom r;

117     if (n < 0) {
119         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
        exit(EXIT_FAILURE);
121     }

123     if (n == 0)
125         return *p;

    r = izvod(p);
127     for (i = 1; i < n; i++)
        r = izvod(&r);

129     return r;
131 }

1
/* Ovim pretrocesorskim direktivama zakljucavamo zaglavlje i time
   onemogucujemo
3   da se sadrzaj zaglavlja vise puta ukljuci
   */
5 #ifndef _POLINOM_H
   #define _POLINOM_H
7
   #include <stdio.h>
9   #include <stdlib.h>

11  /* Maksimalni stepen polinoma */
   #define MAX_STEPEN 20
13

15  /* Polinome predstavljamo strukturom */
   typedef struct {
17     /* u kojoj imamo koeficijente; koef[i] je koeficijent uz clan x^
        i */
        double koef[MAX_STEPEN + 1];
19     /* i stepen polinoma */
        int stepen;
21 } Polinom;

23 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku
   Polinom prenosimo po adresi, da bi uštedeli kopiranje cele
   strukture,
```



```

25     vec samo prenosimo adresu na kojoj se nalazi polinom kog
        ispisujemo */
void ispisi(const Polinom * p);
27
/* Funkcija koja ucitava polinom sa tastature */
29 Polinom ucitaj();

31 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
        algoritmom */
/*  $x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 2)*x + 1$  */
33 double izracunaj(const Polinom * p, double x);

35 /* Funkcija koja sabira dva polinoma */
Polinom saberi(const Polinom * p, const Polinom * q);
37
/* Funkcija mnozi dva polinoma p i q */
39 Polinom pomnozi(const Polinom * p, const Polinom * q);

41 /* Funkcija racuna izvod polinoma p */
Polinom izvod(const Polinom * p);
43
/* Funkcija racuna n-ti izvod polinoma p */
45 Polinom nIzvod(const Polinom * p, int n);
#endif

```

---

```

#include <stdio.h>
2 #include "polinom.h"

4 /*
Kako se za prevodjenje naSeg programa sada koriste 3
6 komande, da bi se ovaj proces kompilacije
automatizovao postoji make alat.
8 Potrebno je da u datoteci koja se zove makefile
popisemo postupak kojim se od izvornih datoteka
10 dobija Zeljeni izvrSni program, a make alat Ce za nas
izvrSi naznaCene komande.
12 Detaljnije uputstvo o make alatu, gcc prevodiocu i joS
nekim korisnim alatima moZete nadji u skripti „GNU”
14 alati Aleksandra SamardZiCa.
http://poincare.matf.bg.ac.rs/~asamardzic/gnu.pdf
16 */

18 int main(int argc, char **argv)
{
20     Polinom p, q, r;
    double x;
22     int n;

24     /* Unos polinoma */
    printf
26     ("Unesite polinom (prvo stepen, pa zatim koeficijente od najveceg
stepena do nultog):\n");
    p = ucitaj();

28     /* Ispis polinoma */
30     ispisi(&p);

```

```
32  /* Unos tacke u kojoj se racuna vrednost polinoma */
printf("\nUnesite tacku u kojoj racunate vrednost polinoma\n");
34  scanf("%lf", &x);

36  /* Ispisujemo vrednost polinoma u toj tacki */
printf("Vrednost polinoma u tacki je %f\n", izracunaj(&p, x));
38

40  /* Unesimo drugi polinom */
printf
42  ("Unesite drugi polinom (prvo stepen, pa zatim koeficijente od
najveceg stepena do nultog):\n");
q = ucitaj();

44  /* Sabiramo polinome i ispisujemo zbir ta dva polinoma */
r = saberi(&p, &q);
46  printf("Zbir polinoma ");
ispisi(&p);
48  printf("i polinoma ");
ispisi(&q);
50  printf("je : ");
ispisi(&r);
52

54  /* Mnozimo polinome i ispisujemo proizvod ta dva polinoma */
r = pomnozi(&p, &q);
printf("\nProzvod polinoma ");
56  ispisi(&p);
printf("i polinoma ");
58  ispisi(&q);
printf("je : ");
60  ispisi(&r);

62  /* Izvod polinoma */
printf("\nUnosite izvod polinoma koji zelite: ");
64  scanf("%d", &n);
r = nIzvod(&p, n);
66  printf("%d. izvod polinoma ", n);
ispisi(&p);
68  printf("je : ");
ispisi(&r);
70

72  /* Uspesno završavamo program */
return 0;
}
```

### Rešenje 1.5

```
/*
2  Napisati:
- funkciju print_bits koja Stampa bitove u binarnom zapisu celog
broja x.
4  - program koji testira print_bits
*/
6
```

```

8  #include <stdio.h>

10 /* funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
void print_bits( int x) {
12     unsigned velicina = sizeof(int)*8;    /* broj bitova celog broja
        */
        unsigned maska;    /* maska koju cemo koristiti za "ocitavanje"
        bitova */

14     /* Bitove u zapisu broja treba da ispisujemo sa leva na desno,
        tj. od bita najvece tezine ka
16         * bitu najmanje tezine. Iz tog razloga, za pocetnu vrednost
        maske uzimamo vrednost
        * ciji binarni zapis je takav da je bit najvece tezine 1, a svi
        ostali nule.
18         * Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u
        desno, kako bismo ocitali
        * naredni bit, gledano s leva na desno. Odgovarajuci karakter,
        ('0' ili '1'), ispisuje se na ekranu.
20         *
        * Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
        postavljen na 1,
22         * neophodno je da maska bude neoznacena ceo broj i da se
        siftovanjem u desno ova 1
        * ne bi smatrala znakom i prepisivala, vec da bi nam se svakim
        siftovanjem sa levog kraja
24         * binarnog zapisa pojavljivale 0. */

26     for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
        putchar( x & maska ? '1' : '0' );

28     putchar('\n');
30 }

32
34 int main() {
    int broj;
    scanf("%x", &broj);
    print_bits(broj);

36
38     return 0;
}

```

### Rešenje 1.6

```

1  #include <stdio.h>

3  /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
void print_bits( int x) {
5     unsigned velicina = sizeof(int)*8;    /* broj bitova celog broja
        */
        unsigned maska;    /* maska koju cemo koristiti za "ocitavanje"
        bitova */

```

```
7      /* Bitove u zapisu broja treba da ispisujemo sa leva na desno,
      tj. od bita najveće težine ka
9      * bitu najmanje težine. Iz tog razloga, za početnu vrednost
      maske uzimamo vrednost
      * čiji binarni zapis je takav da je bit najveće težine 1, a svi
      ostali nule.
11     * Nakon toga, u svakoj iteraciji ćemo tu jedinicu pomerati u
      desno, kako bismo očitali
      * naredni bit, gledano s leva na desno. Odgovarajući karakter,
      ('0' ili '1'), ispisuje se na ekranu.
13     *
      * Zbog siftovanja maske u desno koja na početku ima najviši bit
      postavljen na 1,
15     * neophodno je da maska bude neoznačen ceo broj i da se
      siftovanjem u desno ova 1
      * ne bi smatrala znakom i prepisivala, već da bi nam se svakim
      siftovanjem sa levog kraja
17     * binarnog zapisa pojavljivale 0. */

19     for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
        putchar( x & maska ? '1' : '0' );
21
23     putchar('\n');
}

25 /* Funkcija vraća broj jedinica u binarnoj reprezentaciji broja x
    pomeranjem broja maske */
int count_bits(int x) {
27     int br=0;
    unsigned maska = 1;
29
    for( ; maska!=0; maska <<=1 )
31         if(x&maska)
            br++;
33
    return br;
35 }

37 /* Funkcija vraća broj jedinica u binarnoj reprezentaciji broja x
    pomeranjem broja x*/
int count_bits1(int x) {
39     int br=0;
    unsigned wl = sizeof(int)*8 -1;
41     unsigned maska= 1 << wl;

43     /* Kako je argument funkcije označen ceo broj x ne možemo da
      siftujemo x u desno.
      naredba x>>=1 vrsila bi aritmetički sift u desno, tj. bitove
45     sa desne strane bi bili popunjavani bitom znaka.
      Npr. -3 bit znaka je 1. U tom slučaju nikad neće biti
47     ispunjen uslov x!=0 i program će biti zarobljen u
      beskonacnoj petlji.
49     */

51     /* Formiramo masku 100000...0000000,koja služi za
```

```

53         ocitavanje bita najveće težine. U svakoj iteraciji
54         x se pomera u levo za 1 mesto, i ocitavamo sledeći
55         bit. Petlja se završava kada više nema jedinica tj.
56         kada x postane nula. */
57     for( ; x!=0 ; x<<=1 )
58         x & maska ? br++ : 1;
59
60     return br;
61 }
62
63 int main() {
64     print_bits(0xABCDE123);
65     printf("Broj bitova u zapisu je %d.\n", count_bits(0xABCDE123));
66     /*
67     printf("Broj bitova u zapisu je %d.\n", count_bits1(0xABCDE123))
68     ;
69     */
70     return 0;
71 }

```

### Rešenje 1.7

```

1  #include <stdio.h>
2
3  /* Funkcija vraća najveći neoznačeni broj sastavljen iz istih
4     bitova kao i x */
5  unsigned najveći(unsigned x) {
6     unsigned velicina = sizeof(unsigned)*8;
7
8     /* Formiramo masku 100000...00000000 */
9     unsigned maska = 1 << (velicina-1);
10
11     /* Inicijalizujemo rezultat na 0 */
12     unsigned rezultat = 0;
13
14     /* Dokle god postoje jedinice u binarnoj reprezentaciji broja x
15        (tj. dokle god je x različit od nule) pomeramo ga ulevo. */
16     for( ; x!=0; x<<=1 ) {
17         /* Za svaku jedinicu, potiskujemo jednu
18            * novu jedinicu sa leva u rezultat */
19         if( x & maska) {
20             rezultat >>= 1;
21             rezultat |= maska;
22         }
23     }
24
25     return rezultat;
26 }
27
28 /* Funkcija vraća najmanji neoznačen broj
29    sa istim binarnim ciframa kao i x */
30 unsigned najmanji(unsigned x) {
31     /* Inicijalizujemo rezultat na 0 */

```

```
32     unsigned rezultat =0;

34     /* Dokle god imamo jedinice u broju x, pomeramo ga udesno. */
35     for( ;x!=0 ; x>=1){
36         /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
37         desna u rezultat */
38         if( x& 1) {
39             rezultat <<=1;
40             rezultat |= 1;
41         }
42     }

43     return rezultat;
44 }

45 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
46 celog broja u memoriji */
47 void print_bits( int x) {
48     unsigned velicina = sizeof(int)*8;    /* Broj bitova celog broja
49     */
50     unsigned maska;    /* Maska koju cemo koristiti za "ocitavanje"
51     bitova */

52     /* Bitove u zapisu broja treba da ispisujemo sa leva na desno,
53     tj od bita najvece tezine ka
54     * bitu najmanje tezine. Iz tog razloga, za pocetnu vrednost
55     maske uzimamo vrednost
56     * ciji binarni zapis je takav da je bit najvece tezine 1, a svi
57     ostali nule.
58     * Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u
59     desno, kako bismo ocitali
60     * naredni bit, gledano s leva na desno. Odgovarajuci karakter,
61     ('0' ili '1'), ispisuje se na ekranu.
62     *
63     * Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
64     postavljen na 1,
65     * neophodno je da maska bude neoznacena ceo broj i da se
66     siftovanjem u desno ova 1
67     * ne bi smatrala znakom i prepisivala, vec da bi nam se svakim
68     siftovanjem sa levog kraja
69     * binarnog zapisa pojavljivale 0. */

70     for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
71         putchar( x & maska ? '1' : '0' );

72     putchar('\n');
73 }

74 int main() {
75     int broj;
76     scanf("%x", &broj);

77     printf("Najveci:\n");
78     print_bits( najveci(broj) );

79     printf("Najmanji:\n");
```

```

76     print_bits( najmanji(broj) );
77     return 0;
78 }

```

### Rešenje 1.8

```

1  #include <stdio.h>

3  /* Funkcija postavlja na nulu n bitova pocev od pozicije p.
   * Pozicije se broje pocev od pozicije najnizeg bita,
   * pri cemu se broji od nule .
   * Npr, za n=5,  p=10
   *      1010 1011 1100 1101 1110 1010 1110 0111
   *      1010 1011 1100 1101 1110 1000 0010 0111
   */
9  unsigned reset(unsigned x, unsigned n, unsigned p ) {
11     /* Cilj nam je da samo zeljene bitove anuliramo, a da ostali
       ostanu nepromenjeni.
       * Formiramo masku koja ima n bitova postavljenih na 0 pocev od
       pozicije p,
13     * dok su svi ostali postavljeni na 1.
       *
15     * Na primer, za n=5 i p=10
       * formiramo masku oblika
17     * 1111 1111 1111 1111 1111 1000 0011 1111
       * To postizemo na sledeci nacin:
19     * ~0                      1111 1111 1111 1111 1111 1111 1111
       1111
       * (~0 << n)                1111 1111 1111 1111 1111 1111 1110 0000
21     * ~(~0 << n)              0000 0000 0000 0000 0000 0000 0001 1111
       * (~(~0 << n) << ( p-n+1)) 0000 0000 0000 0000 0000 0000
       0111 1100 0000
23     * ~(~(~0 << n) << ( p-n+1)) 1111 1111 1111 1111 1111
       1000 0011 1111
       */
25     unsigned maska = ~(~(~0 << n) << ( p-n+1));

27     return x & maska;

29 }

31 /* Funkcija postavlja na 1 n bitova pocev od pozicije p.
   * Pozicije se broje pocev od pozicije najnizeg bita,
   * pri cemu se broji od nule .
   * Npr, za n=5,  p=10
   *      1010 1011 1100 1101 1110 1010 1110 0111
   *      1010 1011 1100 1101 1110 1111 1110 0111
   */
33 /*
35     unsigned set(unsigned x, unsigned n, unsigned p ) {
       /* Kako zelimo da samo odredjenih n bitova postavimo na 1, dok
       ostali treba da ostanu netaknuti.
39     * Na primer, za n=5 i p=10
       * formiramo masku oblika 0000 0000 0000 0000 0000 0111 1100
       0000
41     * prateci vrlo slican postupak kao za prethodnu funkciju

```

```

43     */
    unsigned maska = ~(~0 << n) << (p-n+1);
45     return x|maska;
47 }
48
49 /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
    predstavlja n bitova
50 * pocev od pozicije p u binarnoj reprezentaciji broja x, pri cemu
    se pozicija broji
51 * sa desna ulevo, gde je pocetna pozicija 0.
52 * Na primer za n = 5 i p = 10 i broj
53 *   1010 1011 1100 1101 1110 1010 1110 0111
54 *   0000 0000 0000 0000 0000 0000 0000 1011
55 */
56 unsigned get_bits(unsigned x, unsigned n, unsigned p ) {
    /* Kreiramo masku kod koje su poslednjih n bitova 1, a
    ostali su 0.
57     * Na primer za n=5
58     * 0000 0000 0000 0000 0000 0000 0001 1111
59     */
    unsigned maska = ~(~0 << n);
61
    /* Pomeramo sadrzaj u desno tako da trazeno polje bude uz desni
    kraj.
62     * Zatim maskiramo ostale bitove, sem zeljenih n i vracamo
    vrednost */
    return maska & ( x >> (p-n+1));
65 }
66
67 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
    postavljeni na
68 vrednosti n bitova najnize tezine binarne reprezentacije broja y
69 */
70 unsigned set_n_bits(unsigned x, unsigned n, unsigned p , unsigned y)
71 {
    /* Kreiramo masku kod koje su poslednjih n bitova 1, a
    ostali su 0.
72     * Na primer za n=5
73     * 0000 0000 0000 0000 0000 0000 0001 1111
74     */
75     unsigned last_n_1 = ~(~0 << n);
76
77     /* Kao ranije u funkciji reset, kreiramo masku koja ima n bitova
    postavljenih
78     * na 0 pocevsi od pozicije p, dok su ostali bitovi 1.
79     * Na primer za n=5 i p =10
80     * 1111 1111 1111 1111 1111 1000 0011 1111
81     */
82     unsigned middle_n_0 = ~(~(~0 << n ) << (p-n+1));
83
84     /* x sa resetovanih n bita na pozicijama pocev od p */
85     unsigned x_reset = x & middle_n_0;
86
87     /* y cijih je n bitova najnize tezine pomereni tako da stoje

```



```

89     * pocev od pozicije p. Ostali bitovi su nule.
    * (y & last_n_1) resetuje sve bitove osim najnižih n
    */
91     unsigned y_shift_middle= (y & last_n_1) << (p-n+1);

93     return x_reset ^ y_shift_middle;
}

95

97 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
    njih n*/
    unsigned invert(unsigned x, unsigned n, unsigned p )
99 {
    /* Formiramo masku sa n jedinica pocev od pozicije p
    * Na primer za n=5 i p=10
    * 0000 0000 0000 0000 0000 0111 1100 0000
    */
103     unsigned maska = ~(~0 << n) << (p-n+1);

105     /* Operator ekskluzivno ili invertuje sve bitove gde je
    odgovarajuci
107     bit maske 1. Ostali bitovi ostaju nepromenjeni. */
    return maska ^ x;
109 }

111

113 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
    void print_bits( int x) {
        unsigned velicina = sizeof(int)*8;    /* Broj bitova celog broja
        */
115     unsigned maska;    /* Maska koju cemo koristiti za "ocitavanje"
        bitova */

117     /* Bitove u zapisu broja treba da ispisujemo sa leva na desno,
        tj. od bita najvece tezine ka
        * bitu najmanje tezine. Iz tog razloga, za pocetnu vrednost
        maske uzimamo vrednost
119     * ciji binarni zapis je takav da je bit najvece tezine 1, a svi
        ostali nule.
        * Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u
        desno, kako bismo ocitali
121     * naredni bit, gledano s leva na desno. Odgovarajuci karakter,
        ('0' ili '1'), ispisuje se na ekranu.
        *
123     * Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
        postavljen na 1,
        * neophodno je da maska bude neoznacena ceo broj i da se
        Siftovanjem u desno ova 1
125     * ne bi smatrala znakom i prepisivala, vec da bi nam se svakim
        Siftovanjem sa levog kraja
        * binarnog zapisa pojavljivala 0. */

127     for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
129         putchar( x & maska ? '1' : '0' );

```

## 1 Uvodni zadaci

```
131     putchar('\n');
132 }
133
134
135 int main() {
136     unsigned broj, p, n, y;
137     scanf("%u%u%u%u", &broj, &n, &p, &y);
138     printf("Broj %5u %25s= ", broj, "");
139     print_bits(broj);
140
141
142     printf("reset(%5u,%5u,%5u)%11s = ", broj, n, p, "");
143     print_bits( reset(broj, n, p));
144
145     printf("set(%5u,%5u,%5u)%13s = ", broj, n, p, "");
146     print_bits( set(broj, n, p));
147
148     printf("get_bits(%5u,%5u,%5u)%8s = ", broj, n, p, "");
149     print_bits( get_bits(broj, n, p));
150
151     printf("y = %31u = ", y);
152     print_bits(y);
153     printf("set_n_bits(%5u,%5u,%5u,%5u) = ", broj, n, p, y);
154     print_bits( set_n_bits(broj, n, p, y));
155
156     printf("invert(%5u,%5u,%5u)%10s = ", broj, n, p, "");
157     print_bits( invert(broj, n, p));
158
159     return 0;
160 }
```

### Rešenje 1.9

```
#include <stdio.h>
2
/* Funkcija broj x rotira u levo za n mesta
4  * Na primer za n =5 i x cija je interna reprezentacija
5  * 1010 1011 1100 1101 1110 0001 0010 0011
6  * 0111 1001 1011 1100 0010 0100 0111 0101
7  */
8 unsigned rotate_left(int x, unsigned n) {
9     unsigned first_bit;
10    /* Maska koja ima samo najvisi bit postavljen na 1 neophodna da
11    bismo pre siftovanja u levo za 1 sacuvali najvisi bit. */
12    unsigned first_bit_mask = 1 << (sizeof(unsigned)*8 -1);
13    int i;
14
15    /* n puta vrsimo rotaciju za jedan bit u levo */
16    for( i= 0; i<n; i++) {
17        /* odredujemo prvi bit*/
18        first_bit = x & first_bit_mask;
19        /* pomeramo sadrzaj broja x u levo za 1, a
20        * potom najnizi bit postavljamo na vrednost koju je imao
21        * prvi bit koji smo istisnuli Siftovanjem */
22    }
```

```

22     x = x<< 1 | first_bit >> (sizeof(unsigned)*8-1);
    }
24     return x;
}

26 /* Funkcija neoznaceni broj x rotira u desno za n
   * Na primer za n =5 i x cija je interna reprezentacija
28   * 1010 1011 1100 1101 1110 0001 0010 0011
   * 0001 1101 0101 1110 0110 1111 0000 1001
30   */
unsigned rotate_right(unsigned x, unsigned n) {
32     unsigned last_bit;
    int i;

34     /* n puta ponavljamo rotaciju u desno za jedan bit */
36     for(i=0; i<n; i++){
        last_bit = x & 1 ; /* bit najmanje tezine */

38         /* last_bit siftujemo u levo tako da najnizi bit dode do
           pozicije najviseg bita i
40         * nakon Siftovanja x za 1 u desno postavljamo x-ov najvisi
           bit na vrednost
           * najnizeg bita.
42         */
        x = x >> 1 | last_bit << (sizeof(unsigned)*8-1);
44     }

46     return x;
}

48 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je x
   oznaceni broj */
50 int rotate_right_signed(int x, unsigned n) {
    unsigned last_bit;
52     int i;

54     for(i=0; i<n; i++) {
        last_bit = x & 1; /* bit najmanje tezine */

56         /* Kako je x oznacen ceo broj, tada se prilikom Siftovanja u
           desno
58         * vrši aritmeticki sift i cuva se znak broja. Iza tog
           razloga imamo dva
           * slucaja u zavisnosti od znaka od x.
60         * Nije dovoljno da se ova provera izvrši pre petlje, jer
           rotiranjem u desno
           * na mesto najviseg bita moze doci i 0 i 1, nezavisno od
           pocetnog znaka x.
62         */
        if( x<0 )
64             /* Siftovanjem u desno broja koji je negativan dobijamo
               1 na najvisoj
               * poziciji. Na primer ako je x
66               * 1010 1011 1100 1101 1110 0001 0010 001b
               * (sa b oznacavamo u primeru 1 ili 0 na najnizoj
               poziciji)

```

```
68      *
69      * last_bit je
70      * 0000 0000 0000 0000 0000 0000 0000 000b
71      * nakon Siftovanja za 1 u desno
72      * 1101 0101 1110 0110 1111 0000 1001 0001
73      *
74      * da bismo najvisu 1 u x postavili na b nije dovoljno
da ga siftujemo
75      * na najvisu poziciju jer bi se time dobile 0, a nama
su potrebne 1
76      * zbog bitovskog &
77      * zato prvo komplementiramo, pa tek onda siftujemo
78      * ~last_bit << (sizeof(int)*8 -1)
79      * B000 0000 0000 0000 0000 0000 0000 0000
80      * (B oznacava ~b )
81      * i ponovo komplementiramo da bismo imali b na najvisoj
poziciji
82      * i sve 1 na ostalim pozicijama
83      * ~(~last_bit << (sizeof(int)*8 -1))
84      * b111 1111 1111 1111 1111 1111 1111 1111
85      */
86      x = (x >>1) & ~(~last_bit << (sizeof(int)*8 -1));
87      else
88      x = (x >>1) | last_bit<< (sizeof(int)*8 -1);
89  }
90
91  return x;
92 }
93
94
95 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
celog broja u memoriji */
96 void print_bits( int x) {
97     unsigned velicina = sizeof(int)*8;    /* Broj bitova celog broja
*/
98     unsigned maska;    /* Maska koju cemo koristiti za "ocitavanje"
bitova */
99
100    /* Bitove u zapisu broja treba da ispisujemo sa leva na desno,
tj od bita najvece tezine ka
    * bitu najmanje tezine. Iz tog razloga, za pocetnu vrednost
maske uzimamo vrednost
101    * ciji binarni zapis je takav da je bit najvece tezine 1, a svi
ostali nule.
    * Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u
desno, kako bismo ocitali
102    * naredni bit, gledano s leva na desno. Odgovarajuci karakter,
('0' ili '1'), ispisuje se na ekranu.
    *
103    * Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
postavljen na 1,
    * neophodno je da maska bude neoznacena ceo broj i da se
siftovanjem u desno ova 1
104    * ne bi smatrala znakom i prepisivala, vec da bi nam se svakim
siftovanjem sa levog kraja
105    * binarnog zapisa pojavljivale 0. */
106 }
```

```

110     for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
112         putchar( x & maska ? '1' : '0' );

114     putchar('\n');
115 }

116 int main() {
117     unsigned x, k;
118     scanf("%x%x", &x, &k);
119     printf("x %36s = ", "");
120     print_bits(x);
121     printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
122     print_bits( rotate_left(x, k));

124     printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
125     print_bits( rotate_right(x, k));

126     printf("rotate_right_signed(%7u,%6u) = ", x, k);
127     print_bits( rotate_right_signed(x, k));

128     return 0;
129 }
130
131
132

```

### Rešenje 1.10

```

1  #include <stdio.h>
2
3  /* Na primer za x cija binarna reprezentacija izgleda ovako
4   * 10101011110011011110000100100011
5   * 11000100100001111011001111010101
6   */
7
8
9  unsigned mirror(unsigned x) {
10     unsigned najnizi_bit;
11     unsigned rezultat = 0;
12
13     int i;
14     /* Krecemo od najnizeg bita u zapisu broja x i dodajemo ga u
15     rezultat */
16     for(i =0; i < sizeof(x)*8; i++) {
17         najnizi_bit = x & 1;
18         x >>=1;
19         /* Potiskujemo trenutni rezultat ka levom kraju tako svi
20         prethodno postavljani bitovi dobijaju vecu poziciju, a novi bit
21         postavljamo na najnizu poziciju */
22         rezultat <<= 1;
23         rezultat |= najnizi_bit;
24     }
25     return rezultat;
26 }
27
28 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju

```

```
    celog broja u memoriji */
void print_bits( int x) {
28     unsigned velicina = sizeof(int)*8;    /* broj bitova celog broja
    */
    unsigned maska;    /* maska koju cemo koristiti za "ocitavanje"
    bitova */
30
    /* Bitove u zapisu broja treba da ispisujemo sa leva na desno,
    tj. od bita najvece tezine ka
32     * bitu najmanje tezine. Iz tog razloga, za pocetnu vrednost
    maske uzimamo vrednost
    * ciji binarni zapis je takav da je bit najvece tezine 1, a svi
    ostali nule.
34     * Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u
    desno, kako bismo ocitali
    * naredni bit, gledano s leva na desno. Odgovarajuci karakter,
    ('0' ili '1'), ispisuje se na ekranu.
36     *
    * Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
    postavljen na 1,
38     * neophodno je da maska bude neoznacena ceo broj i da se
    siftovanjem u desno ova 1
    * ne bi smatrala znakom i prepisivala, vec da bi nam se svakim
    siftovanjem sa levog kraja
40     * binarnog zapisa pojavljivale 0. */

    for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
        putchar( x & maska ? '1' : '0' );
42
44     putchar('\n');
46 }

48 int main() {
    int broj;
50     scanf("%x", &broj);

52     /*Ispisujemo binarnu reprezentaciju unetog broja*/
    print_bits(broj);
54     putchar('\n');
    /*Ispisujemo binarnu reprezentaciju broja dobijenog
56     pozivom funkcije mirror
    */
58     print_bits( mirror(broj));

60     return 0;
}
```

### Rešenje 1.11

```
#include <stdio.h>
2
int Broj01(unsigned int n){
4
    int broj_nula, broj_jedinica;
6    unsigned int maska;
```

```

8  /* Postavljamo broj jedinica i broj nula na 0 */
   broj_nula=0;
10 broj_jedinica=0;

12 /* Postavljamo masku tako da pocinjemo sa analiziranjem bita
   najvece tezine */
   maska=1<<(sizeof(unsigned int)*4-1);
14

16 /* Dok ne obidjemo sve bitove u zapisu broj n */
18 while(maska!=0){

20     /* Proveravamo da li se na poziciji koju odredjuje maska nalazi
       0 ili 1 i uvecavamo odgovarajuci brojac */
       if(n&maska){
22         broj_jedinica++;
       }
24     else{
       broj_nula++;
26     }

28     /* Pomeramo masku u desnu stranu tako da mozemo da očitamo
       vrednost narednog bita */
       maska=maska>>1;
30 }

32 /* Ako je broj jedinica veci od broja nula vracamo 1, u suprotnom
   vracamo 0 */
   return (broj_jedinica>broj_nula)? 1: 0;
34 }

36 int main(){
38     unsigned int n;

40     /* Ucitavamo broj */
     scanf("%u", &n);

42     /* Ispsujemo vrednost funkcije */
44     printf("%d\n", Broj01(n));

46     return 0;
   }

```

### Rešenje 1.12

```

#include <stdio.h>

2 int broj_parova(unsigned int x){
4     int broj_parova;
6     unsigned int maska;

```

## 1 Uvodni zadaci

```
8  /* Postavljamo broj parova na 0 */
   broj_parova=0;
10
   /* Postavljamo masku tako da mozemo da procitamo da li su dva
      najmanja bita u zapisu broja x 11 */
12  /* broj 3 je binarno 000....00011 */
   maska=3;
14
16
   /* Dok ne obidjemo sve parove bitova u zapisu broja x */
18  while(x!=0){
20
      /* Proveravamo da li se na najmanjim pozicijama broj x nalazi 11
         par */
      if((x & maska) == maska){
22         broj_parova++;
      }
24
      /* Pomeramo broj u desnu stranu tako da mozemo da ocitamo
         vrednost sledeceg para bitova */
26     x=x>>1;
   }
28
30  return broj_parova;
32 }
34 int main(){
   unsigned int x;
36
   /* Ucitavamo broj */
38  scanf("%u", &x);
40
   /* Ispsujemo vrednost funkcije */
   printf("%d\n", broj_parova(x));
42
   return 0;
44 }
```

### Rešenje 1.13

### Rešenje 1.14

```
#include <stdio.h>
2
/*
4  Niska koju formiramo je duzine
   (sizeof(unsigned int)*8)/4 +1
6  jer za svaku heksadekadnu cifru nam trebaju 4 binarne cifre i
   jedna dodatna pozicija nam treba za terminirajucu nulu.
8
   Prethodni izraz je identican sa sizeof(unsigned int)*2+1.
```



```

10  Na primer, ako je duzina unsigned int 4 bajta onda je MAX_DUZINA 9
11  */
12
13  #define MAX_DUZINA sizeof(unsigned int)*2 +1
14
15
16  void prevod(unsigned int x, char s[]){
17
18      int i;
19      unsigned int maska;
20      int vrednost;
21
22      /* Heksadekadni zapis broja 15 je 000...0001111 - ovo nam
23       odgovara ako hocemo da citamo 4 uzastopne cifre */
24      maska=15;
25
26      /*
27       Broj cemo citati od pozicije najmanje tezine ka poziciji najvece
28       tezine;
29       npr. za broj 00000000001101000100001111010101
30       u prvom koraku cemo procitati bitove:
31       0000000000110100010000111101<0101> (bitove izdvojene sa <...>)
32       u drugom koraku cemo procitati:
33       000000000011010001000011<1101>0101
34       u trecem koraku cemo procitati:
35       00000000001101000100<0011>11010101
36       i tako redom
37
38       indeks i oznacava poziciju na koju smestamo vrednost
39
40      */
41      for(i=MAX_DUZINA-2; i>=0; i--){
42          /* Vrednost izdvojene cifre */
43          vrednost=x&maska;
44
45          /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter
46             dobijamo dodavanjem ASCII koda '0' */
47          if(vrednost<10){
48              s[i]=vrednost+'0';
49          }
50          else{
51              /*
52               Ako je vrednost iz opsega od 10 do 15 odgovarajuci karakter
53               dobijamo tako sto prvo oduzmemo 10
54               (dobijamo vrednosti od 0 do 5) pa dodamo ASCII kod 'A' (
55               time dobijamo slova 'A', 'B', ... 'F')
56              */
57              s[i]=vrednost-10+'A';
58          }
59
60          /* Broj pomeramo za 4 bita u desnu stranu tako da mozemo da
61             procitamo sledecu cifru */
62          x=x>>4;
63      }
64

```

## 1 Uvodni zadaci

---

```
56     s[MAX_DUZINA-1]='\0';
57 }
58
59 int main(){
60
61     unsigned int x;
62     char s[MAX_DUZINA];
63
64     /* Ucitavamo broj */
65     scanf("%u", &x);
66
67     /* Pozivamo funkciju */
68     prevod(x, s);
69
70     /* Ispsujemo dobijenu nisku */
71     printf("%s\n", s);
72
73     return 0;
74 }
```

Rešenje 1.15

Rešenje 1.16

Rešenje 1.17

```
#include <stdio.h>
2 /* Bitno!
   * Kad pisemo rekursivnu funkciju moramo da obezbedimo:
4   * - Izlazak iz rekurziije, rekurziije obicno trivijalnim slucajem.
   * - Rekursivni poziv kojim se resava problem manje dimenzije.
6   * - Rekurzija nam omogucava pisanje elegantnijih resenja.
   * - Rekursivne funkcije trose mnogo vise memorije nego
8   *   iterativne koje resavaju isti problem.
   */
10
11 /* Iskomentarisan je deo koji se ispisuje svaki put kad se
12 * udje u funkciju. Odkomentarisati pozive printf funkcije u
13 * obe funkcije da uocite razliku u broju rekursivnih poziva
14 * obe verzije.
15 *
16 * Kako se menja stanje na sistemskom steku,
17 * dok se funkcija izvrsava?
18 */
19
20 /* Linearno resenje se zasniva na cinjenici:
21 *  $x^0 = 1$ 
22 *  $x^k = x * x^{(k-1)}$ 
23 */
24 int stepen(int x, int k)
25 {
26     printf("Racunam stepen (%d, %d)\n", x, k);
27     if(k==0)
28         return 1;
```

```

    return x * stepen(x, k-1);
30
    /*Celo telo funkcije se moze ovako kratko zapisati
32    return k == 0 ? 1 : x * stepen(x,k-1); */
}
34
/*Druga verzija prethodne funkcije.
36 Obratiti paznju na efikasnost u odnosu na prvu verziju! */
38
/* Logaritamsko resenje je zasnovano na cinjenicama:
40 * -  $x^0 = 1$ ;
42 * -  $x^k = x * (x^2)^{(k/2)}$  , za neparno k
44 * -  $x^k = (x^2)^{(k/2)}$  , za parno k
46 *
48 * Ovom resenju ce biti potrebno manje
49 * rekurzivnih poziva da bi doslo do rezultata,
50 * i stoga je efikasnije.
51 */
52
53 int stepen2(int x, int k)
54 {
55     printf("Racunam stepen2 (%d, %d)\n",x,k);
56     if( k == 0)
57         return 1;
58
59     /*Ako je stepen paran*/
60     if((k % 2) == 0)
61         return stepen2(x*x, k/2);
62     /*Inace (ukoliko je stepen neparan) */
63     return x*stepen2(x*x, k/2);
64 }
65
66 main() {
67     int x, k;
68     scanf("%d%d", &x, &k);
69
70     printf("%d",stepen(2,10));
71     printf("\n-----\n");
72     printf("%d\n",stepen2(2,10));
73 }

```

### Rešenje 1.18

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje
7  * uzajamnu (posrednu) rekurziju.
8  */
9
10 /* Deklaracija funkcije neparan mora da bude navedena
11  * jer se ta funkcija koristi u telu funkcije paran,
12  * tj. koristi se pre svoje definicije.
13  * Funkcija je mogla biti deklarisan i u telu funkcije paran.

```

## 1 Uvodni zadaci

```
14  */
16  unsigned neparan(unsigned n);
18  /* Funkcija vraca 1 ako broj n ima paran broj cifara inace vraca 0.
   */
   unsigned paran(unsigned n) {
20     if(n>=0 && n<=9)
       return 0;
22     else
       return neparan(n/10);
24 }

26  /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace vraca
   0. */
   unsigned neparan(unsigned n) {
28     if(n>=0 && n<=9)
       return 1;
30     else
       return paran(n/10);
32 }

34  /* Glavna funkcija za testiranje */
   int main( ) {
36     int n;
       printf("Unesite ceo broj: ");
38     scanf("%d", &n);

40     printf("Uneti broj ima %sparan broj cifara\n", (paran(n) == 1 ?
       " ": " ne"));
42     return 0;
   }
```

### Rešenje 1.19

```
#include <stdio.h>

2
/* Repno-rekurzivna (eng. tail recursive) je ona funkcija
4  * Cije se telo završava rekurzivnim pozivom, pri čemu
   * taj rekurzivni poziv ne učestvuje u nekom izrazu.
6  *
   * Kod ovih funkcija se po završetku za tekuci rekurzivni poziv
8  * umesto skoka na adresu povratka skace na adresu
   * povratka za prethodni poziv, odnosno za poziv na manjoj
10  * dubini. Time se stedi i prostor i vreme.
   *
12  * Ovakve funkcije se mogu lako zameniti odgovarajućom
   * iterativnom funkcijom,
14  * čime se smanjuje prostorna složenost algoritma.
   */
16
   /* Pomoćna funkcija koja izračunava n! * result.
18  * Koristi repnu rekurziju. */
```

```

20 /* Result je argument u kom cemo akumulirati do tada izracunatu
   * vrednost faktoriijela.
22 * Kada zavrismo, tj. kada dodjemo do izlaza iz rekurzije potrebno
   * je
   * da vratimo result. */
24 int faktoriijelRepna(int n, int result) {
   if (n == 0)
26     return result;

28     return faktoriijelRepna(n - 1, n * result);
   }
30
31 /* Sada Zelimo da se oslobodimo repne rekurzije koja postoji u
32 * funkciji faktoriijelRepna, koristeci algoritam sa predavanja.
   *
34 * Najpre cemo vrednost argumenta funkcije postaviti na vrednost
   * koja bi se prosledjivala rekurzivnom pozivu i pomocu goto naredbe
36 * vratiti se na pocetak tela funkcije.
   */
38
39 int faktoriijelRepna_v1(int n, int result) {
40     pocetak:
   if (n == 0)
42         return result;

44     result = n*result;
   n=n-1;
46     goto pocetak;
   }
48
49 /* Pisanje bezuslovnih skokova (goto naredbi)
50 * nije dobra programerska praksa.
   * Iskoristicemo prethodni medjukorak da bismo
52 * dobili iterativno resenje bez bezuslovnih skokova.
   */
54 int faktoriijelRepna_v2(int n, int result) {
   while( n!=0){
56         result = n*result;
   n=n-1;
58     }

60     return result;
   }
62
63
64
65 /* Nasim gore navedenim funkcijama pored n, mora da se salje
66 * i 1 za vrednost drugog argumenta u kome ce se akumulirati
   * rezultat. Funkcija faktoriijel(n) je ovde radi udobnosti
68 * korisnika, jer je sasvim prirodno da za faktoriijel zahteva
   * samo 1 parametar.
70 * Funkcija faktoriijel izracunava n!, tako Sto odgovarajucoj gore
   * navedenoj funkciji koja zaista racuna faktoriijel,
72 * salje ispravne argumente i vraca rezultat koju joj ta funkcija
   * vrati.
74 * Za testiranje, zameniti u telu funkcije faktoriijel poziv

```

## 1 Uvodni zadaci

---

```
76  * faktorijelRepna sa pozivom faktorijelRepna_v1, a zatim sa pozivom
    * funkcije faktorijelRepna_v2.
    */
78  int faktorijel(int n) {
    return faktorijelRepna(n, 1);
80  }

82  /* Test program */
    int main(){
84      int n;

86      printf("Unesite n (<= 12): ");
      scanf("%d", &n);

88      printf("%d! = %d\n", n , faktorijel(n));

90      return 0;
92  }
```

Rešenje 1.20

Rešenje 1.21

Rešenje 1.22

```
1  #include <stdio.h>
2  #define MAX_DIM 1000

4  /* Bitno!
    * Kad pisemo rekurzivnu funkciju moramo da obezbedimo:
6  * - Izlazak iz rekurzije (obicno trivijalnim slucajem).
    * - Rekurzivni poziv kojim se reSava problem manje
8  *   dimenzije.
    *
10  * Rekurzija nam omogucava pisanje elegantnijih resenja.
    * Rekurzivne funkcije troSe mnogo vise memorije nego
12  *   iterativne koje reSavaju isti problem.
    */

14
16  /*
    * n==0, suma(a,0) = 0
18  * n >0, suma(a,n) = a[n-1]+suma(a,n-1)
    *   Suma celog niza je jednaka sumi prvih n-1 elementa
20  *   uveCennoj za poslednji element celog niza.
    */
22  int sumaNiza(int *a, int n)
    {
24      /* Ne stavljamo strogu jednakost n==0,
        * za slucaj da korisnik prilikom prvog poziva,
26      * poSalje negativan broj za velicinu niza.
        */
28      if(n<=0 )
        return 0;
```

```

30     return a[n-1] + sumaNiza(a,n-1);
32 }

34 /*
35  * n==0, suma(a,0) = 0
36  * n >0, suma(a,n) = a[0]+suma(a+1,n-1)
37  *      Suma celog niza je jednaka zbiru prvog elementa
38  *      niza i sume preostalih n-1 elementa.
39  */
40 int sumaNiza2(int *a, int n)
41 {
42     if(n<=0)
43         return 0;
44
45     return a[0] + sumaNiza2(a+1,n-1);
46 }
47
48 int main()
49 {
50     int x, a[MAX_DIM];
51     int n, i=0;
52
53     /* Ucitavamo broj elemenata niza */
54     scanf("%d", &n);
55
56     /* Ucitavamo n elemenata niza. */
57     for(i=0; i<n; i++)
58         scanf("%d", &a[i]);
59
60     /*      int a[]={ 10, 2, 3, 45, 21};
61     int n = sizeof(a)/sizeof(int);
62     * Ovako odredjivanje velicine niza je primenljivo
63     * samo na nizove koji su definisani i inicijalizovani
64     * kao u prethodnom redu, navodjenjem elemenata.
65     */
66
67     printf("Suma elemenata je %d\n",sumaNiza(a, n));
68
69     /*
70     printf("Suma elemenata je %d\n",sumaNiza2(a, n));
71     */
72     return 0;
73 }

```

### Rešenje 1.23

```

1  #include <stdio.h>
2  #define MAX_DIM 256
3
4  /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
5     dimenzije n */
6  int maksimum_niza(int niz[], int n) {
7     /* Izlazak iz rekurziije: ako je niz dimenzije jedan, najveći je
8     ujedno i jedini element niza */

```

## 1 Uvodni zadaci

```
7   if (n==1) return niz[0];

9   /* ReSavamo problem manje dimenzije */
   int max=maksimum_niza(niz, n-1);

11  /* Ako nam je poznato resenje problema dimenzije n-1, reSavamo
   problem dimenzije n */
13  return niz[n-1] > max ? niz[n-1] : max ;
}

15
16 int main ()
17 {
   int brojevi[MAX_DIM];
19   int n;

21   /* Sve dok ne dodjemo do kraja ulaza, ucitavamo brojeve u niz; i
   predstavlja indeks tekućeg broja. */
   int i=0;
23   while(scanf("%d", &brojevi[i])!=EOF){
       i++;
25   }
   n=i;

27   /* Stampamo maksimum unetog niza brojeva */
29   printf("%d\n", maksimum_niza(brojevi, n));
   return 0;
31 }
```

### Rešenje 1.24

```
1  #include <stdio.h>
2  #define MAX_DIM 256

4  int skalarno(int a[], int b[], int n)
   {
6     /* Izlazak iz rekurzije */
     if(n==0) return 0;

8     /* Na osnovu rešenja problema dimenzije n-1, resavamo problem
     dimenzije n */
10    else return a[n-1] * b[n-1] + skalarno(a,b,n-1);
   }

12
13 int main()
14 {
   int i, a[MAX_DIM], b[MAX_DIM], n;

16   /* Unosimo dimenziju nizova, */
18   scanf("%d",&n);

20   /* a zatim i same nizove. */
   for(i=0; i<n; i++)
22       scanf("%d",&a[i]);

24   for(i=0; i<n; i++)
```



```

    scanf("%d",&b[i]);
26
    /* Ispisujemo rezultat skalarnog proizvoda dva učitana niza. */
28    printf("%d\n", skalarno(a,b,n));
30    return 0;
}

```

### Rešenje 1.25

```

#include<stdio.h>
2 #define MAX_DIM 256

4
int br_pojave(int x, int a[], int n)
6 {
    /* Izlazak iz rekurziije */
8     if(n==1) return a[0]==x ? 1 : 0;

10     int bp = br_pojave(x, a, n-1);
    return a[n-1]==x ? 1 + bp : bp;
12 }

14 int main()
{
16     int x, a[MAX_DIM];
    int n, i=0;

18     /* UCitavamo broj koji se trazi */
20     scanf("%d", &x);

22     /* Sve dok ne dodjemo do kraja ulaza, ucitavamo brojeve u niz; i
    predstavlja indeks tekuceg broja */
    i=0;
24     while(scanf("%d", &a[i])!=EOF){
        i++;
26     }
    n=i;

28     /* Ispisujemo broj pojave broja x u niz a */
30     printf("%d\n", br_pojave(x,a,i));
    return 0;
32 }

```

### Rešenje 1.26

```

#include<stdio.h>
2 #define MAX_DIM 256

4
int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
{
6     /* Ako niz ima manje od tri elementa izlazimo iz rekurziije */
    if(n<3) return 0;
8

```

## 1 Uvodni zadaci

```

    else return (a[n-3]==x)&&(a[n-2]==y)&&(a[n-1]==z) ||
    tri_uzastopna_clana(x,y,z,a,n-1);
10 }

12 int main ()
13 {
14     int x,y,z, a[MAX_DIM];
15     int n;
16
17     /* UCitavaju se tri cela broja za koje se ispituje da li su
18     uzastopni Clanovi niza */
19     scanf("%d%d%d",&x,&y,&z);
20
21     /* Sve dok ne dodjemo do kraja ulaza, uCitavamo brojeve u niz */
22     int i=0;
23     while(scanf("%d", &a[i])!=EOF){
24         i++;
25     }
26     n=i;
27
28     if(tri_uzastopna_clana(x,y,z,a,i))
29         printf("da\n");
30     else
31         printf("ne\n");
32
33     return 0;
34 }
```

### Rešenje 1.27

```

#include<stdio.h>
2 #include<string.h>
/* niska moze imati najviše 32 karaktera + 1 za terminalnu nulu */
4 #define MAX_DIM 33

6 int palindrom(char s[], int n)
7 {
8     if((n==1) || (n==0)) return 1;
9     return (s[n-1]==s[0]) && palindrom(s+1, n-2);
10 }

12
13
14 int main()
15 {
16     char s[MAX_DIM];
17     int n;
18
19     /* Ucitavamo nisku sa ulaza */
20     scanf("%s",s);
21
22     /* Odredjujemo duzinu niske */
23     n=strlen(s);
24
25     /* Ispisujemo na izlazu poruku da li je niska palindrom ili nije
26     */
27 }
```

```

    if(palindrom(s,n))
26         printf("da\n");
    else printf("ne\n");
28
    return 0;
30 }

```

### Rešenje 1.28

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #define MAX_DUZINA_NIZA 50

5  void ispisiNiz(int a[], int n){
   int i;

7
   for(i=1;i<=n;i++)
9       printf("%d ", a[i]);
   printf("\n");
11 }

13 /* Funkcija proverava da li se x vec
   nalazi u permutaciji na prethodnih 1...n mesta*/
15 int koriscen(int a[], int n, int x){
   int i;
17   for(i=1; i<=n; i++)
       if(a[i] == x) return 1;
19
   return 0;
21 }

23 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n}
   * a[] je niz u koji smesta permutacije
25 * m - oznacava da se na m-tu poziciju u permutaciji
   * smesta jedan od preostalih celih brojeva
27 * n- je velicina skupa koji se permutuje
   * Funkciju pozivamo sa argumentom m=1 jer krecemo da
29 * formiramo permutaciju od 1. pozicije i nikada
   * ne koristimo a[0].
31 */
void permutacija(int a[], int m, int n){
33     int i;

35     /* Izlaz iz rekurzije:
   * Ako je pozicija na koju treba smestiti broj premasila
37 * velicinu skupa, onda se svi brojevi vec nalaze u
   * permutaciji i ispisujemo permutaciju. */
39     if(m>n) {
        ispisiNiz(a,n);
41         return;
   }

43
   /*Ideja: pronalazimo prvi broj koji mozemo da
45   postavimo na m-to mesto u nizu (broj koji se do
   sada nije pojavio u permutaciji). Zatim, rekurzivno

```

## 1 Uvodni zadaci

```
47     pronalazimo one permutacije koje odgovaraju
48     ovako postavljenom pocetku permutacije.
49     Kada to završimo, proveravamo da li postoji jos neki
50     broj koji moze da se stavi na m-to mesto u nizu
51     (to se radi u petlji). Ako
52     ne postoji, funkcija je završila sa radom.
53     Ukoliko takav broj postoji, onda ponovo pozivamo
54     rekursivno pronalazenje odgovarajucih permutacija,
55     ali sada sa drugacije postavljanim prefiksom. */
56
57
58     for(i=1;i<=n;i++){
59         /* Ako se broj i nije do sada pojavio u permutaciji
60          * od 1 do m-1 pozicije, onda ga stavljamo na poziciju m
61          * i pozivamo funkciju da napravi permutaciju za jedan
62          * vece duzine, tj. m+1. Inace, nastavljamo dalje, trazeci
63          * broj koji se nije pojavio do sada u permutaciji.
64          */
65         if(! koriscen(a,m-1,i)) {
66             a[m]=i;
67             /* Pozivamo ponovo funkciju da dopuni ostatak
68              * permutacije posle upisivanja i na poziciju m.
69              */
70             permutacija(a,m+1,n);
71         }
72     }
73 }
74
75 int main(void) {
76     int n;
77     int a[MAX_DUZINA_NIZA];
78
79     printf("Unesite duzinu permutacije: ");
80     scanf("%d", &n);
81     if ( n < 0 || n >= MAX_DUZINA_NIZA ) {
82         fprintf(stderr, "Duzina permutacije mora biti broj veci od 0
83         i manji od %d!\n", MAX_DUZINA_NIZA);
84         exit(EXIT_FAILURE);
85     }
86
87     permutacija(a,1,n);
88
89     exit(EXIT_SUCCESS);
90 }
```

### Rešenje 1.29

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Rekursivna funkcija za racunanje binomnog koeficijenta.      */
5  /* ako je k=0 ili k=n, onda je binomni koeficijent 0
6   * ako je k izmedju 0 i n, onda je bk(n,k) = bk(n-1,k-1) + bk(n-1,k)
7   */
8  /*
9   */
10 int binomniKoeficijent(int n, int k) {
```

```

    return (0 < k && k < n) ? binomniKoeficijent (n-1, k-1) +
    binomniKoeficijent (n-1, k) : 1;
9 }
/* Iterativno izracunavanje datog binomnog koeficijenta.
11
int binomniKoeficijent (int n, int k) {
13     int i, j, b;
    for (b=i=1, j=n; i<=k; b=b*j--/i++);
15     return b;
}
17
*/
19
/* Prostim opažanjem se uocava da se svaki element n-te hipotenuze (
    osim ivicnih 1)
21 * dobija kao zbir 2 elementa iz n-1 hipotenuze. Uz pomenute dve
    nove ivicne jedinice lako se zakljucuje
    * da ce suma elementa n-te hipotenuze biti tacno 2 puta veca.
23 */
int sumaElemenataHipotenuze(int n)
25 {
    return n > 0 ? 2 * sumaElemenataHipotenuze(n-1) : 1;
27 }
29
int main () {
31     int n, k, i, d;
33
    printf("Unesite velicinu Paskalovog trougla: \n");
35     scanf("%d", &d);
37
    /* Ispisivanje Paskalovog trougla */
    putchar ('\n');
39     for (n=0; n<=d; n++) {
        for (i=0; i<d-n; i++) printf (" ");
41         for (k=0; k<=n; k++) printf ("%4d", binomniKoeficijent(n, k));
        putchar ('\n');
43     }
45
    printf("Racunamo sumu koje hipotenuze: \n");
    scanf("%d", &n);
47
    if(n<0){
49         fprintf(stderr, "Redni broj hipotenuze mora biti veci ili
            jednak od 0!\n");
            exit(EXIT_FAILURE);
51     }
    printf("Suma %d. hipotenuze je: %d\n",n, sumaElemenataHipotenuze(n
    ));
53
    exit(EXIT_SUCCESS);
55 }

```

## Rešenje 1.33

```
1  #include <stdio.h>
2
3  /* funkcija koja broji bitove svog argumenta*/
4  /*
5   * ako je x ==0, onda je  count(x) = 0
6   * inace count(x) = najvisi_bit +count(x<<1)
7   *
8   * Za svaki naredni rekurzivan poziv prosleduje se x<<1.
9   * Kako se siftovanjem sa desne strane uvek dopisuju 0,
10  * argument x ce u nekom rekurzivnom pozivu biti baS 0 i
11  * izacicemo iz rekurzije.
12  */
13
14  int count(int x) {
15      /* izlaz iz rekurzije*/
16      if(x==0)
17          return 0;
18
19      /*Dakle, neki bit je postavljen na 1.*/
20      /* Proveravamo vrednost najviseg bita
21       * Kako za rekurzivni poziv moramo slati siftovano x i
22       * x je oznacen ceo broj, onda ne smemo koristiti siftovanje
23       * desno, jer funkciji moze biti prosleden i negativan broj.
24       * Iz tog razloga, odlucujemo se da proveramo najvisi,
25       * umesto najnizeg bita*/
26      if( x& (1<<(sizeof(x)*8-1)))
27          return 1 +count(x<<1);
28      /* Najvisi bit je 1.
29       * Sacekacemo da zavrshi poziv koji racuna koliko ima
30       * jedinica u ostatku binarnog zapisa x i potom uvecati
31       * taj rezultat za 1. */
32      else
33          /* Najvisi bit je 0. Stoga je broj jedinica u zapisu x isti
34           * kao broj jedinica u zapisu broja x<<1, jer se
35           * siftovanjem u levo sa desne stane dopisuju 0.*/
36          return count(x<<1);
37
38      /* jednolinijska return naredba sa proverom i rekurzivnim
39      pozivom
40      return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) +  count(x<<1);
41      */
42  }
43
44  int main() {
45      int x;
46      scanf("%x", &x);
47      printf("%d\n",count(x));
48
49      return 0;
50  }
```

Rešenje 1.35

---

```

1  #include<stdio.h>
2
3
4  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
   broj u */
5  int max_heksadekadna_cifra(unsigned x) {
6      /* izlazak iz rekurzije */
7      if(x==0) return 0;
8      /* Odredjivanje poslednje heksadekadne cifre u broju*/
9      int poslednja_cifra = x&15;
10     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz
       njega izbrise poslednja heksadekadna cifra*/
11     int max_bez_poslednje_cifre = max_heksadekadna_cifra(x>>4);
12     return poslednja_cifra > max_bez_poslednje_cifre ?
       poslednja_cifra : max_bez_poslednje_cifre;
13 }
14
15
16
17
18 int main()
19 {
20     unsigned x;
21     scanf("%u", &x);
22     printf("%d\n", max_heksadekadna_cifra(x));
23     return 0;
24 }

```

### Rešenje 1.36

```

1  #include<stdio.h>
2
3
4  /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
   */
5  int max_oktalna_cifra(unsigned x) {
6      /* Izlazak iz rekurzije */
7      if(x==0) return 0;
8      /* Odredjivanje poslednje oktalne cifre u broju*/
9      int poslednja_cifra = x&7;
10     /* Odredjivanje maksimalne oktalne cifre broja kada se iz njega
       izbrise poslednja oktalna cifra*/
11     int max_bez_poslednje_cifre = max_oktalna_cifra(x>>3);
12     return poslednja_cifra > max_bez_poslednje_cifre ?
       poslednja_cifra : max_bez_poslednje_cifre;
13 }
14
15
16
17
18 int main()
19 {
20     unsigned x;
21     scanf("%u", &x);
22     printf("%d\n", max_oktalna_cifra(x));
23     return 0;
24 }

```





# Glava 2

## Pokazivači

### 2.1 Pokazivačka aritmetika

**Zadatak 2.1** Milen: ovako definisan zadatak zahteva dva programa kao resenja, a ne jedan sa definisane dve funkcije. Za dati celobrojni niz dimenzije  $n$ , napisati funkciju koja obrće njegove elemente:

- (a) korišćenjem indeksne sintakse,
- (b) korišćenjem pokazivačke sintakse.

Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju niza  $n$  ( $0 < n \leq 100$ ), a zatim elemente niza. Prikazati sadržaj niza posle poziva funkcije za obrtanje elemenata niza.

<i>Test 1</i>	<i>Test 2</i>
<pre>Ulaz:  3       1 -2 3 Izlaz: 3 -2 1</pre>	<pre>Ulaz:  0 Izlaz: Greska: neodgovarajuca dimenzija niza.</pre>

**Zadatak 2.2** Dat je niz realnih brojeva dimenzije  $n$ .

- (a) Napisati funkciju `zbir` koja izračunava zbir elemenata niza.
- (b) Napisati funkciju `proizvod` koja izračunava proizvod elemenata niza.
- (c) Napisati funkciju `min_element` koja izračunava najmanji elemenat niza.
- (d) Napisati funkciju `max_element` koja izračunava najveći elemenat niza.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju  $n$  ( $0 < n \leq 100$ ) realnog niza, a zatim i elemente niza. Na standardni izlaz ispisati zbir, proizvod, minimalni i maksimalni element učitano niza.

### Test 1

```
|| Ulaz: 3
||      -1.1 2.2 3.3
|| Izlaz: zbir = 4.400
||         proizvod = -7.986
||         min = -1.100
||         max = 3.300
```

**Zadatak 2.3** Korišćenjem pokazivačke sintakse, napisati funkciju koja vrednosti elemenata u prvoj polovini niza povećava za jedan, a u drugoj polovini smanjuje za jedan. Ukoliko niz ima neparan broj elemenata, onda vrednost srednjeg elementa niza ostaviti nepromenjenim. Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojong niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom. *Jelena: Sta kazete na to da prekoracenja dimenzije niza u razlicitim zadacima razlicito obradjujemo. Na primer, mozemo da unosimo dimenziju niza sve dok se ne unese broj koji je u odgovarajucem opsegu, ili mozemo da dimenziju postavimo na 1 ako je korisnik uneo broj manji od 1, a na MAX ako je korisnik uneo broj veci od MAX, itd?*

### Test 1

```
|| Ulaz: 5
||      1 2 3 4 5
|| Izlaz: 2 3 3 3 4
```

### Test 2

```
|| Ulaz: 4
||      4 -3 2 -1
|| Izlaz: 5 -2 1 -2
```

### Test 3

```
|| Ulaz: 0
|| Izlaz: Greska: neodgovarajuca dimenzija niza.
```

### Test 4

```
|| Ulaz: 101
|| Izlaz: Greska: neodgovarajuca dimenzija niza.
```

**Zadatak 2.4** Napisati program koji ispisuje broj prihvaćenih argumenata komandne linije, a zatim i same argumenate kojima prethode njihovi redni brojevi. Nakon toga ispisati prve karaktere svakog od argumenata. Zadatak rešiti:

- (a) korišćenjem indeksne sintakse,
- (b) korišćenjem pokazivačke sintakse.

*Jelena: Da li je ok da ovaj zadatak pod a i b resim na nacin na koji sam resila, odnosno, da jedno od ta dva resenja iskomentarisem? Milena: Meni se cini da je bolje bez komentarisanja, vec da su oba prisutna.*

### Test 1

```

Poziv: ./a.out prvi 2. treci -4
Izlaz: 5
      0 ./a.out
      1 prvi
      2 2.
      3 treci
      4 -4
      . p 2 -

```

### Test 2

```

Poziv: ./a.out
Izlaz: 1
      0 ./a.out
      .

```

**Zadatak 2.5** Korišćenjem pokazivačke sintakse, napisati funkciju koja za datu nisku ispituje da li je palindrom. Napisati program koji vrši prebrojavanje argumenata komandne linije koji su palindromi.

### Test 1

```

Poziv: ./a.out programiranje anavolimilovana topot ana anagram t
Izlaz: 4

```

### Test 2

```

Poziv: ./a.out a b 11 212
Izlaz: 4

```

### Test 3

```

Poziv: ./a.out
Izlaz: 0

```

**Zadatak 2.6** Napisati program koji kao prvi argument komandne linije prihvata putanju do datoteke za koju treba proveriti koliko reči ima  $n$  karaktera, gde se  $n$  zadaje kao drugi argument komandne linije. Smatrati da reč ne sadrži više od 100 karaktera. U zadatku ne koristiti ugrađene funkcije za rad sa niskama, već implementirati svoje koristeći pokazivačku sintaksu.

### Test 1

```

Poziv: ./a.out ulaz.txt 1
ulaz.txt: Ovo je sadržaj datoteke i u njoj ima reci koje imaju
          1 karakter
Izlaz: 3

```

### Test 2

```

Poziv: ./a.out ulaz.txt
Izlaz: Greska: Nedovoljan broj argumenata komandne linije.
      Program se poziva sa ./a.out ime_dat br_karaktera.

```

### Test 3

```

Poziv: ./a.out ulaz.txt 2
      (ne postoji datoteka ulaz.txt)
Izlaz: Greska: Neuspesno otvaranje datoteke ulaz.txt.

```

**Zadatak 2.7** Napisati program koji kao prvi argument komandne linije prihvata putanju do datoteke za koju treba proveriti koliko reči ima zadati sufiks (ili prefiks), koji se zadaje kao drugi argument komandne linije. Smatrati da reč ne sadrži više od 100 karaktera. Program je neophodno pozvati sa jednom od opcija `-s` ili `-p` u zavisnosti od čega treba proveriti koliko reči ima zadati sufiks (ili prefiks). U zadatku ne koristiti ugrađene funkcije za rad sa niskama, već implementirati svoje koristeći pokazivačku sintaksu.

### Test 1

```
Poziv:    ./a.out ulaz.txt ke -s
ulaz.txt: Ovo je sadržaj datoteke i u njoj ima reci koje se
          završavaju na ke
Izlaz:    2
```

### Test 2

```
Poziv:    ./a.out ulaz.txt sa -p
ulaz.txt: Ovo je sadržaj datoteke i u njoj ima reci koje
          počinju sa sa
Izlaz:    3
```

### Test 3

```
Poziv:    ./a.out ulaz.txt sa -p
          (ne postoji datoteka ulaz.txt)
Izlaz:    Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

### Test 3

```
Poziv:    ./a.out ulaz.txt
Izlaz:    Greska: Nedovoljan broj argumenata komandne linije.
          Program se poziva sa ./a.out ime_dat suf/pref -s/-p.
```

## 2.2 Višedimenzioni nizovi

**Zadatak 2.8** Data je kvadratna matrica dimenzije  $n$ .

- Napisati funkciju koja izračunava trag matrice (sumu elemenata na glavnoj dijagonali).
- Napisati funkciju koja izračunava euklidsku normu matrice (koren sume kvadrata svih elemenata).
- Napisati funkciju koja izračunava gornju vandijagonalnu normu matrice (sumu apsolutnih vrednosti elemenata iznad glavne dijagonale).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju kvadratne matrice  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrice. Na standardni izlaz ispisati učitanu matricu a zatim trag, euklidsku normu i vandijagonalnu normu učitanе matrice.

## Test 1

```

Ulaz:  3 1 -2 3 4 -5 6 7 -8 9
Izlaz: 1 -2 3
        4 -5 6
        7 -8 9
        trag = 5
        euklidska norma = 16.88
        vandijagonalna norma = 11

```

## Test 2

```

Ulaz:  0
Izlaz: Greska: neodgovarajuca dimenzija matrice.

```

**Zadatak 2.9** Date su dve kvadratne matrice istih dimenzija  $n$ .

- (a) Napisati funkciju koja proverava da li su matrice jednake.
- (b) Napisati funkciju koja izračunava zbir matrica.
- (c) Napisati funkciju koja izračunava proizvod matrica.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju kvadratnih matrica  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrica. Na standardni izlaz ispisati „da“ ako su matrice jednake, „ne“ ako nisu a zatim ispisati zbir i proizvod učitanih matrica.

## Test 1

```

Ulaz:  3
        1 2 3 1 2 3 1 2 3
        1 2 3 1 2 3 1 2 3
Izlaz:  da
        Zbir matrica je:
        2 4 6
        2 4 6
        2 4 6
        Proizvod matrica je:
        6 12 18
        6 12 18
        6 12 18

```

**Zadatak 2.10** Relacija se može predstaviti kvadratnom matricom nula i jedinica na sledeći način: dva elementa  $i$  i  $j$  su u relaciji ukoliko se u preseku  $i$ -te vrste i  $j$ -te kolone matrice nalazi broj 1, a nisu u relaciji ukoliko se tu nalazi broj 0.

- (a) Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- (b) Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- (c) Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.

- (d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).
- (e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).
- (f) Napisati funkciju koja određuje refleksivno-tranzitivno zatvorenje relacije (najmanju refleksivnu i tranzitivnu relaciju koja sadrži datu)(Napomena: koristiti Varšalov algoritam).

Napisati program koji učitava matricu iz datoteke čije se ime zadaje kao prvi argument komandne linije. U prvoj liniji datoteke nalazi se dimenzija matrice  $n$  ( $0 < n \leq 64$ ), a potom i sami elementi matrice. Na standardni izlaz ispisati rezultat testiranja napisanih funkcija.

### Test 1

```
Poziv: ./a.out ulaz.txt
ulaz.txt: 4
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 0
Izlaz:    Refleksivnost: ne
          Simetricnost: ne
          Tranzitivnost: da
          Refleksivno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 1
          Simetricno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 1 1 0
          0 0 0 0
          Refleksivno-tranzitivno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 0
```

**Zadatak 2.11** Data je kvadratna matrica dimenzije  $n$ .

- (a) Napisati funkciju koja određuje najveći element matrice na sporednoj dijagonali.
- (b) Napisati funkciju koja određuje indeks kolone koja sadrži najmanji element matrice.
- (c) Napisati funkciju koja određuje indeks vrste koja sadrži najveći element matrice.

(d) Napisati funkciju koja određuje broj negativnih elemenata matrice.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati elemente celobrojne kvadratne matrice čija se dimenzija  $n$  ( $0 < n \leq 32$ ) zadaje kao argument komandne linije. Na standardni izlaz ispisati najveći element matrice na sporednoj dijagonali, indeks kolone koja sadrži najmanji element, indeks vrste koja sadrži najveći element i broj negativnih elemenata učitane matrice.

<i>Test 1</i>	<i>Test 2</i>
<pre> Poziv: ./a.out 3 Ulaz:  1 2 3         -4 -5 -6         7 8 9 Izlaz:  7 2 2 3           </pre>	<pre> Poziv: ./a.out 4 Ulaz:  -1 -2 -3 -4         -5 -6 -7 -8         -9 -10 -11 -12         -13 -14 -15 -16 Izlaz:  -4 3 0 16           </pre>
<i>Test 3</i>	
<pre> Poziv: ./a.out Izlaz: Greska: Nedovoljan broj argumenata komandne linije.         Program se poziva sa ./a.out dim_matrice.           </pre>	

**Zadatak 2.12** Napisati funkciju kojom se proverava da li je zadata kvadratna matrica dimenzije  $n$  ortonormirana. Matrica je ortonormirana ako je skalarni proizvod svakog para različitih vrsta jednak nuli, a skalarni proizvod vrste sa samom sobom jednak jedinici. Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju celobrojne kvadratne matrice  $n$  ( $0 < n \leq 32$ ), a zatim i njene elemente. Na standardni izlaz ispisati rezultat primene napisane funkcije na učitanoj matrici.

<i>Test 1</i>	<i>Test 2</i>
<pre> Ulaz:  4         1 0 0 0         0 1 0 0         0 0 1 0         0 0 0 1 Izlaz: da           </pre>	<pre> Ulaz:  3         1 2 3         5 6 7         1 4 2 Izlaz: ne           </pre>
<i>Test 3</i>	
<pre> Ulaz:  33 Izlaz: Greska: neodgovarajuca dimenzija matrice.           </pre>	

**Zadatak 2.13** Data je matrica dimenzije  $n \times m$ .

- (a) Napsiati funkciju koja učitava elemente matrice sa standardnog ulaza
- (b) Napsiati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenzije matrice  $n$  ( $0 < n \leq 10$ ) i  $m$  ( $0 < m \leq 10$ ), a zatim i elemente matrice

## 2 Pokazivači

---

(pozivom gore napisane funkcije). Na standardni izlaz spiralno ispisati elemente učitane matrice.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  3 3          1 2 3          4 5 6          7 8 9    Izlaz: 1 2 3 6 9 8 7 4 5</pre>	<pre>   Ulaz:  3 4          1 2 3 4          5 6 7 8          9 10 11 12    Izlaz: 1 2 3 4 8 12 11 10 9 5 6 7</pre>
<i>Test 3</i>	
<pre>   Ulaz:  11 4    Izlaz: Greska: neodgovarajuće dimenzije matrice.</pre>	

**Zadatak 2.14** Napisati funkciju koja izračunava  $k$ -ti stepen kvadratne matrice dimenzije  $n$  ( $0 < n \leq 32$ ). Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju celobrojne matrice  $n$ , elemente matrice i stepen  $k$  ( $0 < k \leq 10$ ). Na standardni izlaz ispisati rezultat primene napisane funkcije. Napomena: voditi računa da se prilikom stepenovanja matrice izvrši što manji broj množenja.

*Test 1*

```
|| Ulaz:  3
||       1 2 3
||       4 5 6
||       7 8 9
||       8
|| Izlaz: 510008400 626654232 743300064
||       1154967822 1419124617 1683281412
||       1799927244 2211595002 2623262760
```

## 2.3 Dinamička alokacija memorije

**Zadatak 2.15** Napisati program koji sa standardnog ulaza učitava dimenziju niza celih brojeva a zatim i njegove elemente. Ne praviti nikakve pretpostavke o dimenziji niza. Na standardni izlaz ispisati ove brojeve u obrnutom poretku.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  3          1 -2 3    Izlaz: 3 -2 1</pre>	<pre>   Ulaz:  -1    Izlaz: malloc(): neuspela alokacija memorije.</pre>

**Zadatak 2.16** Napisati program koji sa standardnog ulaza učitava niz celih brojeva. Brojevi se unose sve dok se ne unese nula. Ne praviti nikakve pretpostavke



o dimenziji niza. Na standardni izlaz ispisati ovaj niz brojeva u obrnutom poretku. Zadatak uraditi na dva načina:

- (a) realokaciju memorije niza vršiti korišćenjem `malloc()` funkcije,
- (b) realokaciju memorije niza vršiti korišćenjem `realloc()` funkcije.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  1 -2 3 -4 0    Izlaz: -4 3 -2 1</pre>	<pre>   Ulaz:  0    Izlaz:</pre>

**Zadatak 2.17** Napisati funkciju koja kao rezultat vraća nisku koja se dobija nadovezivanjem dve niske, bez promene njihovog sadržaja. Napisati program koji testira rad napisane funkcije. Sa standardnog ulaza učitati dve niske karaktera (pretpostaviti da niske nisu duže od 1000 karaktera i da ne sadrže praznine). Na standardni izlaz ispisati nisku koja se dobija njihovim nadovezivanjem. Za rezultujuću nisku dinamički alocirati memoriju.

<i>Test 1</i>
<pre>   Ulaz:  Jedan Dva    Izlaz: JedanDva</pre>

**Zadatak 2.18** Napisati program koji sa standardnog ulaza učitava matricu celih brojeva. Prvo se učitavaju dimenzije matrice  $n$  i  $m$  (ne praviti nikakve pretpostavke o njihovoj veličini), a zatim i elementi matrice. Na standardni izlaz ispisati trag matrice.

<i>Test 1</i>
<pre>   Ulaz:  2 3            1.2 2.3 3.4            4.5 5.6 6.7    Izlaz: 6.80</pre>

**Zadatak 2.19** Data je celobrojna matrica dimenzije  $n \times m$  napisati:

- (a) Napisati funkciju koja vrši učitavanje matrice sa standardnog ulaza.
- (b) Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati  $n$  i  $m$  (ne praviti nikakve pretpostavke o njihovoj veličini), zatim učitati elemente matrice i na standardni izlaz ispisati elemente ispod glavne dijagonale matrice.

### Test 1

```
Ulaz:  2 3
       1 -2 3
       -4 5 -6
Izlaz:  1
       -4 5
```

**Zadatak 2.20** Za zadatu matricu dimenzije  $n \times m$  napisati funkciju koja izračunava redni broj kolone matrice čiji je zbir maksimalan. Napisati program koji testira ovu funkciju. Sa standardnog ulaza učitati dimenzije matrice  $n$  i  $m$  (ne praviti nikakve pretpostavke o njihovoj veličini), a zatim elemente matrice. Na standardni izlaz ispisati redni broj kolone matrice sa maksimalnim zbirom.

### Test 1

```
Ulaz:  Unesite dimenzije matrice:
       2 3
       Unesite elemente matrice:
       1 2 3
       4 5 6
Izlaz: Kolona pod rednim brojem 3 ima najveći zbir.
```

**Zadatak 2.21** Data je kvadratna realna matrica dimenzije  $n$ .

- Napisati funkciju koja izračunava zbir apsolutnih vrednosti matrice ispod sporedne dijagonale.
- Napisati funkciju koja menja sadržaj matrice tako što polovi elemente iznad glavne dijagonale, duplira elemente ispod glavne dijagonale, dok elemente na glavnoj dijagonali ostavlja nepromenjene.

Napisati program koji testira ove funkcije za matricu koja se učitava iz datoteke čije se ime zadaje kao argument komandne linije. U datoteci se nalazi prvo dimenzija matrice, a zatim redom elementi matrice.

### Test 1

```
Poziv: ./a.out matrica.txt
matrica.txt:  3
              1.1 -2.2 3.3
              -4.4 5.5 -6.6
              7.7 -8.8 9.9
Izlaz: Zbir apsolutnih vrednosti ispod sporedne dijagonale je 25.30.
       Transformisana matrica je:
       1.10 -1.10 1.65
       -8.80 5.50 -3.30
       15.40 -17.60 9.90
```

**Zadatak 2.22** Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci „slicice.txt“ se nalaze informacije o sličicama koje mu nedostaju

u formatu: `redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`. Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronade ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. Napomena: za realokaciju memorije koristiti `realloc()` funkciju. **Jelena: treba dodati test primer.**

**Zadatak 2.23** U datoteci „temena.txt“ se nalaze tačke koje predstavljaju temena nekog  $n$ -tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom  $n$ -touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan. **Jelena: treba dodati test primer.**

**Zadatak 2.24** Napisati program koji na osnovu dve matrice dimenzija  $m \times n$  formira matricu dimenzije  $2 \cdot m \times n$  tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matrice su zapisane u datoteci „matrice.txt“. U prvom redu se nalaze dimenzije matrica  $m$  i  $n$ , u narednih  $m$  redova se nalaze vrste prve matrice, a u narednih  $m$  redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz. **Jelena: treba dodati test primer.**

**Zadatak 2.25** Na ulazu se zadaje niz celih brojeva čiji se unos završava nulom. Napisati funkciju koja od zadatog niza formira matricu tako da prva vrsta odgovara unetom nizu, a svaka naredna se dobija cikličkim pomeranjem elemenata niza za jednu poziciju ulevo. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz. **Jelena: treba dodati test primer.**

## 2.4 Pokazivači na funkcije

**Zadatak 2.26** Napisati program koji tabelarno štampa vrednosti proizvoljne realne funkcije sa jednim realnim argumentom, odnosno izračunava i ispisuje vrednosti date funkcije na diskretnoj ekvidistantnoj mreži od  $n$  tačaka intervala  $[a, b]$ . Realni brojevi  $a$  i  $b$  ( $a < b$ ) kao i ceo broj  $n$  ( $n \geq 2$ ) se učitavaju sa standardnog ulaza. Ime funkcije se zadaje kao argument komandne linije (`sin`, `cos`, `tan`, `atan`, `acos`, `asin`, `exp`, `log`, `log10`, `sqrt`, `floor`, `ceil`, `sqr`).

## Test 1

```
Poziv: ./a.out sin
Ulaz: Unesite krajeve intervala:
      -0.5 1
      Koliko tacaka ima na ekvidistantnoj mrezi (ukljucujuci krajeve
      intervala)?
      4
Izlaz:      x      sin(x)
      -----
      | -0.50000 | -0.47943 |
      |  0.00000 |  0.00000 |
      |  0.50000 |  0.47943 |
      |  1.00000 |  0.84147 |
      -----
```

## Test 2

```
Poziv: ./a.out cos
Ulaz: Unesite krajeve intervala:
      0 2
      Koliko tacaka ima na ekvidistantnoj mrezi (ukljucujuci krajeve
      intervala)?
      4
Izlaz:      x      cos(x)
      -----
      |  0.00000 |  1.00000 |
      |  0.66667 |  0.78589 |
      |  1.33333 |  0.23524 |
      |  2.00000 | -0.41615 |
      -----
```

**Zadatak 2.27** Napisati funkciju koja izračunava limes funkcije  $f(x)$  u tački  $a$ . Adresa funkcije  $f$  čiji se limes računa se prenosi kao parametar funkciji za računanje limesa. Limes se računa sledećom aproksimacijom (vrednosti  $n$  i  $a$  uneti sa standardnog ulaza kao i ime funkcije):

$$\lim_{x \rightarrow a} f(x) = \lim_{n \rightarrow \infty} f(a + \frac{1}{n})$$

## Test 1

```
Ulaz:  tan 1.570795 10000
Izlaz: -10134.5
```

## Test 2

```
Ulaz:  log 0 1000000
Izlaz: -13.81551
```

**Zadatak 2.28** Napisati funkciju koja određuje integral funkcije  $f(x)$  na intervalu  $[a, b]$ . Adresa funkcije  $f$  se prenosi kao parametar. Integral se računa prema formuli:

$$\int_a^b f(x) = h \cdot \left( \frac{f(a) + f(b)}{2} + \sum_{i=1}^n f(a + i \cdot h) \right)$$

Vrednost  $h$  se izračunava po formuli  $h = (b - a)/n$ , dok se vrednosti  $n$ ,  $a$  i  $b$  unose sa standardnog ulaza kao i ime funkcije iz zaglavlja `math.h`. Na standardni izlaz

ispisati vrednost integrala. **Jelena: treba dodati test primer.**

**Zadatak 2.29** Napisati funkciju koja približno izračunava integral funkcije  $f(x)$  na intervalu  $[a, b]$ . Funkcija  $f$  se prosleđuje kao parametar, a integral se procenjuje po Simpsonovoj formuli:

$$I = \frac{h}{3} \left( f(a) + 4 \sum_{i=1}^{n/2} f(a + (2i-1)h) + 2 \sum_{i=1}^{n/2-1} f(a + 2ih) + f(b) \right)$$

Granice intervala i  $n$  su argumenti funkcije. Napisati program, koji kao argumente komandne linije prihvata ime funkcije iz zaglavlja `math.h`, krajeve intervala pretrage i  $n$ , a na standardni izlaz ispisuje vrednost odgovarajućeg integrala. **Jelena: treba dodati test primer.**

## 2.5 Rešenja

### Rešenje 2.1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija obrće elemente niza koriscenjem indekse sintakse */
7 void obrni_niz_v1(int a[] , int n)
8 {
9     int i, j;
10
11     for(i = 0, j = n-1; i < j; i++, j--) {
12         int t = a[i];
13         a[i] = a[j];
14         a[j] = t;
15     }
16 }
17
18 /* Funkcija obrće elemente niza koriscenjem pokazivacke
19    sintakse. Umesto "void obrni_niz(int *a, int n)" potpis
20    metode bi mogao da bude i "void obrni_niz(int a[], int n)".
21    U oba slucaja se argument funkcije "a" tumaci kao pokazivac,
22    ili tacnije, kao adresa prvog elementa niza. U odnosu na
23    njega se odredjuju adrese ostalih elemenata u nizu */
24 void obrni_niz_v2(int *a, int n)
25 {
26     /* Pokazivaci na elemente niza a */
27     int *prvi, *poslednji;
28
29     for(prvi = a, poslednji = a + n - 1;
30         prvi < poslednji; prvi++, poslednji--) {
31         int t = *prvi;
32         *prvi = *poslednji;

```

```
34     *poslednji = t;
35 }
36 }

38 /* Funkcija obrće elemente niza koriscenjem pokazivacke
   sintakse - modifikovano koriscenje pokazivaca */
40 void obrni_niz_v3(int *a, int n)
41 {
42     /* Pokazivaci na elemente niza a */
43     int *prvi, *poslednji;
44
45     /* Obrćemo niz */
46     for(prvi = a, poslednji = a + n - 1; prvi < poslednji; ) {
47         int t = *prvi;
48
49         /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
50          vrednost koja se nalazi na adresi na koju pokazuje
51          pokazivac "poslednji". Nakon toga se pokazivac "prvi"
52          uvecava za jedan sto za posledicu ima da "prvi" pokazuje
53          na sledeci element u nizu */
54         *prvi++ = *poslednji;
55
56         /* Vrednost promenljive "t" se postavlja na adresu na koju
57          pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
58          umanjuje za jedan, sto za posledicu ima da pokazivac
59          "poslednji" sada pokazuje na element koji mu prethodi u
60          nizu */
61         *poslednji-- = t;
62     }
63 }
64
65 int main()
66 {
67     /* Deklaracija niza a od najvise MAX elemenata */
68     int a[MAX];
69
70     /* Broj elemenata niza a */
71     int n;
72
73     /* Pokazivac na elemente niza a */
74     int *p;
75
76     /* Unosimo dimenziju niza */
77     scanf("%d", &n);
78
79     /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
80     if(n <= 0 || n > MAX) {
81         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
82         exit(EXIT_FAILURE);
83     }
84
85     /* Unosimo elemente niza */
86     for(p = a; p - a < n; p++)
87         scanf("%d", p);
88
89     obrni_niz_v1(a,n);
```

```

90 // obrni_niz_v2(a,n);
91 // obrni_niz_v3(a,n);
92
93 /* Prikazujemo sadržaj niza nakon obrtanja */
94 for(p = a; p - a < n; p++)
95     printf("%d ", *p);
96 printf("\n");
97
98 return 0;
99 }

```

## Rešenje 2.2

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija racuna zbir elemenata niza */
7 double zbir(double *a, int n)
8 {
9     double s = 0;
10    int i;
11
12    for(i = 0; i < n; s += a[i++]) ;
13
14    return s;
15 }
16
17 /* Funkcija racuna proizvod elemenata niza */
18 double proizvod(double a[], int n)
19 {
20    double p = 1;
21
22    for(; n; n--)
23        p *= *a++;
24
25    return p;
26 }
27
28 /* Funkcija racuna najmanji element niza */
29 double min(double *a, int n)
30 {
31     /* Za najmanji element se najpre postavlja prvi element */
32     double min = a[0];
33     int i;
34
35     /* Ispitujemo da li se medju ostalim elementima niza
36        nalazi najmanji */
37     for(i = 1; i < n; i++)
38         if ( a[i] < min )
39             min = a[i];
40
41     return min;
42 }

```

```
44 /* Funkcija racuna najveći element niza */
double max(double *a, int n)
46 {
    /* Za najveći element se najpre postavlja prvi element */
48     double max = *a;

50     /* Ispitujemo da li se među ostalim elementima niza
        nalazi najveći */
52     for(a++, n--; n > 0; a++, n--)
        if (*a > max)
54             max = *a;

56     return max;
}
58

60 int main()
{
62     double a[MAX];
    int n, i;

64     /* Učitavamo dimenziju niza */
66     scanf("%d", &n);

68     /* Proveravamo da li je prekoraceno ograničenje dimenzije */
    if(n <= 0 || n > MAX) {
70         fprintf(stderr, "Greska: neodgovarajuća dimenzija niza.\n");
        exit(EXIT_FAILURE);
72     }

74     /* Unosimo elemente niza */
    for(i = 0; i < n; i++)
76         scanf("%lf", a + i);

78     /* Testiramo definisane funkcije */
    printf("zbir = %5.3f\n", zbir(a, n));
80     printf("proizvod = %5.3f\n", proizvod(a, n));
    printf("min = %5.3f\n", min(a, n));
82     printf("max = %5.3f\n", max(a, n));

84     return 0;
}
```

### Rešenje 2.3

```
#include <stdio.h>
2 #include <stdlib.h>
#define MAX 100

4 /* Funkcija povećava za jedan sve elemente u prvoj polovini niza
6     a smanjuje za jedan sve elemente u drugoj polovini niza.
    Ukoliko niz ima neparan broj elemenata, srednji element
8     ostaje nepromenjen */
void povecaj_smanji (int *a , int n)
```



```
10 {
11     int *prvi = a;
12     int *poslednji = a+n-1;
13
14     while( prvi < poslednji ){
15
16         /* Povecava se vrednost elementa na koji pokazuje
17            pokazivac prvi */
18         (*prvi)++;
19
20         /* Pokazivac prvi se pomera na sledeci element */
21         prvi++;
22
23         /* Smanjuje se vrednost elementa na koji pokazuje
24            pokazivac poslednji */
25         (*poslednji)--;
26
27         /* Pokazivac poslednji se pomera na prethodni element */
28         poslednji--;
29     }
30 }
31
32 void povecaj_smanji_sazetije(int *a , int n)
33 {
34     int *prvi = a;
35     int *poslednji = a+n-1;
36
37     while( prvi < poslednji ){
38         (*prvi++)++;
39         (*poslednji--)--;
40     }
41 }
42
43 int main()
44 {
45     int a[MAX];
46     int n;
47     int *p;
48
49     /* Unosimo broj elemenata */
50     scanf("%d", &n);
51
52     /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
53     if(n <= 0 || n > MAX) {
54         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
55         exit(EXIT_FAILURE);
56     }
57
58     /* Unosimo elemente niza */
59     for(p = a; p - a < n; p++)
60         scanf("%d", p);
61
62     povecaj_smanji(a,n);
63     /* povecaj_smanji_sazetije(a,n); */
64
65     /* Prikaz niza nakon modifikacije */
```

## 2 Pokazivači

```
66     for(p = a; p - a < n; p++)
        printf("%d ", *p);
68     printf("\n");
70     return 0;
}
```

### Rešenje 2.4

```
#include <stdio.h>
2
/* Argumenti funkcije main mogu da budu broj argumenta komandne
4   linije (int argc) i niz argumenata komandne linije
   (niz niski) (char *argv[] <=> char** argv) */
6 int main(int argc, char *argv[])
{
8     int i;

10    /* Ispisujemo broj argumenata komandne linije */
    printf("%d\n", argc);
12
    /* Ispisujemo argumente komandne linije */
    /* koristeći indeksnu sintaksu */
14    for(i=0; i<argc; i++) {
16        printf("%d %s\n", i, argv[i]);
    }
18
    /* koristeći pokazivačku sintaksu */
20    i=argc;
    for (; argc>0; argc--)
22        printf("%d %s\n", i-argc, *argv++);
24
    /* Nakon ove petlje "argc" će biti jednako nuli a "argv" će
26    pokazivati na polje u memoriji koje se nalazi iza
    poslednjeg argumenta komandne linije. Kako smo u
28    promenljivoj "i" sacuvali vrednost broja argumenta
    komandne linije to sada možemo ponovo da postavimo
30    "argv" da pokazuje na nulti argument komandne linije */
    argv = argv - i;
32    argc = i;
34    /* Ispisujemo 0-ti karakter svakog od argumenata komandne linije
    */

36    /* koristeći indeksnu sintaksu */
    for(i=0; i<argc; i++)
38        printf("%c ", argv[i][0]);
    printf("\n");
40
    /* koristeći pokazivačku sintaksu */
42
    for (i=0 ; i<argc; i++ )
44        printf("%c ", **argv++);
```

```
46     return 0;
    }
```

### Rešenje 2.5

```
1  #include<stdio.h>
2  #include<string.h>
3  #define MAX 100
4
5  /* Funkcija ispituje da li je niska palindrom */
6  int palindrom(char *niska)
7  {
8      int i, j;
9      for(i = 0, j = strlen(niska)-1; i < j; i++, j--)
10         if(*(niska+i) != *(niska+j))
11             return 0;
12     return 1;
13 }
14
15 int main(int argc, char **argv)
16 {
17     int i, n = 0;
18
19     /* Multi argument komandne linije je ime izvrsnog programa */
20     for(i = 1; i < argc; i++)
21         if(palindrom(*(argv+i)))
22             n++;
23
24     printf("%d\n", n);
25     return 0;
26 }
```

### Rešenje 2.6

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Funkcija strlen() iz standardne biblioteke */
7  int duzina(char *s)
8  {
9      int i;
10     for(i = 0; *(s+i); i++)
11         ;
12     return i;
13 }
14
15 int main(int argc, char **argv)
16 {
17     char rec[MAX_KARAKTERA];
18     int br = 0, i = 0, n;
19     FILE *in;
```

```
21  /* Ako korisnik nije uneo trazene argumente,
    prijavljujemo gresku */
23  if(argc < 3) {
    printf("Greska: ");
25  printf("Nedovoljan broj argumenata komandne linije.\n");
    printf("Program se poziva sa %s ime_dat br_karaktera.\n",
27                                     argv[0]);
    exit(EXIT_FAILURE);
29  }

31  /* Otvaramo datoteku sa imenom koje se zadaje kao prvi
    argument komandne linije. */
33  in = fopen(*(argv+1), "r");
    if(in == NULL){
35      fprintf(stderr, "Greska: ");
      fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
37                                     argv[1]);
      exit(EXIT_FAILURE);
39  }

41  n = atoi(*(argv+2));

43  /* Broje se reci cija je duzina jednaka broju zadatom drugim
    argumentom komandne linije */
45  while(fscanf(in, "%s", rec) != EOF)
    if(duzina(rec) == n)
47      br++;

49  printf("%d\n", br);

51  /* Zatvaramo datoteku */
    fclose(in);
53  return 0;
}
```

### Rešenje 2.7

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

3

    #define MAX_KARAKTERA 100

5
    /* Funkcija strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
    {
9      int i;
      for (i = 0; *(src+i); i++)
11         *(dest+i) = *(src+i);
    }

13
    /* Funkcija strcmp() iz standardne biblioteke */
15  int poredjenje_niski(char *s, char *t)
    {
17      int i;
      for (i = 0; *(s+i) == *(t+i); i++)
```

```

19     if(*(s+i) == '\0')
20         return 0;
21     return *(s+i) - *(t+i);
22 }
23
24 /* Funkcija strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
26 {
27     int i;
28     for(i = 0; *(s+i); i++)
29         ;
30     return i;
31 }
32
33 /* Funkcija ispituje da li je niska zadata drugim argumentom
34    funkcije sufiks niske zadate prvi argumentom funkcije */
35 int sufiks_niske(char *niska, char *sufiks) {
36     if(duzina_niske(sufiks) <= duzina_niske(niska) &&
37        poredjenje_niski(niska + duzina_niske(niska) -
38                        duzina_niske(sufiks), sufiks) == 0)
39         return 1;
40     return 0;
41 }
42
43 /* Funkcija ispituje da li je niska zadata drugim argumentom
44    funkcije prefiks niske zadate prvi argumentom funkcije */
45 int prefiks_niske(char *niska, char *prefiks) {
46     int i;
47     if(duzina_niske(prefiks) <= duzina_niske(niska)) {
48         for(i=0; i<duzina_niske(prefiks); i++)
49             if(*(prefiks+i) != *(niska+i))
50                 return 0;
51         return 1;
52     }
53     else return 0;
54 }
55
56 int main(int argc, char **argv)
57 {
58     /* Ako korisnik nije uneo trazene argumente,
59        prijavljujemo gresku */
60     if(argc < 4) {
61         printf("Greska: ");
62         printf("Nedovoljan broj argumenata komandne linije.\n");
63         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
64                argv[0]);
65         exit(EXIT_FAILURE);
66     }
67
68     FILE *in;
69     int br = 0, i = 0, n;
70     char rec[MAX_KARAKTERA];
71
72     in = fopen(*(argv+1), "r");
73     if(in == NULL) {
74         fprintf(stderr, "Greska: ");

```

```
75     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
76                                     argv[1]);
77     exit(EXIT_FAILURE);
78 }
79
80 /* Proveravamo kojom opcijom je pozvan program a zatim
81    učitavamo reci iz datoteke brojimo koliko reci
82    zadovoljava trazeni uslov */
83 if(!(poredjenje_niski(*(argv + 3), "-s")))
84     while(fscanf(in, "%s", rec) != EOF)
85         br += sufiks_niske(rec, *(argv+2));
86 else if (!(poredjenje_niski(*(argv+3), "-p")))
87     while(fscanf(in, "%s", rec) != EOF)
88         br += prefiks_niske(rec, *(argv+2));
89
90 printf("%d\n", br);
91 fclose(in);
92 return 0;
93 }
```

### Rešenje 2.8

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

#define MAX 100

/* Deklarisemo funkcije koje cemo kasnije da definisemo */
double euklidska_norma( int M[][MAX], int n);
int trag(int M[][MAX], int n);
int gornja_vandijagonalna_norma(int M[][MAX], int n);

int main()
{
    int A[MAX][MAX];
    int i,j,n;

    /* Unosimo dimenziju kvadratne matrice */
    scanf("%d",&n);

    /* Proveravamo da li je prekoraceno ogranicenje */
    if( n > MAX || n <= 0) {
        fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
        fprintf(stderr, "matrice.\n");
        exit(EXIT_FAILURE);
    }

    /* Popunjavamo vrstu po vrstu matrice */
    for(i = 0; i<n; i++)
        for (j=0 ; j<n; j++)
            scanf("%d",&A[i][j]);

    /* Ispis elemenata matrice koriscenjem indeksne sintakse.
       Ispis vrsimo vrstu po vrstu */
}
```

```

34  for(i = 0; i<n; i++) {
    /* Ispisujemo elemente i-te vrste */
36  for ( j=0 ; j<n; j++)
    printf("%d ", A[i][j]);
38  printf("\n");
    }

40
    /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
42     Kod ovako definisane matrice, elementi su uzastopno
    smesteni u memoriju, kao na traci. To znaci da su svi
44     elementi prve vrste redom smesteni jedan iza drugog. Odmah
    iza poslednjeg elementa prve vrste smesten je prvi element
46     druge vrste za kojim slede svi elementi te vrste
    i tako dalje redom */
48  /*
    for( i = 0; i<n; i++) {
50      for ( j=0 ; j<n; j++)
        printf("%d ", *(A+i+j));
52      printf("\n");
    }
54  */

56  int tr = trag(A,n);
    printf("trag = %d\n",tr);

58
    printf("euklidska norma = %.2f\n",euklidska_norma(A,n));
60  printf ("vandijagonalna norma = %d\n",
        gornja_vandijagonalna_norma(A,n));

62
    return 0;
64 }

66 /* Definisemo funkcije koju smo ranije deklarirali */

68 /* Funkcija izracunava trag matrice */
int trag(int M[][MAX], int n)
70 {
    int trag = 0,i;
72  for(i=0; i<n; i++)
    trag += M[i][i];
74  return trag;
    }

76
    /* Funkcija izracunava euklidsku normu matrice */
78 double euklidska_norma(int M[][MAX], int n)
    {
80  double norma = 0.0;
    int i,j;

82
    for(i= 0; i<n; i++)
84      for(j = 0; j<n; j++)
        norma += M[i][j] * M[i][j];

86
    return sqrt(norma);
88 }

```

```
90 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
91 int gornja_vandijagonalna_norma(int M[][MAX], int n)
92 {
93     int norma =0;
94     int i,j;
95
96     for(i=0 ;i<n; i++) {
97         for(j = i+1; j<n; j++)
98             norma += abs(M[i][j]);
99     }
100
101     return norma;
102 }
```

### Rešenje 2.9

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

#define MAX 100

/* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
standardnog ulaza */
void ucitaj_matricu(int m[][MAX], int n)
{
    int i, j;

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

/* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
standardni izlaz */
void ispisi_matricu(int m[][MAX], int n) {
    int i, j;

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

/* Funkcija proverava da li su zadate kvadratne matrice a i b
dimenzije n jednake */
int jednake_matrice(int a[][MAX], int b[][MAX], int n) {
    int i, j;

    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            /* Nasli smo elemente na istim pozicijama u matricama
            koji se razlikuju */
            if(a[i][j]!=b[i][j])
                return 0;
}
```



```
40
41     /* Prosla je provera jednakosti za sve parove elemenata koji
42        su na istim pozicijama sto znaci da su matrice jednake */
43     return 1;
44 }
45
46 /* Funkcija izracunava zbir dve kvadratne matice */
47 void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
48 {
49     int i, j;
50
51     for(i=0; i<n; i++)
52         for(j=0; j<n; j++)
53             c[i][j] = a[i][j] + b[i][j];
54 }
55
56 /* Funkcija izracunava proizvod dve kvadratne matice */
57 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
58 {
59     int i, j, k;
60
61     for(i=0; i<n; i++)
62         for(j=0; j<n; j++) {
63             /* Mnozimo i-tu vrstu prve sa j-tom kolonom druge matrice */
64             c[i][j] = 0;
65             for(k=0; k<n; k++)
66                 c[i][j] += a[i][k] * b[k][j];
67         }
68 }
69
70 int main()
71 {
72     /* Matrice ciji se elementi zadaju sa ulaza */
73     int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
74
75     /* Matrice zbira i proizvoda */
76     int zbir[MAX][MAX], proizvod[MAX][MAX];
77
78     /* Dimenzija matrica */
79     int n;
80     int i, j;
81
82     /* Ucitavamo dimenziju kvadratnih matrica i proveravamo njenu
83        korektnost */
84     scanf("%d", &n);
85
86     /* Proveravamo da li je prekoraceno ogranicenje */
87     if( n > MAX || n <= 0) {
88         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
89         fprintf(stderr, "matrica.\n");
90         exit(EXIT_FAILURE);
91     }
92
93     /* Ucitavamo matrice */
94     ucitaj_matricu(a, n);
95     ucitaj_matricu(b, n);
```

```
96      /* Izracunavamo zbir i proizvod matrica */
97      saberi(a, b, zbir, n);
98      pomnozi(a, b, proizvod, n);
100
101      /* Ispisujemo rezultat */
102      if(jednake_matrice(a, b, n) == 1)
103          printf("da\n");
104      else
105          printf("ne\n");
106
107      printf("Zbir matrica je:\n");
108      ispisi_matricu(zbir, n);
109
110      printf("Proizvod matrica je:\n");
111      ispisi_matricu(proizvod, n);
112
113      return 0;
114 }
```

### Rešenje 2.10

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

#define MAX 64

/* Funkcija proverava da li je relacija refleksivna. Relacija je
   refleksivna ako je svaki element u relaciji sam sa sobom,
   odnosno ako se u matrici relacije na glavnoj dijagonali
   nalaze jedinice */
int refleksivnost(int m[][MAX], int n)
{
    int i;

    /* Obilazimo glavnu dijagonalu matrice. Za elemente na glavnoj
       dijagonali vazi da je indeks vrste jednak indeksu kolone */
    for(i=0; i<n; i++) {
        if(m[i][i] != 1)
            return 0;
    }

    return 1;
}

/* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono
   je odredjeno matricom koja sadrzi sve elemente polazne matrice
   dopunjene jedinicama na glavnoj dijagonali */
void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
{
    int i, j;

    /* Prepisujemo vrednosti elemenata matrice pocetne matrice */
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
```

```

34     zatvorenje[i][j] = m[i][j];

36     /* Postavljamo na glavnoj dijagonali jedinice */
37     for(i=0; i<n; i++)
38         zatvorenje[i][i] = 1;
39 }

40 /* Funkcija proverava da li je relacija simetricna. Relacija je
41    simetricna ako za svaki par elemenata vazi: ako je element
42    "i" u relaciji sa elementom "j", onda je i element "j" u
43    relaciji sa elementom "i". Ovakve matrice su simetricne u
44    odnosu na glavnu dijagonalu */
45 int simetricnost (int m[][MAX], int n)
46 {
47     int i, j;

48     /* Obilazimo elemente ispod glavne dijagonale matrice i
49        uporedjujemo ih sa njima simetricnim elementima */
50     for(i=0; i<n; i++)
51         for(j=0; j<i; j++)
52             if(m[i][j] != m[j][i])
53                 return 0;

54     return 1;
55 }

56 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono
57    je odredjeno matricom koja sadrzi sve elemente polazne matrice
58    dopunjene tako da matrica postane simetricna u odnosu na
59    glavnu dijagonalu */
60 void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
61 {
62     int i, j;

63     /* Prepisujemo vrednosti elemenata matrice m */
64     for(i=0; i<n; i++)
65         for(j=0; j<n; j++)
66             zatvorenje[i][j] = m[i][j];

67     /* Odredjujemo simetricno zatvorenje matrice */
68     for(i=0; i<n; i++)
69         for(j=0; j<n; j++)
70             if(zatvorenje[i][j] == 1)
71                 zatvorenje[j][i] = 1;
72 }

73 /* Funkcija proverava da li je relacija tranzitivna. Relacija je
74    tranzitivna ako ispunjava sledece svojstvo: ako je element "i"
75    u relaciji sa elementom "j" i element "j" u relaciji sa
76    elementom "k", onda je i element "i" u relaciji sa elementom
77    "k" */
78 int tranzitivnost (int m[][MAX], int n)
79 {
80     int i, j, k;

```

```
90     for(i=0; i<n; i++)
91         for(j=0; j<n; j++)
92             /* Pokusavamo da pronadjemo element koji narusava
93              * tranzitivnost */
94             for(k=0; k<n; k++)
95                 if(m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
96                     return 0;
97
98     return 1;
99 }
100
101 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje
102    zadate relacije koriscenjem Varsalovog algoritma */
103 void tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
104 {
105     int i, j, k;
106
107     /* Kopiramo pocetnu matricu u matricu rezultata */
108     for(i=0; i<n; i++)
109         for(j=0; j<n; j++)
110             zatvorenje[i][j] = m[i][j];
111
112     /* Primenom Varsalovog algoritma odredjujemo
113        refleksivno-tranzitivno zatvorenje matrice */
114     for(k=0; k<n; k++)
115         for(i=0; i<n; i++)
116             for(j=0; j<n; j++)
117                 if((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
118                    && (zatvorenje[i][j] == 0))
119                     zatvorenje[i][j] = 1;
120 }
121
122 /* Funkcija ispisuje elemente matrice */
123 void pisi_matricu(int m[][MAX], int n)
124 {
125     int i, j;
126
127     for(i=0; i<n; i++) {
128         for(j=0; j<n; j++)
129             printf("%d ", m[i][j]);
130         printf("\n");
131     }
132 }
133
134 int main(int argc, char* argv[])
135 {
136     FILE* ulaz;
137     int m[MAX][MAX];
138     int pomocna[MAX][MAX];
139     int n, i, j, k;
140
141     /* Ako korisnik nije uneo trazene argumente,
142        prijavljujemo gresku */
143     if(argc < 2) {
144         printf("Greska: ");
```

```
146 printf("Nedovoljan broj argumenata komandne linije.\n");
147 printf("Program se poziva sa %s ime_dat.\n", argv[0]);
148 exit(EXIT_FAILURE);
149 }
150
151 /* Otvaramo datoteku za citanje */
152 ulaz = fopen(argv[1], "r");
153 if(ulaz == NULL) {
154     fprintf(stderr, "Greska: ");
155     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
156             argv[1]);
157     exit(EXIT_FAILURE);
158 }
159
160 /* Ucitavamo dimenziju matrice */
161 fscanf(ulaz, "%d", &n);
162
163 /* Proveravamo da li je prekoraceno ogranicenje */
164 if( n > MAX || n <= 0) {
165     fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
166     fprintf(stderr, "matrice.\n");
167     exit(EXIT_FAILURE);
168 }
169
170 /* Ucitavamo element po element matrice */
171 for(i=0; i<n; i++)
172     for(j=0; j<n; j++)
173         fscanf(ulaz, "%d", &m[i][j]);
174
175 /* Ispisujemo trazene vrednosti */
176 printf("Refleksivnost: %s\n",
177        refleksivnost(m, n) == 1 ? "da" : "ne");
178
179 printf("Simetricnost: %s\n",
180        simetricnost(m, n) == 1 ? "da" : "ne");
181
182 printf("Tranzitivnost: %s\n",
183        tranzitivnost(m, n) == 1 ? "da" : "ne");
184
185 printf("Refleksivno zatvorenje:\n");
186 ref_zatvorenje(m, n, pomocna);
187 pisi_matricu(pomocna, n);
188
189 printf("Simetricno zatvorenje:\n");
190 sim_zatvorenje(m, n, pomocna);
191 pisi_matricu(pomocna, n);
192
193 printf("Refleksivno-tranzitivno zatvorenje:\n");
194 tran_zatvorenje(m, n, pomocna);
195 pisi_matricu(pomocna, n);
196
197 /* Zatvaramo datoteku */
198 fclose(ulaz);
199
200 return 0;
201 }
```

### Rešenje 2.11

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 32
5
6  int max_sporedna_dijagonala(int m[][MAX], int n)
7  {
8      int i, j;
9      /* Trazimo najveći element na sporednoj dijagonali. Za
10         elemente sporedne dijagonale vazi da je zbir indeksa vrste
11         i indeksa kolone jednak n-1. Za početnu vrednost maksimuma
12         uzimamo element u gornjem desnom uglu */
13     int max_na_sporednoj_dijagonali = m[0][n-1];
14     for(i=1; i<n; i++)
15         if(m[i][n-1-i] > max_na_sporednoj_dijagonali)
16             max_na_sporednoj_dijagonali = m[i][n-1-i];
17
18     return max_na_sporednoj_dijagonali;
19 }
20
21 /* Funkcija izracunava indeks kolone najmanjeg elementa */
22 int indeks_min(int m[][MAX], int n)
23 {
24     int i, j;
25     /* Za početnu vrednost minimuma uzimamo element u gornjem
26        levom uglu */
27     int min=m[0][0], indeks_kolone=0;
28
29     for(i=0; i<n; i++)
30         for(j=0; j<n; j++)
31             /* Ako je tekuci element manji od minimalnog */
32             if(m[i][j]<min) {
33                 /* cuvamo njegovu vrednost */
34                 min=m[i][j];
35                 /* i cuvamo indeks kolone u kojoj se nalazi */
36                 indeks_kolone=j;
37             }
38     return indeks_kolone;
39 }
40
41 /* Funkcija izracunava indeks vrste najveceg elementa */
42 int indeks_max(int m[][MAX], int n) {
43     int i, j;
44     /* Za maksimalni element uzimamo gornji levi ugao */
45     int max=m[0][0], indeks_vrste=0;
46
47     for(i=0; i<n; i++)
48         for(j=0; j<n; j++)
49             /* Ako je tekuci element manji od minimalnog */
50             if(m[i][j]>max) {
51                 /* cuvamo njegovu vrednost */
```

```

        max=m[i][j];
53     /* i cuvamo indeks vrste u kojoj se nalazi */
        indeks_vrste=i;
55     }
    return indeks_vrste;
57 }

59 /* Funkcija izracunava broj negativnih elemenata matrice */
int broj_negativnih(int m[][MAX], int n) {
61     int i, j;

63     int broj_negativnih=0;

65     for(i=0; i<n; i++)
        for(j=0; j<n; j++)
67         if(m[i][j]<0)
            broj_negativnih++;
69     return broj_negativnih;
}

71
int main(int argc, char* argv[])
73 {
    int m[MAX][MAX];
75     int n;
    int i, j;

77     /* Proveravamo broj argumenata komandne linije */
79     if(argc < 2) {
        printf("Greska: ");
81     printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
83     exit(EXIT_FAILURE);
    }

85     /* Ucitavamo vrednost dimenzije i proveravamo njenu
87     korektnost */
    n = atoi(argv[1]);

89     if( n > MAX || n <= 0) {
91         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
        fprintf(stderr, "matrice.\n");
93         exit(EXIT_FAILURE);
    }

95     /* Ucitavamo element po element matrice */
97     for(i=0; i<n; i++)
        for(j=0; j<n; j++)
99         scanf("%d", &m[i][j]);

101     int max_sd = max_sporedna_dijagonala(m, n);
    int i_min = indeks_min(m, n);
103     int i_max = indeks_max(m, n);
    int bn = broj_negativnih(m, n);

105     /* Ispisujemo rezultat */
107     printf("%d %d %d %d\n", max_sd, i_min, i_max, bn);

```

```
109  /* Prekidamo izvršavanje programa */
      return 0;
111 }
```

### Rešenje 2.12

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

4  #define MAX 32

6  /* Funkcija učitava elemente kvadratne matrice sa
   standardnog ulaza */
8  void ucitaj_matricu(int m[][MAX], int n)
   {
10     int i, j;

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

16
18  /* Funkcija ispisuje elemente kvadratne matrice na
   standardni izlaz */
20  void ispsi_matricu(int m[][MAX], int n)
   {
22     int i, j;

24     for(i=0; i<n; i++) {
         for(j=0; j<n; j++)
26             printf("%d ", m[i][j]);
         printf("\n");
28     }

30  /* Funkcija proverava da li je zadata matrica ortonormirana */
32  int ortonormirana(int m[][MAX], int n)
   {
34     int i, j, k;
     int proizvod;

36     /* Proveravamo uslov normiranosti, odnosno da li je proizvod
       svake vrste matrice sa samom sobom jednak jedinici */
38     for(i=0; i<n; i++) {

40         /* Izracunavamo skalarni proizvod vrste sa samom sobom */
         proizvod = 0;

42         for(j=0; j<n; j++)
44             proizvod += m[i][j]*m[i][j];

46         /* Ako proizvod bar jedne vrste nije jednak jedinici, odmah
           zakljucujemo da matrica nije normirana */
48         if(proizvod!=1)
```



```

    return 0;
50 }

52 /* Proveravamo uslov ortogonalnosti, odnosno da li je proizvod
    dve bilo koje razlicite vrste matrice jednak nuli */
54 for(i=0; i<n-1; i++) {
    for(j=i+1; j<n; j++) {
56
58         /* Izracunavamo skalarni proizvod */
        proizvod = 0;

60         for(k=0; k<n; k++)
            proizvod += m[i][k] * m[j][k];

62         /* Ako proizvod dve bilo koje razlicite vrste nije jednak
64            nuli, odmah zakljucujemo da matrica nije ortogonalna */
            if(proizvod!=0)
66                 return 0;
        }
68     }

70 /* Ako su oba uslova ispunjena, vracamo jedinicu kao
    rezultat */
72 return 1;
}

74
76 int main()
77 {
    int A[MAX][MAX];
78     int n;

80     /* Ucitavamo vrednost dimenzije i proveravamo njenu
        korektnost */
82     scanf("%d", &n);

84     if( n > MAX || n <= 0) {
        fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
86         fprintf(stderr, "matrice.\n");
        exit(EXIT_FAILURE);
88     }

90     /* Ucitavamo matricu */
    ucitaj_matricu(A, n);

92
94     /* Ispisujemo rezultat rada funkcije */
    if(ortonormirana(A,n))
        printf("da\n");
96     else
        printf("ne\n");
98
    return 0;
100 }

```

## Rešenje 2.13

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_V 10
5  #define MAX_K 10
6
7  /* Funkcija proverava da li su ispisani svi elementi iz matrice,
8     odnosno da li se nariusio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
10 {
11     return !(top <= bottom && left <= right);
12 }
13
14 /* Funkcija spiralno ispisuje elemente matrice */
15 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16 {
17     int i,j,top, bottom,left, right;
18
19     top=left = 0;
20     bottom=n-1;
21     right = m-1;
22
23     while( !krajIspisa(top, bottom, left, right) ) {
24         /* Ispisuje se prvi red*/
25         for(j=left; j<=right; j++)
26             printf("%d ",a[top][j]);
27
28         /* Spustamo prvi red */
29         top++;
30
31         if(krajIspisa(top,bottom,left,right))
32             break;
33
34         for(i=top; i<=bottom; i++ )
35             printf("%d ",a[i][right]);
36
37         /* Pomeramo desnu kolonu za naredni krug ispisa
38            blize levom kraju */
39         right--;
40
41         if(krajIspisa(top,bottom,left,right))
42             break;
43
44         /* Ispisujemo donju vrstu */
45         for(j=right; j>=left; j-- )
46             printf("%d ",a[bottom][j]);
47
48         /* Podizemo donju vrstu za naredni krug ispisa */
49         bottom--;
50
51         if(krajIspisa(top,bottom,left,right))
52             break;
53
54         /* Ispisujemo prvu kolonu*/
55         for(i=bottom; i>=top; i-- )
```

```

56     printf("%d ",a[i][left]);
58     /* Pripremamo levu kolonu za naredni krug ispisa */
59     left++;
60 }
61 putchar('\n');
62 }
63
64 void ucitaj_matricu(int a[][MAX_K], int n, int m)
65 {
66     int i, j;
67
68     for(i=0 ;i<n; i++)
69         for(j=0; j<m; j++)
70             scanf("%d", &a[i][j]);
71 }
72
73 int main( )
74 {
75     int a[MAX_V][MAX_K];
76     int m,n;
77
78     /* Ucitaj broj vrsta i broj kolona matrice */
79     scanf("%d",&n);
80     scanf("%d", &m);
81
82     if( n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
83         fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
84         fprintf(stderr, "matrice.\n");
85         exit(EXIT_FAILURE);
86     }
87
88     ucitaj_matricu(a, n, m);
89     ispisi_matricu_spiralno(a, n, m);
90
91     return 0;
92 }

```

### Rešenje 2.14

### Rešenje 2.15

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* NAPOMENA: Primer demonstrira dinamičku alokaciju niza od n
5    elemenata. Dovoljno je alocirati n * sizeof(T) bajtova, gde
6    je T tip elemenata niza. Povratnu adresu malloc()-a treba
7    pretvoriti iz void * u T *, kako bismo dobili pokazivac
8    koji pokazuje na prvi element niza tipa T. Na dalje se
9    elementima može pristupati na isti način kao da nam
10   je dato ime niza (koje se tako i ponasa - kao pokazivac
11   na element tipa T koji je prvi u nizu) */
12 int main()

```

```
{
14  int *p = NULL;
16  int i, n;

18  /* Unosimo dimenziju niza. Ova vrednost nije ogranicena
    bilo kakvom konstantom, kao sto je to ranije bio slucaj
    kod staticke alokacije gde je dimenzija niza bila unapred
    ogranicena definisanim prostorom. */
20  scanf("%d", &n);

22

24  /* Alociramo prostor za n celih brojeva */
    if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
        fprintf(stderr, "malloc(): ");
26        fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
28    }

30    /* Od ovog trenutka pokazivac "p" mozemo da koristimo kao da
       je ime niza, odnosno i-tom elementu se moze pristupiti
32       sa p[i] */

34    /* Unosimo elemente niza */
    for (i = 0; i < n; i++)
36        scanf("%d", &p[i]);

38    /* Ispisujemo elemente niza unazad */
    for (i = n - 1; i >= 0; i--)
40        printf("%d ", p[i]);
    printf("\n");

42

44    /* Oslobadjamo prostor */
    free(p);

46    return 0;
}
```

### Rešenje 2.16

```
#include <stdio.h>
2 #include <stdlib.h>
#define KORAK 10

4
int main(void)
6 {
    /* Adresa prvog alociranog bajta*/
8    int* a = NULL;

10    /* Velicina alocirane memorije */
    int alocirano;

12

    /* Broj elemenata niza */
14    int n;

16    /* Broj koji se ucitava sa ulaza */
    int x;
```

```

18  int i;
19  int* b = NULL;
20
21  /* Inicijalizacija */
22  alocirano = n = 0;
23
24  /* Unosimo brojeve sa ulaza */
25  scanf("%d", &x);
26
27  /* Sve dok je procitani broj razlicit od nule... */
28  while(x!=0) {
29
30      /* Ako broj ucitanih elemenata niza odgovara broju
31         alociranih mesta, za smestanje novog elementa treba
32         obezbediti dodatni prostor. Da se ne bi za svaki sledeci
33         element pojedinačno alocirala memorija, prilikom
34         alokacije se vrši rezervacija za još KORAK dodatnih
35         mesta za buduće elemente */
36      if(n == alocirano) {
37          /* Povecava se broj alociranih mesta */
38          alocirano = alocirano + KORAK;
39
40          /* Vrši se realokacija memorije sa novom velicinom */
41          /******
42          /* Resenje sa funkcijom malloc() */
43          /******
44          /* Vrši se alokacija memorije sa novom velicinom, a adresa
45             početka novog memorijskog bloka se čuva u
46             promenljivoj b */
47          b = (int*) malloc (alocirano * sizeof(int));
48
49          /* Ako prilikom alokacije dodje do neke greske */
50          if(b == NULL) {
51              /* poruku ispisujemo na izlaz za greske */
52              fprintf(stderr, "malloc(): ");
53              fprintf(stderr, "greska pri alokaciji memorije.\n");
54
55              /* Pre kraja programa moramo svu dinamički alociranu
56                 memoriju da oslobodimo. U ovom slučaju samo memoriju
57                 na adresi a */
58              free(a);
59
60              /* Završavamo program */
61              exit(EXIT_FAILURE);
62          }
63
64          /* Svih n elemenata koji počinju na adresi a prepisujemo
65             na novu adresu b */
66          for(i = 0; i < n; i++)
67              b[i] = a[i];
68
69          /* Posle prepisivanja oslobadjamo blok memorije sa početnom
70             adresom u a */
71          free(a);
72
73          /* Promenljivoj a dodeljujemo adresu početka novog, većeg

```

```
74         bloka koji je prilikom alokacije zapamcen u
           promenljivoj b */
76     a = b;

78     /******
       /* Resenje sa funkcijom realloc() */
80     /******
       /* Zbog funkcije realloc je neophodno da i u prvoj
82         iteraciji "a" bude inicijalizovano na NULL */
       /*
84     a = (int*) realloc(a,alocirano*sizeof(int));

86     if(a == NULL) {
           fprintf(stderr, "realloc(): ");
88         fprintf(stderr, "greska pri alokaciji memorije.\n");
           exit(EXIT_FAILURE);
90     }
       */
92 }

94 /* Smestamo element u niz */
a[n++] = x;

96
98 /* i učitavamo sledeći element */
scanf("%d", &x);
100 }

102 /* Ispisujemo brojeve u obrnutom poretaku */
for(n--; n>=0; n--)
    printf("%d ", a[n]);
104 printf("\n");

106 /* Oslobadjamo dinamički alociranu memoriju */
free(a);

108
110 /* Program se završava */
exit(EXIT_SUCCESS);
}
```

### Rešenje 2.17

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX 1000
6
/* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
8   Ovako nesto se moze improvizovati tako sto se u funkciji
   dinamički kreira niz potrebne velicine, popuni se potrebnim
10   informacijama, a zatim se vrati njegova adresa. Imajuci u
   vidu cinjenicu da dinamički kreiran objekat ne nestaje
12   kada se izadje iz funkcije koja ga je kreirala, vraceni
   pokazivac se kasnije u pozivajucoj funkciji moze koristiti
14   za pristup "vracenom" nizu. Medjutim, pozivajuca funkcija
```

```

    ima odgovornost i da se brine o dealokaciji istog prostora */
16
/* Funkcija dinamički kreira niz karaktera u koji smesta
18 rezultat nadovezivanja niski. Adresa niza se vraća kao
    povratna vrednost. */
20 char *nadovezi(char *s, char *t) {
    /* Dinamički kreiramo prostor dovoljne velicine */
22 char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
                            * sizeof(char));
24
    /* Proveravamo uspeh alokacije */
26 if (p == NULL) {
        fprintf(stderr, "malloc(): ");
28         fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
30     }

32 /* Kopiramo i nadovezujemo stringove */

34 /* Resenje bez koriscenja biblioteckih funkcija */
    /*
36     int i,j;
    for(i=j=0; s[j]!='\0'; i++, j++)
38         p[i]=s[j];

40     for(j=0; t[j]!='\0'; i++, j++)
        p[i]=t[j];
42
    p[i]='\0';
44 */

46 /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
    string.h */
48 strcpy(p, s);
    strcat(p, t);
50
    /* Vracamo pokazivac p */
52 return p;
}
54

56 int main() {
    char *s = NULL;
    char s1[MAX], s2[MAX];
58
    /* Ucitavamo dve niske koje cemo da nadovezemo */
60 scanf("%s", s1);
    scanf("%s", s2);
62

    /* Pozivamo funkciju da nadoveze stringove */
64 s = nadovezi(s1, s2);

66 /* Prikazujemo rezultat */
    printf("%s\n", s);
68

    /* Oslobadjamo memoriju alociranu u funkciji nadovezi() */
70 free(s);

```

```
72     return 0;
    }
```

### Rešenje 2.18

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

int main()
{
    int i,j;

    /* Pokazivac na dinamički alociran niz pokazivaca na vrste
       matrice */
    double** A = NULL;

    /* Broj vrsta i broj kolona */
    int n =0, m =0;

    /* Trag matrice */
    double trag = 0;

    /* Unosimo dimenzije matrice*/
    scanf("%d%d", &n, &m);

    /* Dinamički alociramo prostor za n pokazivaca na double */
    A = malloc(sizeof(double*) * n);

    /* Proveramo da li je doslo do greske pri alokaciji */
    if(A == NULL) {
        fprintf(stderr, "malloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
    }

    /* Dinamički alociramo prostor za elemente u vrstama */
    for(i = 0; i < n; i++) {
        A[i] = malloc(sizeof(double) * m);

        if(A[i] == NULL) {
            /* Alokacija je neuspesna. Pre zavrsetka programa
               moramo da oslobodimo svih i-1 prethodno alociranih
               vrsta, i alociran niz pokazivaca */
            for( j=0; j<i; j++)
                free(A[j]);
            free(A);

            exit( EXIT_FAILURE);
        }
    }

    /* Unosimo sa standardnog ulaza brojeve u matricu.
       Popunjavamo vrstu po vrstu */
```



```

50  for(i = 0; i < n; i++)
    for(j = 0; j < m; j++ )
52      scanf("%lf", &A[i][j]);

54  /* Racunamo trag matrice, odnosno sumu elemenata
    na glavnoj dijagonali */
56  trag = 0.0;

58  for(i=0; i<n; i++)
    trag += A[i][i];
60
62  printf("%.2f\n", trag);

64  /* Oslobadjamo prostor rezervisan za svaku vrstu */
    for( j=0; j<n; j++)
        free(A[j]);
66
68  /* Oslobadjamo memoriju za niz pokazivaca na vrste */
    free(A);

70  return 0;
}

```

### Rešenje 2.19

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>

5  void ucitaj_matricu(int ** M, int n, int m)
   {
7      int i, j;

9      /* Popunjavamo matricu vrstu po vrstu */
      for(i=0; i<n; i++)
11         /* Popunjavamo i-tu vrstu matrice */
            for(j=0; j<m; j++)
13                 scanf("%d", &M[i][j]);
   }

15 void ispisujelemente_ispod_dijagonale(int ** M, int n, int m)
17 {
   int i, j;

19     for(i=0; i<n; i++) {
21         /* Ispisujemo elemente ispod glavne dijagonale matrice */
            for(j=0; j<=i; j++)
23                 printf("%d ", M[i][j]);
                printf("\n");
25     }
   }

27
29 int main() {
   int m, n, i, j;
   int **matrica = NULL;

```

```
31  /* Unosimo dimenzije matrice */
33  scanf("%d %d",&n, &m);

35  /* Alociramo prostor za niz pokazivaca na vrste matrice */
   matrica = (int **) malloc(n * sizeof(int*));
37  if(matrica == NULL) {
   fprintf(stderr,"malloc(): Neuspela alokacija\n");
39  exit(EXIT_FAILURE);
   }

41  /* Alociramo prostor za svaku vrstu matrice */
43  for(i=0; i<n; i++) {
   matrica[i] = (int*) malloc(m * sizeof(int));

45      if(matrica[i] == NULL) {
47          fprintf(stderr,"malloc(): Neuspela alokacija\n");
   for(j=0; j<i; j++)
49       free(matrica[j]);
   free(matrica);
51       exit(EXIT_FAILURE);
   }
53  }

55  ucitaj_matricu(matrica, n, m);

57  ispisi_elemente_ispod_dijagonale(matrica, n, m);

59  /* Oslobadjamo dinamički alociranu memoriju za matricu.
   Prvo oslobadjamo prostor rezervisan za svaku vrstu */
61  for( j=0; j<n; j++)
   free(matrica[j]);

63  /* Zatim oslobadjamo memoriju za niz pokazivaca na vrste
   matrice */
65  free(matrica);

67  return 0;
69 }
```

### Rešenje 2.20

```
#include<stdio.h>

2  int main(){
4  printf("Hello pokazivaci!\n");
   return 0;
6  }
```

### Rešenje 2.21

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

```

4
/* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni (float** a, int n)
{
8     int i, j;

10    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
12    /* Ako je indeks vrste manji od indeksa kolone */
        if(i<j)
14        /* element se nalazi iznad glavne dijagonale
            pa ga polovimo */
            a[i][j]/=2;
16        else
18        /* Ako je indeks vrste veci od indeksa kolone */
            if(i>j)
20            /* element se nalazi ispod glavne dijagonale
                pa ga dupliramo*/
                a[i][j] *= 2;
22    }

24    /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
26    sporedne dijagonale */
    float zbir_ispod_sporedne_dijagonale(float** m, int n)
28    {
        int i, j;
30        float zbir=0;

32        for(i=0; i<n; i++)
            for(j=0; j<n; j++)
34            /* Ukoliko je zbir indeksa vrste i indeksa kolone
                elementa veci od n-1, to znaci da se element nalazi
36            ispod sporedne dijagonale */
                if(i+j>n-1)
38                zbir+=fabs(m[i][j]);

40        return zbir;
    }

42    /* Funkcija ucitava elemente kvadratne matrice dimenzije n
44    iz zadate datoteke */
    void ucitaj_matricu(FILE* ulaz, float** m, int n) {
46        int i, j;

48        for(i=0; i<n; i++)
            for(j=0; j<n; j++)
50            fscanf(ulaz, "%f", &m[i][j]);
    }

52    /* Funkcija ispisuje elemente kvadratne matrice dimenzije n
54    na standardni izlaz */
    void ispisi_matricu(float** m, int n) {
56        int i, j;

58        for(i=0; i<n; i++){
            for(j=0; j<n; j++)

```

```
60     printf("%.2f ", m[i][j]);
61     printf("\n");
62 }
63 }
64
65 /* Funkcija alokira memoriju za kvadratnu matricu dimenzije n */
66 float** alociraj_memoriju(int n) {
67     int i, j;
68     float** m;
69
70     m = (float**) malloc(n * sizeof(float*));
71     if(m == NULL) {
72         fprintf(stderr, "malloc(): Neuspela alokacija\n");
73         exit(EXIT_FAILURE);
74     }
75
76     /* Za svaku vrstu matrice */
77     for(i=0; i<n; i++) {
78         /* Alociramo memoriju */
79         m[i] = (float*) malloc(n * sizeof(float));
80
81         /* Proveravamo da li je doslo do greske pri alokaciji */
82         if(m[i] == NULL) {
83             /* Ako jeste, ispisujemo poruku */
84             printf("malloc(): neuspela alokacija memorije!\n");
85
86             /* Oslobadjamo memoriju zauzetu do ovog koraka */
87             for(j=0; j<i; j++)
88                 free(m[j]);
89             free(m);
90             exit(EXIT_FAILURE);
91         }
92     }
93     return m;
94 }
95
96 /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom
97    dimenzije n */
98 void oslobodi_memoriju(float** m, int n)
99 {
100     int i;
101
102     for(i=0; i<n; i++)
103         free(m[i]);
104     free(m);
105 }
106
107 int main(int argc, char* argv[])
108 {
109     FILE* ulaz;
110     float** a;
111     int n;
112
113     /* Ako korisnik nije uneo trazene argumente,
114        prijavljujemo gresku */
115     if(argc < 2) {
```

```
116     printf("Greska: ");
117     printf("Nedovoljan broj argumenata komandne linije.\n");
118     printf("Program se poziva sa %s ime_dat.\n", argv[0]);
119     exit(EXIT_FAILURE);
120 }
121
122 /* Otvaramo datoteku za citanje */
123 ulaz = fopen(argv[1], "r");
124 if(ulaz == NULL) {
125     fprintf(stderr, "Greska: ");
126     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
127               argv[1]);
128     exit(EXIT_FAILURE);
129 }
130
131 /* citamo dimenziju matrice */
132 fscanf(ulaz, "%d", &n);
133
134 /* Alociramo memoriju */
135 a = alociraj_memoriju(n);
136
137 /* Ucitavamo elemente matrice */
138 ucitaj_matricu(ulaz, a, n);
139
140 float zbir = zbir_ispod_sporedne_dijagonale(a, n);
141
142 /* Pozivamo funkciju za modifikovanje elemenata */
143 izmeni(a, n);
144
145 /* Ispisujemo rezultat */
146 printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
147 printf("je %.2f.\n", zbir);
148
149 printf("Transformisana matrica je:\n");
150 ispisi_matricu(a,n);
151
152 /* Oslobadjamo memoriju */
153 oslobodi_memoriju(a, n);
154
155 /* Zatvaramo datoteku */
156 fclose(ulaz);
157
158 /* i prekidamo sa izvršavanjem programa */
159 return 0;
160 }
```

Rešenje [2.22](#)

Rešenje [2.23](#)

Rešenje [2.24](#)

Rešenje [2.25](#)

## Rešenje 2.26

```
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <math.h>
5 #include <string.h>
6
7 /* NAPOMENA:
8  Zaglavlje math.h sadrži deklaracije raznih matematičkih
9  funkcija. Izmeđ ostalog, to su sledeće funkcije:
10 double sin(double x);
11 double cos(double x);
12 double tan(double x);
13 double asin(double x);
14 double acos(double x);
15 double atan(double x);
16 double atan2(double y, double x);
17 double sinh(double x);
18 double cosh(double x);
19 double tanh(double x);
20 double exp(double x);
21 double log(double x);
22 double log10(double x);
23 double pow(double x, double y);
24 double sqrt(double x);
25 double ceil(double x);
26 double floor(double x);
27 double fabs(double x);
28 */
29
30 /* Funkcija tabela() prihvata granice intervala a i b, broj
31 ekvidistantnih čtaaka n, kao i pokaziva f koji pokazuje
32 na funkciju koja prihvata double argument, i čvraa double
33 vrednost. Za tako datu funkciju ispisuje njene vrednosti
34 u intervalu [a,b] u n ekvidistantnih čtaaka intervala */
35 void tabela(double a, double b, int n, double (*fp)(double))
36 {
37     int i;
38     double x;
39
40     printf("-----\n");
41     for(i=0; i<n; i++) {
42         x= a + i*(b-a)/(n-1);
43         printf("| %8.5f | %8.5f |\n", x, (*fp)(x));
44     }
45     printf("-----\n");
46 }
47
48 /* Umesto da koristimo stepenu funkciju iz zaglavlja
49 math.h -> pow(a,2) čpozivaemo čkorisniku sqr(a) */
50 double sqr (double a)
51 {
52     return a*a;
53 }
54
```

```
int main(int argc, char *argv[])
{
    double a, b;
    int n;
    /* Imena funkcija koja ćemo navoditi su čkraa ili čtano duga
       5 karaktera */
    char ime_fje[6];
    /* Pokazivac na funkciju koja ima jedan argument tipa double i
       povratnu vrednost istog tipa */
    double (*fp)(double);

    /* Ako korisnik nije uneo žtraene argumente,
       prijavljujemo šgreku */
    if(argc < 2) {
        printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
               argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Niska ime_fje žsadri ime žtraene funkcije koja je navedena
       u komandnoj liniji */
    strcpy(ime_fje, argv[1]);

    /* Inicijalizujemo čpokaziva na funkciju koja treba da se
       tabelira */
    if(strcmp(ime_fje, "sin") == 0)
        fp=&sin;
    else if(strcmp(ime_fje, "cos") == 0)
        fp=&cos;
    else if(strcmp(ime_fje, "tan") == 0)
        fp=&tan;
    else if(strcmp(ime_fje, "atan") == 0)
        fp=&atan;
    else if(strcmp(ime_fje, "acos") == 0)
        fp=&acos;
    else if(strcmp(ime_fje, "asin") == 0)
        fp=&asin;
    else if(strcmp(ime_fje, "exp") == 0)
        fp=&exp;
    else if(strcmp(ime_fje, "log") == 0)
        fp=&log;
    else if(strcmp(ime_fje, "log10") == 0)
        fp=&log10;
    else if(strcmp(ime_fje, "sqrt") == 0)
        fp=&sqrt;
    else if(strcmp(ime_fje, "floor") == 0)
        fp=&floor;
    else if(strcmp(ime_fje, "ceil") == 0)
        fp=&ceil;
    else if(strcmp(ime_fje, "sqr") == 0)
        fp=&sqr;
    else {
        printf("Program jos uvek ne podrzava trazenu funkciju!\n");
        exit(EXIT_SUCCESS);
    }
}
```

```
112     }
113
114     printf("Unesite krajeve intervala:\n" );
115     scanf("%lf %lf", &a, &b);
116
117     printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
118     printf("(ukljucujuci krajeve intervala)?\n");
119     scanf("%d", &n );
120
121     /* Mreza mora da čukljuuje bar krajeve intervala,
122        tako da se mora uneti broj veci od 2 */
123     if (n < 2) {
124         fprintf(stderr, "Broj čtaaka žmree mora biti bar 2!\n");
125         exit(EXIT_FAILURE);
126     }
127
128     /* Ispisujemo ime funkcije */
129     printf("      x %10s(x)\n", ime_fje);
130
131     /* dProsleujemo funkciji tabela() funkciju zadatu kao
132        argument komandne linije */
133     tabela(a, b, n, fp);
134
135     exit(EXIT_SUCCESS);
136 }
```

Rešenje [2.27](#)

Rešenje [2.28](#)

Rešenje [2.29](#)



## Glava 3

# Algoritmi pretrage i sortiranja

### 3.1 Pretraživanje

**Zadatak 3.1** Napisati iterativne funkcije pretraga nizova. Svaka funkcija treba da vrati indeks pozicije na kojoj je pronađen traženi element ili broj  $-1$  ukoliko element nije pronađen.

- (a) Napisati funkciju koja vrši linearnu pretragu niza celih brojeva  $a$ , dužine  $n$ , tražeći u njemu broj  $x$ .
- (b) Napisati funkciju koja vrši binarnu pretragu sortiranog niza  $a$ , dužine  $n$ , tražeći u njemu broj  $x$ .
- (c) Napisati funkciju koja vrši interpoacionu pretragu sortiranog niza  $a$ , dužine  $n$ , tražeći u njemu broj  $x$ .

Napisati i program koji generiše slučajni rastući niz dimenzije  $n$  (prvi argument komandne linije), i u njemu već napisanim funkcijama traži element  $x$  (drugi argument komandne linije). Potrebna vremena za izvršavanje ovih funkcija upisati u fajl `vremena.txt`.

*Test 1*

```
Poziv: ./a.out 1000000 235423
```

```
Izlaz:
```

```
Linearna pretraga
```

```
Element nije u nizu
```

```
-----  
Binarna pretraga
```

```
Element nije u nizu
```

```
-----  
Interpolaciona pretraga
```

```
Element nije u nizu
```

```
-----
```

[Rešenje 3.1]

### 3 Algoritmi pretrage i sortiranja

**Zadatak 3.2** Napisati rekurzivne funkcije algoritama linearne, binarne i interpolacione pretrage i program koji ih testira za brojeve koji se unose sa standardnog ulaza. Linearnu pretragu implementirati na dva načina, svodenjem pretrage na prefiks i na sufiks niza. Pretpostaviti da niz brojeva koji se unosi neće biti duži od 1024 elemenata. Prvo se unosi broj koji se traži, a zatim sortirani elementi niza sve do kraja ulaza.

<i>Test 1</i>	<i>Test 2</i>
<pre>Ulaz: 11 2 5 6 8 10 11 23 Izlaz: Linearna pretraga Pozicija elementa je 5. Binarna pretraga Pozicija elementa je 5. Interpolaciona pretraga Pozicija elementa je 5.</pre>	<pre>Ulaz: 14 10 32 35 43 66 89 100 Izlaz: Linearna pretraga Element se ne nalazi u nizu. Binarna pretraga Element se ne nalazi u nizu. Interpolaciona pretraga Element se ne nalazi u nizu.</pre>

[Rešenje 3.2]

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks studenta čije informacije se potom prikazuju na ekranu. Zatim, korisnik učitava prezime studenta i prikazuju mu se informacije o prvom studentu sa unetim prezimenom. Pretrage implementirati u vidu iterativnih funkcija što bolje manje složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata, i da su imena i prezimena svih kraća od 16 slova.

*Test 1*

```
Datoteka:
20140003 Marina Petrovic
20140012 Stefan Mitrovic
20140032 Dejan Popovic
20140049 Mirko Brankovic
20140076 Sonja Stevanovic
20140104 Ivan Popovic
20140187 Vlada Stankovic
20140234 Darko Brankovic
Ulaz:
20140076
Popovic
Izlaz:
Indeks: 20140076, Ime i prezime: Sonja Stevanovic
Indeks: 20140032, Ime i prezime: Dejan Popovic
```

[Rešenje 3.3]

**Zadatak 3.4** Modifikovati prethodni zadatak 3.3 tako da tražene funkcije budu rekurzivne.

[Rešenje 3.4]

**Zadatak 3.5** U datoteci koja se zadaje kao prvi argument komandne linije, nalaze se koordinate tačaka. U zavisnosti od prisustva opcija komandne linije ( $-x$  ili  $-y$ ), pronaći onu koja je najbliža  $x$  (ili  $y$ ) osi, ili koordinatnom početku, ako nije prisutna nijedna opcija. Pretpostaviti da je broj tačaka u datoteci veći od 0 i ne veći od 1024.

*Test 1*

```
|| Poziv: ./a.out dat.txt -x
|| Datoteka:
|| 12 53
|| 2.342 34.1
|| -0.3 23
|| -1 23.1
|| 123.5 756.12
|| Izlaz: -0.3 23
```

[Rešenje 3.5]

**Zadatak 3.6** Napisati funkciju koja određuje nulu funkcije  $\cos(x)$  na intervalu  $[0, 2]$  metodom polovljenja intervala. Algoritam se završava kada se vrednost kosinusne funkcije razlikuje za najviše 0.001 od nule. Uputstvo: koristiti algoritam analogan algoritmu binarne pretrage.

*Test 1*

```
|| Izlaz: 1.57031
```

[Rešenje 3.6]

**Zadatak 3.7** Napisati funkciju koja u rastuće sortiranom nizu celih brojeva binarnom pretragom pronalazi prvi element veći od nule i kao rezultat vraća njegovu poziciju u nizu. Ukoliko nema elemenata većih od nule, funkcija kao rezultat vraća  $-1$ . Napisati program koji testira ovu funkciju za niz elemenata koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

*Test 1*

```
|| Ulaz: -151 -44 5
|| 12 13 15
|| Izlaz: 2
```

*Test 2*

```
|| Ulaz: -100 -15 -11
|| -8 -7 -5
|| Izlaz: -1
```

*Test 3*

```
|| Ulaz: -100 -15 0 13
|| 55 124 258
|| 315 516 7000
|| Izlaz: 3
```

[Rešenje 3.7]

**Zadatak 3.8** Napisati funkciju koja u opadajuće sortiranom nizu celih brojeva binarnom pretragom pronalazi prvi element manji od nule i kao rezultat vraća njegovu poziciju u nizu. Ukoliko nema elemenata manjih od nule, funkcija kao rezultat vraća  $-1$ . Napisati program koji testira ovu funkciju za niz elemenata koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

*Test 1*

```
|| Ulaz:  151 44 5 -12 -13 -15
|| Izlaz: 3
```

*Test 2*

```
|| Ulaz:  100 55 15 0 -15 -124 -155
||      -258 -315 -516 -7000
|| Izlaz: 4
```

*Test 3*

```
|| Ulaz:  100 15 11 8 7 5 4 3 2
|| Izlaz: -1
```

[Rešenje 3.8]

**Zadatak 3.9** Napisati funkciju koja određuje ceo deo logaritma za osnovu 2 datog neoznačenog celog broja, koristeći samo bitske i relacione operatore.

- (a) Napisati funkciju, linearne složenosti, koja određuje logaritam pomeranjem broja udesno.
- (b) Napisati funkciju, logaritmske složenosti, koja određuje logaritam koristeći binarnu pretragu.

Tražene funkcije testirati programom koji broj učitava sa standardnog ulaza, a logaritam ispisuje na standardni izlaz.

*Test 1*

```
|| Ulaz:  10
|| Izlaz: 3 3
```

*Test 2*

```
|| Ulaz:  4
|| Izlaz: 2 2
```

*Test 3*

```
|| Ulaz:  17
|| Izlaz: 4 4
```

*Test 4*

```
|| Ulaz:  1031
|| Izlaz: 10 10
```

[Rešenje 3.9]

**\*\* Zadatak 3.10** U prvom kvadrantu dato je  $1 \leq N \leq 10000$  duži svojim koordinatama (duži mogu da se seku, preklapaju, itd.). Napisati program koji pronalazi najmanji ugao  $0 \leq \alpha \leq 90^\circ$ , na dve decimale, takav da je suma dužina duži sa obe strane polupoluprave iz koordinatnog početka pod uglom  $\alpha$  jednak (neke duži bivaju presečene, a neke ne). Program prvo učitava broj  $N$ , a zatim i same koordinate temena duži. Uputstvo: vršiti binarnu pretragu intervala  $[0, 90^\circ]$ .

*Test 1*

```
|| Ulaz:
||      2
||      2 0 2 1
||      1 2 2 2
|| Izlaz: 26.57
```

**Zadatak 3.11** Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime (uređen u rastućem poretку prezimena) sa manje od 10 elemenata, a zatim se učitava jedan karakter i pronalazi (bibliotečkom funkcijom `bsearch`) i štampa jedna struktura iz niza osoba čije prezime počinje tim karakterom. Ako takva osoba ne postoji, štampati `-1` na standardni izlaz.

```
Osoba niz_osoba[]={{"Mika", "Antic"},
                   {"Dobrica", "Eric"},
                   {"Desanka", "Maksimovic"},
                   {"Dusko", "Radovic"},
                   {"Ljubivoje", "Rsumovic"}};
```

*Test 1*

```
|| Ulaz:  R
|| Izlaz: Dusko Radovic
```

## 3.2 Sortiranje

**Zadatak 3.12** U datom nizu brojeva pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, i neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati njihovu razliku. Uputstvo: prvo sortirati niz.

*Test 1*

```
|| Ulaz:  23 64 123 76 22 7
|| Izlaz:  1
```

*Test 2*

```
|| Ulaz:  21 654 65 123 65 12 61
|| Izlaz:  0
```

[Rešenje 3.12]

**Zadatak 3.13** Dve niske su anagrami ako se sastoje od istog broja istih karaktera. Napisati program koji proverava da li su dve niske karaktera anagrami. Niske se zadaju sa standardnog ulaza, i neće biti duže od 127 karaktera. Uputstvo: napisati funkciju koja sortira slova unutar niske karaktera, a zatim za sortirane niske proveriti da li su identične.

*Test 1*

```
|| Ulaz:  anagram ramgana
|| Izlaz:  jesu
```

*Test 2*

```
|| Ulaz:  anagram anagrm
|| Izlaz:  nisu
```

[Rešenje 3.13]

### 3 Algoritmi pretrage i sortiranja

---

**Zadatak 3.14** Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza, i neće biti duži od 256 i kraći od jednog elemenata. Uputstvo: prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.

	<i>Test 1</i>		<i>Test 2</i>
	Ulaz: 4 23 5 2 4 6 7 34 6 4 5		Ulaz: 2 4 6 2 6 7 99 1
	Izlaz: 4		Izlaz: 2

[Rešenje 3.14]

**Zadatak 3.15** Napisati funkciju koja proverava da li u datom nizu postoje dva elementa kojima je zbir zadati ceo broj. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz (pretpostaviti da za niz neće biti uneto više od 256 brojeva). Elementi niza se unose sve do kraja ulaza. Uputstvo: prvo sortirati niz.

	<i>Test 1</i>		<i>Test 2</i>
	Ulaz: 34 134 4 1 6 30 23		Ulaz: 12 53 1 43 3 56 13
	Izlaz: da		Izlaz: ne

[Rešenje 3.15]

**Zadatak 3.16** Napisati funkciju potpisa `int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3, int dim3)` koja prima dva sortirana niza, i na osnovu njih pravi novi sortirani niz koji koji sadrži elemente oba niza. Treća dimenzija predstavlja veličinu niza u koji se smešta rezultat. Ako je ona manja od potrebne dužine, funkcija vraća -1, kao indikator neuspeha, inače vraća 0. Napisati i program koji testira funkciju, u kome se nizovi unose sa standardnog ulaza, sve dok se ne unese 0. Dimenzija nizova neće biti preko 256.

	<i>Test 1</i>
	Ulaz: 3 6 7 11 14 35 0 3 5 8 0
	Izlaz: 3 3 5 6 7 8 11 14 35

	<i>Test 2</i>
	Ulaz: 1 4 7 0 9 11 23 54 75 0
	Izlaz: 1 4 7 9 11 23 54 75

[Rešenje 3.16]

**Zadatak 3.17** Napisati program koji čita sadržaj dve datoteke od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima i kreira jedinstven spisak studenata sortiranih takođe po imenu rastuće. Program dobija nazive datoteka iz komandne linije, i jedinstven spisak upisuje u datoteku `ceo-tok.txt`. Pretpostaviti da je ime studenta nije duže od 10, a prezime od 15 karaktera.

*Test 1*

```

Poziv: ./a.out prvi-deo.txt drugi-deo.txt
Ulazne datoteke:
    prvi-deo.txt:          drugi-deo.txt:

    Andrija Petrovic      Aleksandra Cvetic
    Anja Ilic             Bojan Golubovic
    Ivana Markovic        Dragan Markovic
    Lazar Micic           Filip Dukic
    Nenad Brankovic       Ivana Stankovic
    Sofija Filipovic      Marija Stankovic
    Vladimir Savic        Ognjen Peric
    Uros Milic

Izlazna datoteka (ceo-tok.txt):

    Aleksandra Cvetic
    Andrija Petrovic
    Anja Ilic
    Bojan Golubovic
    Dragan Markovic
    Filip Dukic
    Ivana Stankovic
    Ivana Markovic
    Lazar Micic
    Marija Stankovic
    Nenad Brankovic
    Ognjen Peric
    Sofija Filipovic
    Uros Milic
    Vladimir Savic

```

[Rešenje 3.17]

**Zadatak 3.18** Napraviti biblioteku `sort.h` i `sort.c` koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži `selection`, `merge`, `quick`, `bubble`, `insertion` i `shell sort`. Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na već sortiranim nizovima i na naopako sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Vreme meriti programom `time`. Analizirati porast vremena sa porastom dimenzije  $n$ .

*Upotreba programa 1*

```

Poziv: time ./a.out 100000 -i -o
Izlaz:
    real    0m17.631s
    user    0m17.604s
    sys     0m0.000s

```

*Upotreba programa 2*

```

Poziv: time ./a.out 100000 -b -r
Izlaz:
    real    0m0.005s
    user    0m0.004s
    sys     0m0.000s

```

#### Upotreba programa 3

```
Poziv: time ./a.out 100000 -s
Izlaz:
    real    0m0.071s
    user    0m0.068s
    sys     0m0.000s
```

[Rešenje 3.18]

**Zadatak 3.19** Napisati funkcije koje sortiraju niz struktura tačaka na osnovu sledećih kriterijuma:

- (a) njihovog rastojanja od koordinatnog početka,
- (b) x koordinata tačaka,
- (c) y koordinata tačaka.

Napisati program koji učitava niz tačaka iz datoteke čije se ime zadaje kao drugi argument komandne linije, i u zavisnosti od prisutnih opcija (prvi argument) u komandnoj liniji (-o, -x ili -y), sortira tačke po jednom od prethodna tri kriterijuma i rezultat upisuje u datoteku čije se ime zadaje kao treći argument komandne linije. U ulaznoj datoteci nije zadato više od 128 tačaka.

#### Test 1

```
Poziv: ./a.out -x in.txt out.txt
Ulazna datoteka (in.txt):
    3 4
    11 6
    7 3
    2 82
    -1 6
Izlazna datoteka (out.txt):
    -1 6
    2 82
    3 4
    7 3
    11 6
```

#### Test 2

```
Poziv: ./a.out -o in.txt out.txt
Ulazna datoteka (on.txt):
    3 4
    11 6
    7 3
    2 82
    -1 6
Izlazna datoteka (out.txt):
    3 4
    -1 6
    7 3
    11 6
    2 82
```

[Rešenje 3.19]

**Zadatak 3.20** Napisati program koji učitava imena i prezimena građana (najviše njih 1000) iz datoteke `biracki-spisak.txt`, i kreira biračke spiskove. Jedan birački spisak je sortiran po imenu građana, a drugi po prezimenu. Program treba da ispisuje koliko građana ima isti redni broj u oba biračka spiska. Pretpostaviti da je za ime, odnosno prezime građana dovoljno 15 karaktera.



*Test 1*

```

Ulazna datoteka (biracki-spisak.txt):
Andrija Petrovic
Anja Ilic
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Ivana Markovic
Lazar Micic
Marija Stankovic
Izlaz: 3

```

[Rešenje 3.20]

**Zadatak 3.21** Definisana je struktura podataka

```

typedef struct dete
{
    char ime[MAX_IME];
    char prezime[MAX_IME];
    unsigned godiste;
} Dete;

```

Napisati funkciju koja sortira niz dece po godištu, a kada su deca istog godišta, tada ih sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci, čije se ime zadaje kao prvi argument komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 deteta.

*Test 1*

```

Poziv: ./a.out in.txt out.txt
Ulazna datoteka (in.out):
Petar Petrovic 2007
Milica Antonic 2008
Ana Petrovic 2007
Ivana Ivanovic 2009
Dragana Markovic 2010
Marija Antic 2007
Izlazna datoteka (out.txt):
Marija Antic 2007
Ana Petrovic 2007
Petar Petrovic 2007
Milica Antonic 2008
Ivana Ivanovic 2009
Dragana Markovic 2010

```

**Zadatak 3.22** Napisati funkciju koja sortira niz niski po broju suglasnika u niski, ukoliko reči imaju isti broj suglasnika tada po dužini niske, a ukoliko su i

### 3 Algoritmi pretrage i sortiranja

---

dužine jednake onda leksikografski. Napisati program koji testira ovu funkciju za niske koje se zadaju u datoteci `niske.txt`. Pretpostaviti da u nizu nema više od 128 elemenata, kao i da svaka niska sadrži najviše 31 karakter.

#### *Test 1*

```
Ulazna datoteka (niske.txt):  
ana petar andjela milos nikola aleksandar ljubica matej milica  
Izlaz:  
ana matej milos petar milica nikola andjela ljubica aleksandar
```

[Rešenje 3.22]

**Zadatak 3.23** Napisati program koji simulira rad kase u prodavnici. Kupci prilaze kasi, a prodavac unošenjem bar-koda kupljenog proizvoda dodaje njegovu cenu na ukupan račun. Na kraju, program ispisuje ukupnu vrednost svih proizvoda. Sve artikle, zajedno sa bar-kodovima, proizvođačima i cenama učitati iz datoteke `artikli.txt`. Pretraživanje niza artikala vršiti binarnom pretragom.

*Upotreba programa 1*

```

Ulazna datoteka (artikli.txt):
1001 Keks Jaffa 120
2530 Napolitanke Bambi 230
0023 Medeno_srce Pionir 150
2145 Pardon Marbo 70
Interakcija programa:
Asortiman:
KOD          Naziv artikla      Ime proizvođača      Cena
      23          Medeno_srce      Pionir      150.00
    1001          Keks          Jaffa      120.00
    2145          Pardon      Marbo      70.00
    2530      Napolitanke      Bambi      230.00
-----
- Za kraj za kraj rada kase, pritisnite CTRL+D!
- Za nov racun unesite kod artikla!

1001
  Trazili ste:  Keks Jaffa      120.00
Unesite kod artikla [ili 0 za prekid]:  23
  Trazili ste:  Medeno_srce Pionir      150.00
Unesite kod artikla [ili 0 za prekid]:  0

  UKUPNO: 270.00 dinara.

-----
- Za kraj za kraj rada kase, pritisnite CTRL+D!
- Za nov racun unesite kod artikla!

232
  GRESKA: Ne postoji proizvod sa traženim kodom!
Unesite kod artikla [ili 0 za prekid]:  2530
  Trazili ste:  Napolitanke Bambi      230.00
Unesite kod artikla [ili 0 za prekid]:  0

  UKUPNO: 230.00 dinara.

-----
- Za kraj za kraj rada kase, pritisnite CTRL+D!
- Za nov racun unesite kod artikla!

Kraj rada kase!

```

[Rešenje 3.23]

**Zadatak 3.24** Napisati program koji iz datoteke `aktivnost.txt` čita podatke o aktivnosti studenata na praktikumima i u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt` upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po

### ***3 Algoritmi pretrage i sortiranja***

---

broju urađenih zadataka, a ukoliko se i on poklapa sortirati po prezimenu opadajuće. U datoteci se nalazi ime, prezime studenta, broj časova na kojima je prisustvovao, kao i ukupan broj urađenih zadataka. Pretpostaviti da studenata neće biti više od 500 i da je za ime studenta dovoljno 20, a za prezime 25 karaktera.

### Test 1

```
Ulazna datoteka (aktivnosti.txt):
Aleksandra Cvetic 4 6
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Marija Stankovic 1 3
Ognjen Peric 1 2
Uros Milic 2 5
Andrija Petrovic 2 5
Anja Ilic 3 1
Lazar Micic 1 3
Nenad Brankovic 2 4
Izlazna datoteka (dat1.txt):
Studenti sortirani po imenu leksikografski rastuce:
Aleksandra Cvetic 4 6
Andrija Petrovic 2 5
Anja Ilic 3 1
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Lazar Micic 1 3
Marija Stankovic 1 3
Nenad Brankovic 2 4
Ognjen Peric 1 2
Uros Milic 2 5
Izlazna datoteka (dat2.txt):
Studenti sortirani po broju zadataka opadajuce,
pa po duzini imena rastuce:
Aleksandra Cvetic 4 6
Uros Milic 2 5
Dragan Markovic 3 5
Andrija Petrovic 2 5
Nenad Brankovic 2 4
Lazar Micic 1 3
Bojan Golubovic 4 3
Marija Stankovic 1 3
Ognjen Peric 1 2
Anja Ilic 3 1
Ivana Stankovic 3 1
Izlazna datoteka (dat3.txt):
Studenti sortirani po prisustvu opadajuce,
pa po broju zadataka,
pa po prezimenima leksikografski opadajuce:
Aleksandra Cvetic 4 6
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Anja Ilic 3 1
Andrija Petrovic 2 5
Uros Milic 2 5
Nenad Brankovic 2 4
Marija Stankovic 1 3
Lazar Micic 1 3
Ognjen Peric 1 2
```

[Rešenje 3.24]

**Zadatak 3.25** U datoteci „pesme.txt“ nalaze se informacije o gledanosti pesama na Youtube-u. Format datoteke sa informacijama je sledeći:

- U prvoj liniji datoteke se nalazi ukupan broj pesama prisutnih u datoteci.
- Svaki naredni red datoteke sadrži informacije o gledanosti pesama u formatu **izvođač - naslov, broj gledanja**.

Napisati program koji učitava informacije o pesmama i vrši sortiranje pesama u zavisnosti od argumenata komandne linije na sledeći način:

- nema opcija, sortiranje se vrši po broju gledanja;
- prisutna je opcija **-i**, sortiranje se vrši po imenima izvođača;
- prisutna je opcija **-n**, sortiranje se vrši po naslovu pesama.

Na standardni izlaz ispisati informacije o pesmama sortirane na opisan način. Uraditi zadatak bez pravljenja pretpostavki o maksimalnoj dužini imena izvođača i naslova pesme.

*Test 1*

```
Poziv: ./a.out
Datoteka: 5
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
        Mika - Kisa, 5341
Izlaz:  Jelena - Sunce, 92321
        Mika - Kisa, 5341
        Ana - Nebo, 2342
        Pera - Ptice, 327
        Laza - Oblaci, 29
```

*Test 2*

```
Poziv: ./a.out -i
Datoteka: 5
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
        Mika - Kisa, 5341
Izlaz:  Ana - Nebo, 2342
        Jelena - Sunce, 92321
        Laza - Oblaci, 29
        Mika - Kisa, 5341
        Pera - Ptice, 327
```

*Test 3*

```
Poziv: ./a.out -n
Datoteka: 5
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
        Mika - Kisa, 5341
Izlaz:  Mika - Kisa, 5341
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
```

[Rešenje 3.25]

**\*\* Zadatak 3.26** Razmatrajmo dve operacije: operacija U je unos novog broja  $x$ , a operacija N određivanje  $n$ -tog po veličini od unetih brojeva. Implementirati program koji izvršava ove operacije. Može postojati najviše 100000 operacija unosa, a uneti elementi se mogu ponavljati, pri čemu se i ponavljanja računaju prilikom brojanja. Napomena: brojeve čuvati u sortiranom nizu i svaki naredni element umetati na svoje mesto. Optimizovati program, ukoliko se zna da neće biti više od 500 različitih unetih brojeva.

#### Test 1

```
Ulaz: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5
Izlaz: 0 2 8 2 6
```

**\*\* Zadatak 3.27** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze najmanja, najveća, itd... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene.

3	5	2	1
4	4	1__	2
5__	3	3	3
1	1	4	4
2	2__	5	5

Napisati program koji u najviše  $2n-3$  okretanja sortira učitani niz. Uputstvo: imitirati `selection sort` i u svakom koraku dovesti jednu palačinku na svoje mesto korišćenjem najviše dva okretanja.

### 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.28** Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova, i mogućnost zadavanja različitih kriterijuma sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva (ne veća od 100), a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu i na standardni izlaz ispisati odgovarajuću poruku.

#### Upotreba programa 1

```
Interakcija programa:
Uneti dimenziju niza: 10
Uneti elemente niza:
5 3 1 6 8 90 34 5 3 432
Sortirani niz u rastucem
poretku:
1 3 3 5 5 6 8 34 90 432
Uneti element koji se trazi u
nizu: 34
Binarna pretraga:
Element je nadjen na poziciji 7
Linearna pretraga (lfind):
Element je nadjen na poziciji 7
```

#### Upotreba programa 2

```
Interakcija programa:
Uneti dimenziju niza: 4
Uneti elemente niza:
4 2 5 7
Sortirani niz u rastucem
poretku:
2 4 5 7
Uneti element koji se trazi u
nizu: 3
Binarna pretraga:
Elementa nema u nizu!
Linearna pretraga (lfind):
Elementa nema u nizu!
```

[Rešenje 3.28]

**Zadatak 3.29** Napisati program koji sa standardnog ulaza učitava dimenziju niza celih brojeva (ne veću od 100), a potom i same elemente niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku prema broju delilaca i tako dobijeni niz odštampati na standardni izlaz.

#### Upotreba programa 1

```
Interakcija programa:
Uneti dimenziju niza: 10
Uneti elemente niza:
1 2 3 4 5 6 7 8 9 10
Sortirani niz u rastucem poretku prema broju delilaca:
1 2 3 5 7 4 9 6 8 10
```

[Rešenje 3.29]

**Zadatak 3.30** Korišćenjem bibliotečke funkcije `qsort` napisati program koji sortira niz niski po sledećim kriterijumima:

- (a) leksikografski,
- (b) po dužini.

Niske se učitavaju iz fajla `niske.txt`, neće ih biti više od 1000, i svaka će biti dužine najviše 30 karaktera. Program prvo leksikografski sortira niz, primenjuje binarnu pretragu (`bsearch`) zarad traženja niske unete sa standardnog ulaza, a potom linearnu pretragu koristeći funkciju `lfind`. Na kraju, niske bivaju sortirane po dužini. Rezultate svih sortiranja i pretraga ispisati na standardni izlaz.



#### Test 1

```
Ulazna datoteka (niske.txt):
ana petar andjela milos nikola aleksandar ljubica matej milica
Interakcija programa:
Leksikografski sortirane niske:
aleksandar ana andjela ljubica matej milica milos nikola petar
Uneti trazenu nisku: matej
Niska "matej" je pronadjena u nizu na poziciji 4
Niska "matej" je pronadjena u nizu na poziciji 4
Niske sortirane po duzini:
ana matej milos petar milica nikola andjela ljubica aleksandar
```

[Rešenje 3.30]

**Zadatak 3.31** Uraditi prethodni zadatak 3.30 sa dinamički alociranim niskama, i sortiranjem niza pokazivača (umesto niza niski).

[Rešenje 3.31]

**Zadatak 3.32** Napisati program koji korišćenjem bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova, ili poruka ukoliko nema takvog. Potom, sa standardnom ulaza, unosi se prezime traženog studenta, i prikazuje se osoba sa tim prezimenom, ili poruka da se nijedan student tako ne preziva. Za pretraživanje, koristiti odgovarajuće bibliotečke funkcije. Podaci o studentima čitaju se iz datoteke čije se ime zadaje preko argumenata komandne linije. Za svakog studenta u datoteci postoje ime, prezime i bodovi osvojeni na kolokvijumu. Pretpostaviti da neće biti više od 500 studenata, i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

#### Test 1

```
Poziv: ./a.out kolokvijum.txt
Ulazna datoteka (kolokvijum.txt):
Aleksandra Cvetic 15
Bojan Golubovic 30
Dragan Markovic 25
Filip Dukic 20
Ivana Stankovic 25
Marija Stankovic 15
Ognjen Peric 20
Uros Milic 10
Andrija Petrovic 0
Anja Ilic 5
Ivana Markovic 5
Lazar Micic 20
Nenad Brankovic 15
Interakcija programa:
Studenti sortirani po broju poena opadajuće, pa po prezimenu rastuće:
Bojan Golubovic 30
Dragan Markovic 25
Ivana Stankovic 25
Filip Dukic 20
Lazar Micic 20
Ognjen Peric 20
Nenad Brankovic 15
Aleksandra Cvetic 15
Marija Stankovic 15
Uros Milic 10
Anja Ilic 5
Ivana Markovic 5
Andrija Petrovic 0
Unesite broj bodova: 20
Pronadjen je student sa unetim brojem bodova: Filip Dukic 20
Unesite prezime: Markovic
Pronadjen je student sa unetim prezimenom: Dragan Markovic 25
```

[Rešenje 3.32]

**Zadatak 3.33** Uraditi zadatak 3.13, ali korišćenjem bibliotečke `qsort` funkcije.

[Rešenje 3.33]

**Zadatak 3.34** Napisati program koji sa standardnog ulaza učitava prvo ceo broj  $n$  ( $n \leq 10$ ), a zatim niz  $S$  od  $n$  stringova (maksimalna dužina stringa je 31 karaktera). Sortirati niz  $S$  (bibliotečkom funkcijom `qsort`) i proveriti da li u njemu ima identičnih stringova.

#### Test 1

#### Test 2

```
Ulaz: 4 prog search sort search
Izlaz: ima

Ulaz: 3 test kol ispit
Izlaz: nema
```

[Rešenje 3.34]

**Zadatak 3.35** Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr97125`, `mm09001`), ime i prezime i broj poena. Napisati program koji sortira (korišćenjem funkcije `qsort`) studente po broju poena (ukoliko je prisutna opcija `-p`) ili po nalogu (ukoliko je prisutna opcija `-n`). Studenti se po nalogu sortiraju tako što se sortiraju na osnovu godine, zatim na osnovu smera, i na kraju na osnovu broja indeksa. Sortirane studente upisati u datoteku `izlaz.txt`. Ukoliko je u komandnoj liniji uz opciju `-n` naveden i nalog nekog studenta, funkcijom `bsearch` potražiti i prijaviti broj poena studenta sa tim nalogom.

#### Test 1

```
Poziv: ./a.out -n mm13321
Ulazna datoteka (studenti.txt):
  mr14123 Marko Antic 20
  mm13321 Marija Radic 12
  ml13011 Ivana Mitrovic 19
  ml13066 Pera Simic 15
  mv14003 Jovan Jovanovic 17
Izlazna datoteka (izlaz.txt):
  ml13011 Ivana Mitrovic 19
  ml13066 Pera Simic 15
  mm13321 Marija Radic 12
  mr14123 Marko Antic 20
  mv14003 Jovan Jovanovic 17
Izlaz:
  mm13321 Marija Radic 12
```

[Rešenje 3.35]

**Zadatak 3.36** Definisana je struktura:

```
typedef struct { int dan; int mesec; int godina; } Datum;
```

Napisati funkciju koja poredi dva datuma i program koji učitava datume iz datoteke koja se zadaje kao prvi argument komandne linije (ne više od 128 datuma), sortira ih pozivajući funkciju `qsort` iz standardne biblioteke i potom pozivanjem funkcije `bsearch` iz standardne biblioteke proverava da li datumi učitani sa standardnog ulaza (sve do kraja ulaza) postoje među prethodno unetim datumima.

#### Test 1

```
Poziv: ./a.out datoteka.txt
Datoteka:      Ulaz:      Izlaz:
1.1.2013        13.12.2016  postoji
13.12.2016      10.5.2015   ne postoji
11.11.2011      5.2.2009    postoji
3.5.2015
5.2.2009
```

**Zadatak 3.37** Za zadatu celobrojnu matricu dimenzije  $n \times m$  napisati funkciju koja vrši sortiranje vrsta matrice, rastuće na osnovu sume elemenata u vrsti.

### 3 Algoritmi pretrage i sortiranja

---

Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz.

*Test 1*

```
Interakcija programa:
  Unesite dimenzije matrice: 3 2
  Unesite elemente matrice po vrstama:
  6 -5
  -4 3
  2 1
  Sortirana matrica je:
  -4 3
  6 -5
  2 1
```

[Rešenje 3.37]

**Zadatak 3.38** Za zadatau kvadratnu matricu dimenzije  $n$  napisati funkciju koja sortira kolone matrice, opadajuće, na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz.

## 3.4 Rešenja

### Rešenje 3.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt
7    opcijom -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz celih brojeva duzine n, trazeci u
10    njemu element x. Pretraga se vrši prostom iteracijom kroz
11    niz. Ako se element pronadje funkcija vraca indeks pozicije
12    na kojoj je pronadjen. Ovaj indeks je uvek nenegativan. Ako
13    element nije pronadjen u nizu, funkcija vraca -1, kao
14    indikator neuspesne pretrage. */
15 int linearna_pretraga(int a[], int n, int x)
16 {
17     int i;
18     for (i = 0; i < n; i++)
19         if (a[i] == x)
20             return i;
21     return -1;
22 }
23
```

```
25 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
    indeks pozicije nadjenog elementa ili -1, ako element nije
    pronadjen */
27 int binarna_pretraga(int a[], int n, int x)
    {
29     int levi = 0;
    int desni = n - 1;
31     int srednji;
    /* Dokle god je indeks levi levo od indeksa desni */
33     while (levi <= desni) {
        /* Racunamo srednji indeks */
35         srednji = (levi + desni) / 2;
        /* Ako je srednji element veci od x, tada se x mora nalaziti
37         u levoj polovini niza */
        if (x < a[srednji])
39             desni = srednji - 1;
        /* Ako je srednji element manji od x, tada se x mora
41         nalaziti u desnoj polovini niza */
        else if (x > a[srednji])
43             levi = srednji + 1;
        else
45             /* Ako je srednji element jednak x, tada smo pronasli x na
                poziciji srednji */
47             return srednji;
    }
49     /* Ako nije pronadjen vracamo -1 */
    return -1;
51 }

53 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
    indeks pozicije nadjenog elementa ili -1, ako element nije
    pronadjen */
55 int interpolaciona_pretraga(int a[], int n, int x)
57 {
    int levi = 0;
59     int desni = n - 1;
    int srednji;
61     /* Dokle god je indeks levi levo od indeksa desni... */
    while (levi <= desni) {
63         /* Ako je element manji od pocetnog ili veci od poslednjeg
            clana u delu niza a[levi],...,a[desni] tada nije u tom
65         delu niza. Ova provera je neophodna, da se ne bi dogodilo
            da se prilikom izracunavanja indeksa srednji izadje izvan
67         opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
69             return -1;
        /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
71         a[levi] i a[desni] jednaki, tada je jasno da je x jednako
            ovim vrednostima, pa vracamo indeks levi (ili indeks
73         desni. Ova provera je neophodna, zato sto bismo inace
            prilikom izracunavanja srednji imali deljenje nulom. */
75         else if (a[levi] == a[desni])
            return levi;
77         /* Racunamo srednji indeks */
        srednji =
79         levi +
```

```

81         ((double) (x - a[levi]) / (a[desni] - a[levi])) *
            (desni - levi);
83     /* Napomena: Indeks srednji je uvek izmedju levi i desni,
84     ali ce verovatno biti blize trazenoj vrednosti nego da
85     smo prosto uvek uzimali srednji element. Ovo se moze
86     porediti sa pretragom recnika: ako neko trazi rec na
87     slovo 'B', sigurno nece da otvori recnik na polovini, vec
88     verovatno negde blize pocetku. */
89     /* Ako je srednji element veci od x, tada se x mora nalaziti
90     u levoj polovini niza */
91     if (x < a[srednji])
92         desni = srednji - 1;
93     /* Ako je srednji element manji od x, tada se x mora
94     nalaziti u desnoj polovini niza */
95     else if (x > a[srednji])
96         levi = srednji + 1;
97     else
98         /* Ako je srednji element jednak x, tada smo pronasli x na
99         poziciji srednji */
100        return srednji;
101    }
102    /* Ako nije pronadjen vratamo -1 */
103    return -1;
104 }
105 /* Funkcija main */
106 int main(int argc, char **argv)
107 {
108     int a[MAX];
109     int n, i, x;
110     struct timespec time1, time2, time3, time4, time5, time6;
111     FILE *f;
112
113     /* Provera argumenata komandne linije */
114     if (argc != 3) {
115         fprintf(stderr,
116             "koriscenje programa: %s dim_niza trazeni_br\n",
117             argv[0]);
118         exit(EXIT_FAILURE);
119     }
120
121     /* Dimenzija niza */
122     n = atoi(argv[1]);
123     if (n > MAX || n <= 0) {
124         fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
125         exit(EXIT_FAILURE);
126     }
127
128     /* Broj koji se trazi */
129     x = atoi(argv[2]);
130
131     /* Elemente niza odredjujemo slucajno, tako da je svaki
132     sledeci veci od prethodnog. srandom() funkcija obezbedjuje
133     novi seed za pozivanje random() funkcije. Kako nas niz ne
134     bi uvek isto izgledao seed smo postavili na tekuce vreme u
135     sekundama od Nove godine 1970. random()%100 daje brojeve
```

```

137     izmedju 0 i 99 */
138     srandom(time(NULL));
139     for (i = 0; i < n; i++)
140         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;

141     /* Linearna pretraga */
142     printf("Linearna pretraga\n");
143     /* Racunamo vreme proteklo od Nove godine 1970 */
144     clock_gettime(CLOCK_REALTIME, &time1);
145     /* Pretrazujemo niz */
146     i = linearna_pretraga(a, n, x);
147     /* Racunamo novo vreme i razlika predstavlja vreme utroseno za
148        lin pretragu */
149     clock_gettime(CLOCK_REALTIME, &time2);
150     if (i == -1)
151         printf("Element nije u nizu\n");
152     else
153         printf("Element je u nizu na poziciji %d\n", i);
154     printf("-----\n");

155     /* Binarna pretraga */
156     printf("Binarna pretraga\n");
157     clock_gettime(CLOCK_REALTIME, &time3);
158     i = binarna_pretraga(a, n, x);
159     clock_gettime(CLOCK_REALTIME, &time4);
160     if (i == -1)
161         printf("Element nije u nizu\n");
162     else
163         printf("Element je u nizu na poziciji %d\n", i);
164     printf("-----\n");

165     /* Interpolaciona pretraga */
166     printf("Interpolaciona pretraga\n");
167     clock_gettime(CLOCK_REALTIME, &time5);
168     i = interpolaciona_pretraga(a, n, x);
169     clock_gettime(CLOCK_REALTIME, &time6);
170     if (i == -1)
171         printf("Element nije u nizu\n");
172     else
173         printf("Element je u nizu na poziciji %d\n", i);
174     printf("-----\n");

175     /* Upisujemo podatke o izvrsavanju programa u log fajl */
176     if ((f = fopen("vremena.txt", "a")) == NULL) {
177         fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
178         exit(EXIT_FAILURE);
179     }

180     fprintf(f, "Dimenzija niza od %d elemenata.\n", n);
181     fprintf(f, "\tLinearna pretraga:%10ld ns\n",
182             (time2.tv_sec - time1.tv_sec) * 1000000000 +
183             time2.tv_nsec - time1.tv_nsec);
184     fprintf(f, "\tBinarna: %19ld ns\n",
185             (time4.tv_sec - time3.tv_sec) * 1000000000 +
186             time4.tv_nsec - time3.tv_nsec);
187     fprintf(f, "\tInterpolaciona: %12ld ns\n",
188             (time6.tv_sec - time5.tv_sec) * 1000000000 +
189             time6.tv_nsec - time5.tv_nsec);
190
191

```

### 3 Algoritmi pretrage i sortiranja

```
193         (time6.tv_sec - time5.tv_sec) * 1000000000 +
            time6.tv_nsec - time5.tv_nsec);
195     /* Zatvaramo datoteku */
    fclose(f);
197
    return 0;
199 }
```

#### Rešenje 3.2

```
#include <stdio.h>
2
int lin_pretgraga_rek_sufiks(int a[], int n, int x)
4 {
    int tmp;
    /* Izlaz iz rekurzije */
    if (n <= 0)
        return -1;
    /* Ako je prvi element trazeni */
    if (a[0] == x)
        return 0;
    /* Pretraga ostatka niza */
    tmp = lin_pretgraga_rek_sufiks(a + 1, n - 1, x);
    return tmp < 0 ? tmp : tmp + 1;
12 }

int lin_pretgraga_rek_prefiks(int a[], int n, int x)
18 {
    /* Izlaz iz rekurzije */
    if (n <= 0)
        return -1;
    /* Ako je poslednji element trazeni */
    if (a[n - 1] == x)
        return n - 1;
    /* Pretraga ostatka niza */
    return lin_pretgraga_rek_prefiks(a, n - 1, x);
26 }

int bin_pretgraga_rek(int a[], int l, int d, int x)
30 {
    int srednji;
    if (l > d)
        return -1;
    /* Srednja pozicija na kojoj se trazi vrednost x */
    srednji = (l + d) / 2;
    /* Ako je sredisnji element trazeni */
    if (a[srednji] == x)
        return srednji;
    /* Ako je trazeni broj veci od srednjeg, pretrazujemo desnu
    polovinu niza */
    if (a[srednji] < x)
        return bin_pretgraga_rek(a, srednji + 1, d, x);
    /* Ako je trazeni broj manji od srednjeg, pretrazujemo levu
    polovinu niza */
44 }
```



```

else
46     return bin_pretgraga_rek(a, l, srednji - 1, x);
}
48
50 int interp_pretgraga_rek(int a[], int l, int d, int x)
{
52     int p;
54     if (x < a[l] || x > a[d])
56         return -1;
58     if (a[d] == a[l])
60         return 1;
62     /* Pozicija na kojoj se trazi vrednost x */
64     p = 1 + (d - l) * (x - a[l]) / (a[d] - a[l]);
66     if (a[p] == x)
68         return p;
70     if (a[p] < x)
72         return interp_pretgraga_rek(a, p + 1, d, x);
74     else
76         return interp_pretgraga_rek(a, l, p - 1, x);
78 }
80
82 #define MAX 1024
84
86 int main()
88 {
90     int a[MAX];
92     int x;
94     int i, indeks;
96
98     /* Ucitavamo trazeni broj */
100     printf("Unesite trazeni broj: ");
    scanf("%d", &x);

```

```

/* Ucitavamo elemente niza sve do kraja ulaza - ocekujemo da
korisnik pritisne CTRL+D za naznaku kraja */
i = 0;
printf("Unesite sortiran niz elemenata: ");
while (scanf("%d", &a[i]) == 1) {
    i++;
}

/* Linearna pretraga */
printf("Linearna pretraga\n");
indeks = lin_pretgraga_rek_sufiks(a, i, x);
if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
else
    printf("Pozicija elementa je %d.\n", indeks);

/* Binarna pretraga */
printf("Binarna pretraga\n");
indeks = bin_pretgraga_rek(a, 0, i - 1, x);
if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
else

```

### 3 Algoritmi pretrage i sortiranja

```
102     printf("Pozicija elementa je %d.\n", indeks);
103
104     /* Interpolaciona pretraga */
105     printf("Interpolaciona pretraga\n");
106     indeks = interp_pretgraga_rek(a, 0, i - 1, x);
107     if (indeks == -1)
108         printf("Element se ne nalazi u nizu.\n");
109     else
110         printf("Pozicija elementa je %d.\n", indeks);
111
112     return 0;
113 }
```

#### Rešenje 3.3

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_STUDENATA 128
6  #define MAX_DUZINA 16
7
8  /* 0 svakom studentu imamo 3 informacije i njih objedinjujemo u
9   strukturu kojom cemo predstavljati svakog studenta. */
10 typedef struct {
11     /* Indeks mora biti tipa long jer su podaci u datoteci
12     preveliki za int, npr. 20140123 */
13     long indeks;
14     char ime[MAX_DUZINA];
15     char prezime[MAX_DUZINA];
16 } Student;
17
18 /* Ucitani niz studenata ce biti sortirani prema indeksu, jer cemo
19 ih, redom, kako citamo smestati u niz, a u datoteci su vec
20 smesteni sortirani rastuce prema broju indeksa. Iz tog
21 razloga pretragu po indeksu cemo vrsiti binarnom pretragom,
22 dok pretragu po prezimenu moramo vrsiti linearno, jer nemamo
23 garancije da postoji uredjenje po prezimenu. */
24
25 /* Funkcija trazi u sortiranom nizu studenata a[] duzine n
26 studenta sa indeksom x. Vraca indeks pozicije nadjenog clana
27 niza ili -1, ako element nije pronadjen */
28 int binarna_pretraga(Student a[], int n, long x)
29 {
30     int levi = 0;
31     int desni = n - 1;
32     int srednji;
33     /* Dokle god je indeks levi levo od indeksa desni */
34     while (levi <= desni) {
35         /* Racunamo srednji indeks */
36         srednji = (levi + desni) / 2;
37         /* Ako je srednji element veci od x, tada se x mora nalaziti
38            u levoj polovini niza */
39         if (x < a[srednji].indeks)
40             desni = srednji - 1;
```

```

41     /* Ako je srednji element manji od x, tada se x mora
        nalaziti u desnoj polovini niza */
43     else if (x > a[srednji].indeks)
        levi = srednji + 1;
45     else
        /* Ako je srednji element jednak x, tada smo pronasli x na
47         poziciji srednji */
        return srednji;
49 }
    /* Ako nije pronadjen vratamo -1 */
51     return -1;
}

53
/* Linearnom pretragom niza studenata trazimo prezime x */
55 int linearna_pretraga(Student a[], int n, char x[])
{
57     int i;
    for (i = 0; i < n; i++)
59         /* Poredimo prezime i-tog studenta i poslato x */
        if (strcmp(a[i].prezime, x) == 0)
61             return i;
    return -1;
63 }

65 /* Main funkcija mora imate argumente jer se ime datoteke dobija
    kao argument komandne linije */
67 int main(int argc, char *argv[])
{
69     /* Ucitacemo redom sve studente iz datoteke u niz. */
    Student dosije[MAX_STUDENATA];
71     FILE *fin = NULL;
    int i;
73     int br_studenata = 0;
    long trazen_indeks = 0;
75     char trazeno_prezime[MAX_DUZINA];

77     /* Proveravamo da li nam je korisnik prilikom poziva prosledio
        ime datoteke sa informacijama o studentima */
79     if (argc != 2) {
        fprintf(stderr,
81             "Greska: Program se poziva sa %s ime_datoteke\n",
                argv[0]);
83         exit(EXIT_FAILURE);
    }

85
    /* Otvaramo datoteku */
87     fin = fopen(argv[1], "r");
    if (fin == NULL) {
89         fprintf(stderr,
                "Neuspesno otvaranje datoteke %s za citanje\n",
91             argv[1]);
        exit(EXIT_FAILURE);
93     }

95     /* Citamo sve dok imamo red sa informacijama o studentu */
    i = 0;

```

### 3 Algoritmi pretrage i sortiranja

```
97  while (1) {
    if (i == MAX_STUDENATA)
99      break;
    if (fscanf
101        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
          dosije[i].prezime) != 3)
103        break;
    i++;
105  }
  br_studenata = i;

107  /* Nakon citanja datoteka nam vise nije neophodna i odmah je
109     zatvaramo */
  fclose(fin);

111  /* Unos indeksa koji se binarno trazi u nizu */
113  printf("Unesite indeks studenta cije informacije zelite: ");
  scanf("%ld", &trazen_indeks);
115  i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
  /* Rezultat binarne pretrage */
117  if (i == -1)
    printf("Ne postoji student sa indeksom %ld\n",
119          trazen_indeks);
  else
121    printf("Indeks: %ld, Ime i prezime: %s %s\n",
          dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
123

  /* Unos prezimena koje se linearno trazi u nizu */
125  printf("Unesite prezime studenta cije informacije zelite: ");
  scanf("%s", trazeno_prezime);
127  i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
  /* Rezultat linearne pretrage */
129  if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n",
131          trazeno_prezime);
  else
133    printf("Indeks: %ld, Ime i prezime: %s %s\n",
          dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135

  return 0;
137 }
```

#### Rešenje 3.4

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

5  #define MAX_STUDENATA 128
   #define MAX_DUZINA 16
7
   typedef struct {
9     long indeks;
     char ime[MAX_DUZINA];
11    char prezime[MAX_DUZINA];
```

```

} Student;

13
14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
15                                long x)
16 {
17     /* Ako je indeks elementa na levom kraju veci od indeksa
18        elementa na desnom kraju dela niza koji se pretrazuje, onda
19        zapravo pretrazujemo prazan deo niza. U praznom nizu nema
20        elementa koji trazimo i zato vracamo -1 */
21     if (levi > desni)
22         return -1;
23     /* Racunamo indeks srednjeg elementa */
24     int srednji = (levi + desni) / 2;
25     /* Da li je srednji, bas onaj kog trazimo? */
26     if (a[srednji].indeks == x) {
27         return srednji;
28     }
29     /* Ako je trazeni indeks manji od indeksa srednjeg, onda
30        potragu nastavljamo u levoj polovini niza jer znamo da je
31        niz sortiran po indeksu u rastucem poretku. */
32     if (x < a[srednji].indeks)
33         return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34     /* Inace ga treba traziti u desnoj polovini */
35     else
36         return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37 }

38
39 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
41     /* Ako je niz prazan, vracamo -1, jer ga ne mozemo naci */
42     if (n == 0)
43         return -1;
44     /* Kako trazimo prvog studenta sa trazenim prezimenom,
45        pocinjemo sa prvim studentom u nizu. */
46     if (strcmp(a[0].prezime, x) == 0)
47         return 0;
48     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
49     return i >= 0 ? 1 + i : -1;
50 }

51
52 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
53 {
54     /* Ako je niz prazan, vracamo -1, jer ga ne mozemo naci */
55     if (n == 0)
56         return -1;
57     /* Kako trazimo poslednjeg studenta sa trazenim prezimenom,
58        pocinjemo sa poslednjim studentom u nizu. */
59     if (strcmp(a[n - 1].prezime, x) == 0)
60         return n - 1;
61     return linearna_pretraga_rekurzivna(a, n - 1, x);
62 }

63
64 /* Main funkcija mora imate argumente jer se Ime datoteke dobija
65    kao argument komandne linije */
66 int main(int argc, char *argv[])
67 {

```

### 3 Algoritmi pretrage i sortiranja

```
69  /* Ucitacemo redom sve studente iz datoteke u niz. */
Student dosije[MAX_STUDENATA];
FILE *fin = NULL;
71  int i;
int br_studenata = 0;
73  long trazen_indeks = 0;
char trazeno_prezime[MAX_DUZINA];
75
/* Proveravamo da li nam je korisnik prilikom poziva prosledio
77  ime datoteke sa informacijama o studentima */
if (argc != 2) {
79  fprintf(stderr,
          "Greska: Program se poziva sa %s ime_datoteke\n",
81  argv[0]);
exit(EXIT_FAILURE);
83 }

85 /* Otvaramo datoteku */
fin = fopen(argv[1], "r");
87 if (fin == NULL) {
fprintf(stderr,
89  "Neuspesno otvaranje datoteke %s za citanje\n",
argv[1]);
91  exit(EXIT_FAILURE);
}

93
/* Citamo sve dok imamo red sa informacijama o studentu */
95 i = 0;
while (1) {
97  if (i == MAX_STUDENATA)
break;
99  if (fscanf
      (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
101  dosije[i].prezime) != 3)
break;
103  i++;
}
105 br_studenata = i;

107 /* Nakon citanja datoteka nam vise nije neophodna i odmah je
zatvaramo */
109 fclose(fin);

111 /* Unos indeksa koji se binarno trazi u nizu */
printf("Unesite indeks studenta cije informacije zelite: ");
113 scanf("%ld", &trazen_indeks);
i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
115 trazen_indeks);

if (i == -1)
117  printf("Ne postoji student sa indeksom %ld\n",
trazen_indeks);
119 else
printf("Indeks: %ld, Ime i prezime: %s %s\n",
121  dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

123 printf("Unesite prezime studenta cije informacije zelite: ");
```

```

scanf("%s", trazeno_prezime);
125 i = linearna_pretraga_rekurzivna(dosije, br_studenata,
                                trazeno_prezime);
127 if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n",
129         trazeno_prezime);
else
131     printf
        ("Poslednji takav student:\nIndeks: %ld, Ime i prezime: %s %
133         s\n",
        dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135 return 0;
}

```

### Rešenje 3.5

```

#include <stdio.h>
2 #include <string.h>
#include <math.h>
4 #include <stdlib.h>

6 /* Struktura koja opisuje tacku u ravni */
typedef struct Tacka {
8     float x;
    float y;
10 } Tacka;

12 /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
    pocetka (0,0) */
14 float rastojanje(Tacka A)
{
16     return sqrt(A.x * A.x + A.y * A.y);
}

18 /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u
    nizu zadatih tacaka t dimenzije n */
20 Tacka najbliza_koordinatnom(Tacka t[], int n)
22 {
    Tacka najbliza;
24     int i;
    najbliza = t[0];
26     for (i = 1; i < n; i++) {
        if (rastojanje(t[i]) < rastojanje(najbliza)) {
28             najbliza = t[i];
        }
30     }
    return najbliza;
32 }

34 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
    tacaka t dimenzije n */
36 Tacka najbliza_x_osi(Tacka t[], int n)
{
38

```

```
    Tacka najbliza;
40  int i;
    najbliza = t[0];
42  for (i = 1; i < n; i++) {
        if (fabs(t[i].x) < fabs(najbliza.x)) {
44      najbliza = t[i];
        }
46  }
    return najbliza;
48 }

50 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih
    tacaka t dimenzije n */
52 Tacka najbliza_y_osi(Tacka t[], int n)
{
54     Tacka najbliza;
56     int i;
    najbliza = t[0];
    for (i = 1; i < n; i++) {
58         if (fabs(t[i].y) < fabs(najbliza.y)) {
            najbliza = t[i];
60         }
    }
62     return najbliza;
}

64 #define MAX 1024

66 int main(int argc, char *argv[])
68 {
    FILE *ulaz;
70     Tacka tacke[MAX];
    Tacka najbliza;
72     int i, n;

74     /* Ocekujemo da korisnik unese barem ime izvrsne verzije
        programa i ime datoteke sa tackama */
76     if (argc < 2) {
        fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
        return EXIT_FAILURE;
80     }

82     /* Otvaramo datoteku za citanje */
    ulaz = fopen(argv[1], "r");
84     if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
        return EXIT_FAILURE;
88     }

90     /* Sve dok ima tacaka u datoteci, smestamo ih u niz sa
        tackama; i predstavlja indeks tekuce tacke */
92     i = 0;
    while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94         i++;
```



```

    }
96   n = i;

98   /* Proveravamo koji su dodatni argumenti komandne linije. Ako
       nema dodatnih argumenata */
100  if (argc == 2)
       /* Trazimo najblizu tacku u odnosu na koordinatni pocetak */
102     najbliza = najbliza_koordinatnom(tacke, n);
       /* Inace proveravamo koji je dodatni argument. Ako je u
104     pitanju opcija -x */
       else if (strcmp(argv[2], "-x") == 0)
106         /* Racunamo rastojanje u odnosu na x osu */
           najbliza = najbliza_x_osi(tacke, n);
108     /* Ako je u pitanju opcija -y */
       else if (strcmp(argv[2], "-y") == 0)
110         /* Racunamo rastojanje u odnosu na y osu */
           najbliza = najbliza_y_osi(tacke, n);
112     else {
       /* Ako nije zadata opcija -x ili -y, ispisujemo obavestenje
114         za korisnika i prekidamo izvršavanje programa */
       fprintf(stderr, "Pogresna opcija\n");
116       return EXIT_FAILURE;
       }

118     /* Stampamo koordinate trazene tacke */
120     printf("%g %g\n", najbliza.x, najbliza.y);

122     /* Zatvaramo datoteku */
       fclose(ulaz);

124

       return 0;
126 }

```

### Rešenje 3.6

```

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

3

   /* Tacnost */
5  #define EPS 0.001

7  int main()
   {
9     double l, d, s;

11    /* Posto je u pitanju interval [0, 2] leva granica je 0, a
       desna 2 */
13    l = 0;
       d = 2;

15

       /* Sve dok ne pronadjemo trazenu vrednost argumenta */
17    while (1) {
       /* Pronalazimo sredinu intervala */
19       s = (l + d) / 2;
       /* Ako je vrednost kosinusa u ovoj tacki manja od zadate

```

### 3 Algoritmi pretrage i sortiranja

```
21         tacnosti, prekidamo pretragu */
22     if (fabs(cos(s)) < EPS) {
23         break;
24     }
25     /* Ako je nula u levom delu intervala, nastavljamo pretragu
26        na intervalu [l, s] */
27     if (cos(l) * cos(s) < 0)
28         d = s;
29     else
30         /* Inace, nastavljamo pretragu na intervalu [s, d] */
31         l = s;
32 }
33
34 /* Stampamo vrednost trazene tacke */
35 printf("%g\n", s);
36
37 return 0;
38 }
```

#### Rešenje 3.7

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

2 int prvi_veci_od_nule(int niz[], int n)
3 {
4     /* Granice pretrage */
5     int l = 0, d = n - 1;
6     int s;
7     /* Sve dok je leva manja od desne granice */
8     while (l <= d) {
9         /* Racunamo sredisnju poziciju */
10        s = (l + d) / 2;
11        /* Ako je broj na toj poziciji veci od nule a eventualni
12           njegov prethodnik manji ili jednak nuli */
13        if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
14            return s;
15        /* Pretrazujemo desnu polovinu niza */
16        if (niz[s] <= 0)
17            l = s + 1;
18        /* Pretrazujemo levu polovinu binarnog zapisa */
19        else
20            d = s - 1;
21    }
22    return -1;
23 }
24
25 #define MAX 256
26
27 int main()
28 {
29     int niz[MAX];
30     int n = 0;
31
32     /* Unos niza */
33 }
```

```

printf("Unesi rastuce sortiran niz celih brojeva: ");
36 while (scanf("%d", &niz[n]) == 1)
    n++;
38
/* Stampanje rezultata */
40 printf("%d\n", prvi_veci_od_nule(niz, n));
42
return 0;
}

```

### Rešenje 3.8

```

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

4 int prvi_manji_od_nule(int niz[], int n)
{
6     /* Granice pretrage */
    int l = 0, d = n - 1;
8     int s;
    /* Sve dok je leva manja od desne granice */
10 while (l <= d) {
    /* Racunamo sredisnju poziciju */
12     s = (l + d) / 2;
    /* Ako je broj na toj poziciji manji od nule a eventualni
14     njegov prethodnik veci ili jednak nuli */
    if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
16         return s;
    /* Pretrazujemo desnu polovinu niza */
18     if (niz[s] >= 0)
        l = s + 1;
20     /* Pretrazujemo levu polovinu binarnog zapisa */
    else
22         d = s - 1;
    }
24     return -1;
}

26 #define MAX 256

28 int main()
30 {
    int niz[MAX];
32     int n = 0;

34     /* Unos niza */
    printf("Unesi opadajuce sortiran niz celih brojeva: ");
36 while (scanf("%d", &niz[n]) == 1)
    n++;
38
    /* Stampanje rezultata */
40 printf("%d\n", prvi_manji_od_nule(niz, n));
42
    return 0;
}

```

#### Rešenje 3.9

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 unsigned int logaritam_a(unsigned int x)
5 {
6     /* Izlaz iz rekurzije */
7     if (x == 1)
8         return 0;
9     /* Rekurzivni korak */
10    return 1 + logaritam_a(x >> 1);
11 }
12
13 unsigned int logaritam_b(unsigned int x)
14 {
15     /* Binarnom pretragom trazimo jedinicu u binarnom zapisu broja
16     x najvece vaznosti, tj. najlevlju. Pretragu radimo od
17     pozicije 0 do 31 */
18     int d = 0, l = sizeof(unsigned int) * 8 - 1;
19     int s;
20     /* Sve dok je desna granica pretrage desnije od leve */
21     while (d <= l) {
22         /* Racunamo sredisnju poziciju */
23         s = (l + d) / 2;
24         /* Proveravamo da li je na toj poziciji trazena jedinica */
25         if ((1 << s) <= x && (1 << (s + 1)) > x)
26             return s;
27         /* Pretrazujemo desnu polovinu binarnog zapisa */
28         if ((1 << s) > x)
29             l = s - 1;
30         /* Pretrazujemo levu polovinu binarnog zapisa */
31         else
32             d = s + 1;
33     }
34     return s;
35 }
36
37 int main()
38 {
39     unsigned int x;
40
41     /* Unos podatka */
42     printf("Unesi pozitivan ceo broj: ");
43     scanf("%u", &x);
44
45     /* Provera da li je uneti broj pozitivan */
46     if (x == 0) {
47         fprintf(stderr, "Logaritam od nule nije definisan\n");
48         exit(EXIT_FAILURE);
49     }
50
51     /* Ispis povratnih vrednosti funkcija */
52     printf("%u %u\n", logaritam_a(x), logaritam_b(x));
53
54     return 0;
```

}

## Rešenje 3.12

```

1  #include<stdio.h>
2  #define MAX 256

4  /* Iterativna verzija funkcije koja sortira niz celih brojeva,
   primenom algoritma Selection Sort */
6  void selectionSort(int a[], int n)
   {
8     int i, j;
     int min;
10    int pom;

12    /* U svakoj iteraciji ove petlje se pronalazi najmanji element
       medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
14    poziciju i, dok se element na poziciji i premesta na
       poziciju min, na kojoj se nalazio najmanji od gore
16    navedenih elemenata. */
     for (i = 0; i < n - 1; i++) {
18         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
            nalazi najmanji od elemenata a[i],...,a[n-1]. */
20         min = i;
         for (j = i + 1; j < n; j++)
22             if (a[j] < a[min])
                 min = j;
24         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
            samo ako su (i) i min razliciti, inace je nepotrebno. */
26         if (min != i) {
             pom = a[i];
28             a[i] = a[min];
             a[min] = pom;
30         }
     }
32 }

34 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja
   u sortiranom nizu celih brojeva */
36 int najmanje_rastojanje(int a[], int n)
   {
38     int i, min;
     min = a[1] - a[0];
40     for (i = 2; i < n; i++)
         if (a[i] - a[i - 1] < min)
42             min = a[i] - a[i - 1];
     return min;
44 }

46
48 int main()
   {
     int i, a[MAX];
50
     /* Ucitavaju se elementi niza sve do kraja ulaza */

```

### 3 Algoritmi pretrage i sortiranja

```
52  i = 0;
    printf("Unesite elemente niza: ");
54  while (scanf("%d", &a[i]) != EOF)
        i++;
56
    /* Sortiranje */
58  selectionSort(a, i);

60  /* Ispis rezultata */
    printf("%d\n", najmanje_rastojanje(a, i));
62
    return 0;
64 }
```

#### Rešenje 3.13

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

4  #define MAX_DIM 128

6  /* Funkcija za sortiranje niza */
void selectionSort(char s[], int n)
8  {
    int i, j, min;
    char pom;
10  for (i = 0; i < n; i++) {
12      min = i;
        for (j = i + 1; j < n; j++)
14          if (s[j] < s[min])
                min = j;
16      if (min != i) {
                pom = s[i];
                s[i] = s[min];
                s[min] = pom;
20      }
    }
22 }

24 /* Funkcija vraca: 1 - ako jesu anagrami; 0 - inace.
    pretpostavlja se da su niske s i t sortirane */
26 int anagrami(char s[], char t[], int n_s, int n_t)
    {
28     int i, n;

30     /* Ako dve niske imaju razlicit broj elemenata onda nisu
        anagrami */
32     if (n_s != n_t)
        return 0;

34     n = n_s;

36     /* Dve sortirane niske su anagrami akko su jednake */
38     for (i = 0; i < n; i++)
        if (s[i] != t[i])
```

```

40     return 0;
    return 1;
42 }

44 int main()
{
46     char s[MAX_DIM], t[MAX_DIM];
    int n_s, n_t;

48     /* Ucitavamo dve niske sa ulaza */
50     printf("Unesite prvu nisku: ");
    scanf("%s", s);
52     printf("Unesite drugu nisku: ");
    scanf("%s", t);

54     /* Odredjujemo duzinu niski */
56     n_s = strlen(s);
    n_t = strlen(t);

58     /* Sortiramo niske */
60     selectionSort(s, n_s);
    selectionSort(t, n_t);

62     /* Proveravamo da li su niske anagrami */
64     if (anagrami(s, t, n_s, n_t))
        printf("jesu\n");
66     else
        printf("nisu\n");
68     return 0;
}

```

### Rešenje 3.14

```

#include<stdio.h>
2 #define MAX_DIM 256

4 /* Funkcija za sortiranje niza */
void selectionSort(int s[], int n)
6 {
    int i, j, min;
    char pom;
    for (i = 0; i < n; i++) {
10         min = i;
        for (j = i + 1; j < n; j++)
12             if (s[j] < s[min])
                min = j;
14         if (min != i) {
            pom = s[i];
16             s[i] = s[min];
            s[min] = pom;
18         }
    }
20 }

22 /* Funkcija za odredjivanje onog elementa sortiranog niza koji

```

### 3 Algoritmi pretrage i sortiranja

```

    se najviše puta pojavio u tom nizu */
24 int najvise_puta(int a[], int n)
{
26     int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
    /* Za i-ti element izracunavamo koliko se puta pojavio u nizu */
28     for (i = 0; i < n; i = j) {
        br_pojava = 1;
30         for (j = i + 1; j < n && a[i] == a[j]; j++)
            br_pojava++;
32         /* Ispitujemo da li se do tog trenutka i-ti element pojavio
            najviše puta u nizu */
34         if (br_pojava > max_br_pojava) {
            max_br_pojava = br_pojava;
36             i_max_pojava = i;
        }
38     }
    /* Vracamo element koji se najviše puta pojavio u nizu */
40     return a[i_max_pojava];
}

42
int main()
44 {
    int a[MAX_DIM], i;
46
    /* Ucitavaju se elementi niza sve do kraja ulaza */
48     i = 0;
    printf("Unesite elemente niza: ");
50     while (scanf("%d", &a[i]) != EOF)
        i++;
52
    /* Niz se sortira */
54     selectionSort(a, i);

    /* Odredjuje se broj koji se najviše puta pojavio u nizu */
56     printf("%d\n", najvise_puta(a, i));
58
    return 0;
60 }
```

#### Rešenje 3.15

```

#include<stdio.h>
2 #define MAX_DIM 256

4 /* Funkcija za sortiranje niza */
void selectionSort(int a[], int n)
6 {
    int i, j, min, pom;
8     for (i = 0; i < n - 1; i++) {
        min = i;
10        for (j = i + 1; j < n; j++)
            if (a[j] < a[min])
12                min = j;
        if (min != i) {
14            pom = a[i];
```



```
16     a[i] = a[min];
17     a[min] = pom;
18 }
19 }
20
21 /* Funkcija za binarnu pretragu niza. funkcija vraca: 1 - ako se
22    element x nalazi u nizu; 0 - inace. pretpostavlja se da je
23    niz sortiran u rastucem poretku */
24 int binarna_pretraga(int a[], int n, int x)
25 {
26     int levi = 0, desni = n - 1, srednji;
27
28     while (levi <= desni) {
29         srednji = (levi + desni) / 2;
30         if (a[srednji] == x)
31             return 1;
32         else if (a[srednji] > x)
33             desni = srednji - 1;
34         else if (a[srednji] < x)
35             levi = srednji + 1;
36     }
37     return 0;
38 }
39
40 int main()
41 {
42     int a[MAX_DIM], n = 0, zbir, i;
43
44     /* Ucitava se trazeni zbir */
45     printf("Unesite trazeni zbir: ");
46     scanf("%d", &zbir);
47
48     /* Ucitavaju se elementi niza sve do kraja ulaza */
49     i = 0;
50     printf("Unesite elemente niza: ");
51     while (scanf("%d", &a[i]) != EOF)
52         i++;
53     n = i;
54
55     /* Sortira se niz */
56     selectionSort(a, n);
57
58     for (i = 0; i < n; i++)
59         /* Za i-ti element niza binarno se pretrazuje da li se u
60            ostatku niza nalazi element koji sabran sa njim ima
61            ucitanu vrednost zbira */
62         if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
63             printf("da\n");
64             return 0;
65         }
66     printf("ne\n");
67
68     return 0;
69 }
```

#### Rešenje 3.16

```
1 #include <stdio.h>
2 #define MAX_DIM 256
3
4 int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
5           int dim3)
6 {
7     int i = 0, j = 0, k = 0;
8     /* U slucaju da je dimenzija treceg niza manja od neophodne,
9        funkcija vraca -1 */
10    if (dim3 < dim1 + dim2)
11        return -1;
12
13    /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja
14       jednog od njih */
15    while (i < dim1 && j < dim2) {
16        if (niz1[i] < niz2[j])
17            niz3[k++] = niz1[i++];
18        else
19            niz3[k++] = niz2[j++];
20    }
21    /* Ostatak prvog niza prepisujemo u treci */
22    while (i < dim1)
23        niz3[k++] = niz1[i++];
24
25    /* Ostatak drugog niza prepisujemo u treci */
26    while (j < dim2)
27        niz3[k++] = niz2[j++];
28    return dim1 + dim2;
29 }
30
31 int main()
32 {
33     int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
34     int i = 0, j = 0, k, dim3;
35
36     /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
37        Pretpostavka je da na ulazu nece biti vise od MAX_DIM
38        elemenata */
39     printf("Unesite elemente prvog niza: ");
40     while (1) {
41         scanf("%d", &niz1[i]);
42         if (niz1[i] == 0)
43             break;
44         i++;
45     }
46     printf("Unesite elemente drugog niza: ");
47     while (1) {
48         scanf("%d", &niz2[j]);
49         if (niz2[j] == 0)
50             break;
51         j++;
52     }
53
54     /* Poziv trazene funkcije */
```

```

56     dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);
57
58     /* Ispis niza */
59     for (k = 0; k < dim3; k++)
60         printf("%d ", niz3[k]);
61     printf("\n");
62
63     return 0;
64 }

```

### Rešenje 3.17

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main(int argc, char *argv[])
6  {
7      FILE *fin1 = NULL, *fin2 = NULL;
8      FILE *fout = NULL;
9      char ime1[11], ime2[11];
10     char prezime1[16], prezime2[16];
11     int kraj1 = 0, kraj2 = 0;
12
13     /* Ako nema dovoljno argumenata komandne linije */
14     if (argc < 3) {
15         fprintf(stderr,
16             "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
17         exit(EXIT_FAILURE);
18     }
19
20     /* Otvaramo datoteku zadatu prvim argumentom komandne linije */
21     fin1 = fopen(argv[1], "r");
22     if (fin1 == NULL) {
23         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n",
24             argv[1]);
25         exit(EXIT_FAILURE);
26     }
27
28     /* Otvaramo datoteku zadatu drugim argumentom komandne linije */
29     fin2 = fopen(argv[2], "r");
30     if (fin2 == NULL) {
31         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n",
32             argv[2]);
33         exit(EXIT_FAILURE);
34     }
35
36     /* Otvaranje datoteke za upis rezultata */
37     fout = fopen("ceo-tok.txt", "w");
38     if (fout == NULL) {
39         fprintf(stderr,
40             "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
41         ;
42         exit(EXIT_FAILURE);
43     }
44 }

```

```
44  /* Citamo narednog studenta iz prve datoteke */
45  if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
46      kraj1 = 1;

48  /* Citamo narednog studenta iz druge datoteke */
49  if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
50      kraj2 = 1;

52  /* Sve dok nismo dosli do kraja neke datoteke */
53  while (!kraj1 && !kraj2) {
54      if (strcmp(ime1, ime2) < 0) {
55          /* Ime i prezime iz prve datoteke je leksikografski
56             ranije, upisujemo ga u izlaznu datoteku */
57          fprintf(fout, "%s %s\n", ime1, prezime1);
58          /* Citamo narednog studenta iz prve datoteke */
59          if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
60              kraj1 = 1;
61      } else {
62          /* Ime i prezime iz druge datoteke je leksikografski
63             ranije, upisujemo ga u izlaznu datoteku */
64          fprintf(fout, "%s %s\n", ime2, prezime2);
65          /* Citamo narednog studenta iz druge datoteke */
66          if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
67              kraj2 = 1;
68      }
69  }

70  /* Ako smo iz prethodne petlje izasli zato sto se doslo do
71     kraja druge datoteke, onda ima jos imena u prvoj datoteci,
72     i prepisujemo ih, redom, jer su vec sortirani po imenu. */
73  while (!kraj1) {
74      fprintf(fout, "%s %s\n", ime1, prezime1);
75      if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
76          kraj1 = 1;
77  }

78  /* Ako smo iz prve petlje izasli zato sto se doslo do kraja
79     prve datoteke, onda ima jos imena u drugoj datoteci, i
80     prepisujemo ih, redom, jer su vec sortirani po imenu. */
81  while (!kraj2) {
82      fprintf(fout, "%s %s\n", ime2, prezime2);
83      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
84          kraj2 = 1;
85  }

86  /* Zatvaramo datoteke */
87  fclose(fin1);
88  fclose(fin2);
89  fclose(fout);

90  return 0;
91 }
```

#### Rešenje 3.18

```

1  /* Datoteka sort.h */
2  #ifndef __SORT_H__
3  #define __SORT_H__ 1
4
5  /* Selection sort */
6  void selectionsort(int a[], int n);
7  /* Insertion sort */
8  void insertionsort(int a[], int n);
9  /* Bubble sort */
10 void bubblesort(int a[], int n);
11 /* Shell sort */
12 void shellsort(int a[], int n);
13 /* Merge sort */
14 void mergesort(int a[], int l, int r);
15 /* Quick sort */
16 void quicksort(int a[], int l, int r);
17
18 #endif

```

```

1  /* Datoteka sort.c */
2
3  #include "sort.h"
4
5  /* Funkcija sortira niz celih brojeva metodom sortiranja
6   izborom. Ideja algoritma je sledeca: U svakoj iteraciji
7   pronalazimo najmanji element i postavljamo ga na pocetak
8   niza. Dakle, u prvoj iteraciji, pronalazimo najmanji element,
9   i dovodimo ga na nulto mesto u nizu. U i-toj iteraciji
10  najmanjih i elemenata su vec na svojim pozicijama, pa od i+1
11  do n-1 elementa trazimo najmanji, koji dovodimo na i+1
12  poziciju. */
13 void selectionsort(int a[], int n)
14 {
15     int i, j;
16     int min;
17     int pom;
18
19     /* U svakoj iteraciji ove petlje se pronalazi najmanji element
20      medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
21      poziciju i, dok se element na poziciji i premesta na
22      poziciju min, na kojoj se nalazio najmanji od gore
23      navedenih elemenata. */
24     for (i = 0; i < n - 1; i++) {
25         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
26          nalazi najmanji od elemenata a[i],...,a[n-1]. */
27         min = i;
28         for (j = i + 1; j < n; j++)
29             if (a[j] < a[min])
30                 min = j;
31
32         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
33          samo ako su (i) i min razliciti, inace je nepotrebno. */
34         if (min != i) {
35             pom = a[i];
36             a[i] = a[min];

```

### 3 Algoritmi pretrage i sortiranja

```

    a[min] = pom;
38 }
    }
40 }
42
44 /* Funkcija sortira niz celih brojeva metodom sortiranja
    umetanjem. Ideja algoritma je sledeca: neka je na pocetku
46 i-te iteracije niz prvih i elemenata (a[0],a[1],...,a[i-1])
    sortirano. U i-toj iteraciji zelimo da element a[i] umetnemo
48 na pravu poziciju medju prvih i elemenata tako da dobijemo
    niz duzine i+1 koji je sortiran. Ovo radimo tako sto i-ti
50 element najpre uporedimo sa njegovim prvim levim susedom
    (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i
52 niz a[0],a[1],...,a[i] je sortiran, pa mozemo preci na
    sledecu iteraciju. Ako je a[i-1] vece, tada zamenjujemo a[i]
54 i a[i-1], a zatim proveravamo da li je potrebno dalje
    potiskivanje elementa u levo, poredeci ga sa njegovim novim
56 levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
    pravo mesto u nizu. */
58 void insertionsort(int a[], int n)
    {
60     int i, j;

62     /* Na pocetku iteracije pretpostavljamo da je niz
        a[0],...,a[i-1] sortiran */
64     for (i = 1; i < n; i++) {

66         /* U ovoj petlji redom potiskujemo element a[i] u levo
            koliko je potrebno, dok ne zauzme pravo mesto, tako da
68            niz a[0],...,a[i] bude sortiran. Indeks j je trenutna
            pozicija na kojoj se element koji umecemo nalazi. Petlja
70            se zavrшава ili kada element dodje do levog kraja (j==0)
            ili dok ne naidjemo na element a[j-1] koji je manji od
72            a[j]. */
            for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
74                 int temp = a[j];
                    a[j] = a[j - 1];
76                 a[j - 1] = temp;
            }
78     }
    }

80

82 /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
    algoritma je sledeca: prolazimo kroz niz redom poredeci
84 susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
    poretku. Ovim se najveći element poput mehurica istiskuje na
86 "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je
    potrebno ovaj postupak ponoviti nad nizom a[0],...,a[n-2],
88 tj. nad prvih n-1 elemenata niza bez poslednjeg koji je
    postavljen na pravu poziciju. Nakon toga se istu postupak
90 ponavlja nad sve kracim i kracim prefiksima niza, cime se
    jedan po jedan istiskuju elementi na svoje prave pozicije. */
92 void bubblesort(int a[], int n)
```

```

{
94     int i, j;
95     int ind;
96
97     for (i = n, ind = 1; i > 1 && ind; i--)
98
99         /* Poput "mehurica" potiskujemo najveći element među
100            elementima od a[0] do a[i-1] na poziciju i-1 upoređujući
101            susedne elemente niza i potiskujući veći u desno */
102         for (j = 0, ind = 0; j < i - 1; j++)
103             if (a[j] > a[j + 1]) {
104                 int temp = a[j];
105                 a[j] = a[j + 1];
106                 a[j + 1] = temp;
107
108                 /* Promenljiva ind registruje da je bilo premestanja.
109                    Samo u tom slučaju ima smisla ici na sledeću
110                    iteraciju, jer ako nije bilo premestanja, znači da su
111                    svi elementi već u dobrom poretku, pa nema potrebe
112                    prelaziti na kraci prefiks niza. Moglo je naravno i
113                    bez ovoga, algoritam bi radio ispravno, ali bi bio
114                    manje efikasan, jer bi često nepotrebno vrsio mnoga
115                    upoređivanja, kada je već jasno da je sortiranje
116                    završeno. */
117                 ind = 1;
118             }
119 }
120
121 /* Selsort je jednostavno proširenje sortiranja umetanjem koje
122    dopušta direktnu razmenu udaljenih elemenata. Proširenje se
123    sastoji u tome da se kroz algoritam umetanja prolazi više
124    puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
125    koji je manji od n (sto omogućuje razmenu udaljenih
126    elemenata) i tako se dobija h-sortiran niz, tj. niz u kome su
127    elementi na rastojanju h sortirani, mada susedni elementi to
128    ne moraju biti. U drugom prolazu kroz isti algoritam sprovodi
129    se isti postupak ali za manji korak h. Sa prolazima se
130    nastavlja sve do koraka h = 1, u kome se dobija potpuno
131    sortirani niz. Izbor početne vrednosti za h, i načina
132    njegovog smanjivanja menja u nekim slučajevima brzinu
133    algoritma, ali bilo koja vrednost će rezultovati ispravnim
134    sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
135    vrednost 1. */
136 void shellsort(int a[], int n)
137 {
138     int h = n / 2, i, j;
139     while (h > 0) {
140         /* Insertion sort sa korakom h */
141         for (i = h; i < n; i++) {
142             int temp = a[i];
143             j = i;
144             while (j >= h && a[j - h] > temp) {
145                 a[j] = a[j - h];
146                 j -= h;
147             }
148             a[j] = temp;

```

### 3 Algoritmi pretrage i sortiranja

---

```
    }
150    h = h / 2;
    }
152 }

154 #define MAX 1000000

156 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem.
    Sortiranje se vrši od elementa na poziciji l do onog na
158 poziciji d. Na pocetku, da bismo dobili niz kompletno
    sortiran, l mora biti 0, a d je jednako poslednjem validnom
160 indeksu u nizu. Funkcija niz podeli na dve polovine, levu i
    desnu, koje zatim rekurzivno sortira. Od ova dva sortirana
162 podniza, dobijamo sortiran niz ucesljavanjem, tj.
    istovremenim prolaskom kroz oba niza i izborom trenutnog
164 manjeg elementa koji se smesta u pomocni niz. Na kraju
    algoritma, sortirani elementi su u pomocnom nizu, koji se
166 kopira u originalni niz. */
void mergesort(int a[], int l, int d)
168 {
    int s;
170    static int b[MAX];          /* Pomocni niz */
    int i, j, k;

172    /* Izlaz iz rekurzije */
174    if (l >= d)
        return;

176    /* Odredjujemo sredisnji indeks */
178    s = (l + d) / 2;

180    /* Rekurzivni pozivi */
    mergesort(a, l, s);
182    mergesort(a, s + 1, d);

184    /* Inicijalizacija indeksa. Indeks i prolazi krozi levu
        polovinu niza, dok indeks j prolazi kroz desnu polovinu
186 niza. Indeks k prolazi kroz pomocni niz b[] */
    i = l;
188    j = s + 1;
    k = 0;

190    /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
192    while (i <= s && j <= d) {
        if (a[i] < a[j])
194            b[k++] = a[i++];
        else
196            b[k++] = a[j++];
    }

198    /* U slucaju da se prethodna petlja zavrсила izlaskom
        promenljive j iz dopustenog opsega u pomocni niz
        prepisujemo ostatak leve polovine niza */
200    while (i <= s)
        b[k++] = a[i++];
202
204
```



```
206  /* U slucaju da se prethodna petlja zavrсила izlaskom
    promenljive i iz dopustenog opsega u pomocni niz
    prepisujemo ostatak desne polovine niza */
208  while (j <= d)
    b[k++] = a[j++];
210
    /* Prepisujemo "ucesljani" niz u originalni niz */
212  for (k = 0, i = 1; i <= d; i++, k++)
    a[i] = b[k];
214 }

216 /* Funkcija menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
218 {
    int tmp = a[i];
220  a[i] = a[j];
    a[j] = tmp;
222 }

224
    /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r.
    Njena ideja sortiranja je izbor jednog elementa niza, koga
    nazivamo pivot, koga cemo dovesti na svoje mesto. Posle ovog
    koraka, svi elementi levo od njega bice manji, a svi desno
    bice veci od njega. Kako smo pivota doveli na svoje mesto, da
    bismo imali kompletno sortiran niz, treba sortirati elemente
    levo (manje) od njega, i elemente desno (vece). Kako je
    dimenzija ova dva podniza manja od dimenzije pocetnog niza
    koji je trebalo sortirati, ovaj deo ce za nas uraditi
    rekurzija. */
234 void quicksort(int a[], int l, int r)
236 {
    int i, pivot_position;
238
    /* Izlaz iz rekurzije -- prazan niz */
240  if (l >= r)
    return;
242

    /* Particionisanje niza. Svi elementi na pozicijama <=
    pivot_position (izuzev same pozicije l) su strogo manji od
    pivota. Kada se pronadje neki element manji od pivota,
    uvecava se pivot_position i na tu poziciju se premesta
    nadjeni element. Na kraju ce pivot_position zaista biti
    pozicija na koju treba smestiti pivot, jer ce svi elementi
    levo od te pozicije biti manji a desno biti veci ili
    jednaki od pivota. */
252  pivot_position = l;
    for (i = l + 1; i <= r; i++)
254  if (a[i] < a[l])
    swap(a, ++pivot_position, i);
256

    /* Postavljamo pivot na svoje mesto */
258  swap(a, l, pivot_position);

260  /* Rekurzivno sortiramo elemente manje od pivota */
```

### 3 Algoritmi pretrage i sortiranja

```
262 quicksort(a, l, pivot_position - 1);
/* Rekurzivno sortiramo elemente vece pivota */
264 quicksort(a, pivot_position + 1, r);
}
```

```
#include <stdio.h>
2 #include <stdlib.h>
#include <time.h>
4 #include "sort.h"

6 /* Maksimalna duzina niza */
#define MAX 1000000

8
int main(int argc, char *argv[])
10 {
    /******
12     tip_sortiranja == 0 => selectionsort
        (podrazumevano)
14     tip_sortiranja == 1 => insertionsort
        -i opcija komandne linije
16     tip_sortiranja == 2 => bubblesort
        -b opcija komandne linije
18     tip_sortiranja == 3 => shellsort
        -s opcija komandne linije
20     tip_sortiranja == 4 => mergesort
        -m opcija komandne linije
22     tip_sortiranja == 5 => quicksort
        -q opcija komandne linije
24     *****/
    int tip_sortiranja = 0;
26     /******
        tip_niza == 0 => slucajno generisani nizovi
        (podrazumevano)
28     tip_niza == 1 => rastuce sortirani nizovi
        -r opcija komandne linije
30     tip_niza == 2 => opadajuce soritrani nizovi
        -o opcija komandne linije
32     *****/
34     int tip_niza = 0;

36     /* Dimenzija niza koji se sortira */
    int dimenzija;
38     int i;
    int niz[MAX];
40

    /* Provera argumenata komandne linije */
42     if (argc < 2) {
        fprintf(stderr,
44             "Program zahteva bar 2 argumenta komandne linije!\n");
        exit(EXIT_FAILURE);
46     }

48     /* Ocitavamo opcije i argumente prilikom poziva programa */
    for (i = 1; i < argc; i++) {
50         /* Ako je u pitanju opcija... */
        if (argv[i][0] == '-') {
```

```

52     switch (argv[i][1]) {
53     case 'i':
54         tip_sortiranja = 1;
55         break;
56     case 'b':
57         tip_sortiranja = 2;
58         break;
59     case 's':
60         tip_sortiranja = 3;
61         break;
62     case 'm':
63         tip_sortiranja = 4;
64         break;
65     case 'q':
66         tip_sortiranja = 5;
67         break;
68     case 'r':
69         tip_niza = 1;
70         break;
71     case 'o':
72         tip_niza = 2;
73         break;
74     default:
75         printf("Pogresna opcija -%c\n", argv[i][1]);
76         return 1;
77         break;
78     }
79 }
80 /* Ako je u pitanju argument, onda je to duzina niza koji
81    treba da se sortira */
82 else {
83     dimenzija = atoi(argv[i]);
84     if (dimenzija <= 0 || dimenzija > MAX) {
85         fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
86         exit(EXIT_FAILURE);
87     }
88 }
89 }
90
91 /* Elemente niza odredjujemo slucajno, ali vodeci racuna o
92    tipu niza dobijenom iz komandni linije. srandom funkcija
93    obezbedjuje novi seed za pozivanje random funkcije, i kako
94    nas niz ne bi uvek isto izgledao seed smo postavili na
95    tekuce vreme u sekundama od Nove godine 1970. random()%100
96    daje brojeve izmedju 0 i 99 */
97 srandom(time(NULL));
98 if (tip_niza == 0)
99     for (i = 0; i < dimenzija; i++)
100         niz[i] = random();
101 else if (tip_niza == 1)
102     for (i = 0; i < dimenzija; i++)
103         niz[i] =
104             i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
105 else
106     for (i = 0; i < dimenzija; i++)
107         niz[i] =

```

### 3 Algoritmi pretrage i sortiranja

```
108         i == 0 ? random() % 100 : niz[i - 1] - random() % 100;

110     /* Ispisujemo elemente niza */
111     /******
112     Ovaj deo je iskomentarisan jer ne zelimo da se sledeci ispis
113     nadje na izlazu. Njegova svrha je samo bila provera da li je
114     niz generisan u skladu sa opcijama komandne linije.

116     printf("Niz koji sortiramo je:\n");
117     for (i = 0; i < dimenzija; i++)
118         printf("%d\n", niz[i]);
119     *****/

120

122     /* Sortiramo niz na odgovarajuci nacin */
123     if (tip_sortiranja == 0)
124         selectionsort(niz, dimenzija);
125     else if (tip_sortiranja == 1)
126         insertionsort(niz, dimenzija);
127     else if (tip_sortiranja == 2)
128         bubblesort(niz, dimenzija);
129     else if (tip_sortiranja == 3)
130         shellsort(niz, dimenzija);
131     else if (tip_sortiranja == 4)
132         mergesort(niz, 0, dimenzija - 1);
133     else
134         quicksort(niz, 0, dimenzija - 1);

136     /* Ispisujemo elemente niza */
137     /******
138     Ovaj deo je iskomentarisan jer nismo zeleli da vreme potrebno
139     za njegovo izvršavanje bude ukljuceno u vreme izmereno
140     programom time. Takodje, kako je svrha ovog programa da prikaze
141     vremena razlicitih algoritama sortiranja, dimenzije nizova ce
142     biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
143     od toliko elemenata. Ovaj deo je koriscen u razvoju programa
144     zarad testiranja korektnosti.

146     printf("Sortiran niz je:\n");
147     for (i = 0; i < dimenzija; i++)
148         printf("%d\n", niz[i]);
149     *****/

150     return 0;
152 }
```

#### Rešenje 3.19

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 #define MAX_BR_TAKAKA 128
7
```

```
/* Struktura koja reprezentuje koordinate tacke */
9 typedef struct Tacka {
    int x;
11    int y;
} Tacka;

13
/* Funkcija racuna rastojanje zadate tacke od koordinatnog
15    pocetka (0,0) */
float rastojanje(Tacka A)
17 {
    return sqrt(A.x * A.x + A.y * A.y);
19 }

21 /* Funkcija koja sortira niz tacaka po rastojanju od
    koordinatnog pocetka */
23 void sortiraj_po_rastojanju(Tacka t[], int n)
{
25     int min, i, j;
    Tacka tmp;

27     for (i = 0; i < n - 1; i++) {
29         min = i;
        for (j = i + 1; j < n; j++) {
31             if (rastojanje(t[j]) < rastojanje(t[min])) {
                min = j;
33             }
        }
35         if (min != i) {
            tmp = t[i];
37             t[i] = t[min];
            t[min] = tmp;
39         }
    }
41 }

43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
void sortiraj_po_x(Tacka t[], int n)
45 {
    int min, i, j;
47    Tacka tmp;

49    for (i = 0; i < n - 1; i++) {
        min = i;
51        for (j = i + 1; j < n; j++) {
            if (abs(t[j].x) < abs(t[min].x)) {
53                min = j;
            }
55        }
        if (min != i) {
57            tmp = t[i];
            t[i] = t[min];
59            t[min] = tmp;
        }
61    }
}
63
```

### 3 Algoritmi pretrage i sortiranja

---

```
/* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
65 void sortiraj_po_y(Tacka t[], int n)
{
67     int min, i, j;
    Tacka tmp;

69     for (i = 0; i < n - 1; i++) {
71         min = i;
        for (j = i + 1; j < n; j++) {
73             if (abs(t[j].y) < abs(t[min].y)) {
                min = j;
75             }
        }
77         if (min != i) {
            tmp = t[i];
79             t[i] = t[min];
            t[min] = tmp;
81         }
    }
83 }

85

87 int main(int argc, char *argv[])
{
89     FILE *ulaz;
    FILE *izlaz;
91     Tacka tacke[MAX_BR_TACAKA];
    int i, n;

93     /* Proveravamo broj argumenata komandne linije: ocekujemo ime
95      izvrsnog programa, opciju, ime ulazne datoteke i ime
      izlazne datoteke tj. ocekujemo 4 argumenta */
97     if (argc != 4) {
        fprintf(stderr,
99             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
        return 0;
101    }

103    /* Otvaramo datoteku u kojoj su zadate tacke */
    ulaz = fopen(argv[2], "r");
105    if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
107                argv[2]);
        return 0;
109    }

111    /* Otvaramo datoteku u koju treba upisati rezultat */
    izlaz = fopen(argv[3], "w");
113    if (izlaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
115                argv[3]);
        return 0;
117    }

119    /* Sve dok ne stignemo do kraja ulazne datoteke ucitavamo
```

```

121     koordinate tacaka i smestamo ih na odgovarajucu poziciju
122     odredjenu brojacem i; prilikom učitavanja oslanjamo se na
123     svojstvo funkcije fscanf povratka EOF vrednosti kada stigne
124     do kraja ulaza */
125     i = 0;
126     while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
127         i++;
128     }
129     /* Cuvamo broj procitanih tacaka */
130     n = i;
131
132     /* Analiziramo zadatu opciju: kako ocekujemo da je argv[1]
133        "-x" ili "-y" ili "-o" sigurni smo da je argv[1][0] crtica
134        (karakter -) i dalje proveravamo sta je na sledecoj
135        poziciji tj. sta je argv[1][1] */
136
137     switch (argv[1][1]) {
138     case 'x':
139         /* Ako je u pitanju karakter x, pozivamo funkciju za
140            sortiranje po vrednosti x koordinate */
141         sortiraj_po_x(tacke, n);
142         break;
143     case 'y':
144         /* Ako je u pitanju karakter y, pozivamo funkciju za
145            sortiranje po vrednosti y koordinate */
146         sortiraj_po_y(tacke, n);
147         break;
148     case 'o':
149         /* Ako je u pitanju karakter o, pozivamo funkciju za
150            sortiranje po udaljenosti od koorinatnog pocetka */
151         sortiraj_po_rastojanju(tacke, n);
152         break;
153     }
154
155     /* Upisujemo dobijeni niz u izlaznu datoteku */
156     for (i = 0; i < n; i++) {
157         fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
158     }
159
160     /* Zatvaramo otvorene datoteke */
161     fclose(ulaz);
162     fclose(izlaz);
163
164     /* Završavamo sa programom */
165     return 0;
166 }

```

### Rešenje 3.20

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 1000

```

### 3 Algoritmi pretrage i sortiranja

---

```
6 #define MAX_DUZINA 16

8 /* Struktura koja reprezentuje jednog gradjanina */
typedef struct gr {
10     char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
12 } Gradjanin;

14 /* Funkcija sortira niz gradjana rastuce po imenima */
void sort_ime(Gradjanin a[], int n)
16 {
    int i, j;
18     int min;
    Gradjanin pom;

20     for (i = 0; i < n - 1; i++) {
22         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
            nalazi najmanji od elemenata a[i].ime,...,a[n-1].ime. */
24         min = i;
        for (j = i + 1; j < n; j++)
26             if (strcmp(a[j].ime, a[min].ime) < 0)
                min = j;
28         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
            samo ako su (i) i min razliciti, inace je nepotrebno. */
30         if (min != i) {
            pom = a[i];
32             a[i] = a[min];
            a[min] = pom;
34         }
    }
36 }

38 /* Funkcija sortira niz gradjana rastuce po prezimenima */
void sort_prezime(Gradjanin a[], int n)
40 {
    int i, j;
42     int min;
    Gradjanin pom;

44     for (i = 0; i < n - 1; i++) {
46         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
            nalazi najmanji od elemenata
48             a[i].prezime,...,a[n-1].prezime. */
            min = i;
            for (j = i + 1; j < n; j++)
50                 if (strcmp(a[j].prezime, a[min].prezime) < 0)
                    min = j;
52             /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
                samo ako su (i) i min razliciti, inace je nepotrebno. */
54             if (min != i) {
                pom = a[i];
56                 a[i] = a[min];
                a[min] = pom;
58             }
    }
60 }
```



```

62  /* Pretraga niza Gradjana */
64  int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
65  {
66      int i;
67      for (i = 0; i < n; i++)
68          if (strcmp(a[i].ime, x->ime) == 0
69              && strcmp(a[i].prezime, x->prezime) == 0)
70              return i;
71      return -1;
72  }

74
76  int main()
77  {
78      Gradjanin spisak1[MAX], spisak2[MAX];
79      int isti_rbr = 0;
80      int i, n;
81      FILE *fp = NULL;

82      /* Otvaranje datoteke */
83      if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
84          fprintf(stderr,
85              "Neuspesno otvaranje datoteke biracki-spisak.txt.\n");
86          exit(EXIT_FAILURE);
87      }

88      /* Citanje sadrzaja */
89      for (i = 0;
90          fscanf(fp, "%s %s", spisak1[i].ime,
91              spisak1[i].prezime) != EOF; i++)
92          spisak2[i] = spisak1[i];
93      n = i;

94      /* Zatvaranje datoteke */
95      fclose(fp);

96      sort_ime(spisak1, n);

97      /******
98      Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
99      sortiranih nizova. Koriscen je samo u fazi testiranja programa.
100
101      printf("Biracki spisak [uredjen prema imenima]:\n");
102      for(i=0; i<n; i++)
103          printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
104      *****/

105      sort_prezime(spisak2, n);

106      /******
107      Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
108      sortiranih nizova. Koriscen je samo u fazi testiranja programa.
109
110      printf("Biracki spisak [uredjen prema prezimenima]:\n");
111      for(i=0; i<n; i++)

```

### 3 Algoritmi pretrage i sortiranja

```
118     printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
120     *****/
122     /* Linearno pretrazivanje nizova */
122     for (i = 0; i < n; i++)
124         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
124             isti_rbr++;
126     /* Alternativno (efikasnije) resenje */
126     /******
128     for(i=0; i<n ;i++)
130         if ( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
130             strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
132             isti_rbr++;
132     *****/
134     /* Ispis rezultata */
134     printf("%d\n", isti_rbr);
136
136     exit(EXIT_SUCCESS);
138 }
```

#### Rešenje 3.22

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  #define MAX_BR_RECII 128
6  #define MAX_DUZINA_RECII 32
7
8  /* Funkcija koja izracunava broj suglasnika u reci */
9  int broj_suglasnika(char s[])
10 {
11     char c;
12     int i;
13     int suglasnici = 0;
14     /* Obilazimo karakter po karakter zadate niske */
15     for (i = 0; s[i]; i++) {
16         /* Ako je u pitanju slovo */
17         if (isalpha(s[i])) {
18             /* Pretvaramo ga u veliko da bismo mogli da pokrijemo
19             slucaj i malih i velikih suglasnika */
20             c = toupper(s[i]);
21             /* Ukoliko slovo nije samoglasnik */
22             if (c != 'A' && c != 'E' && c != 'I' && c != 'O'
23                 && c != 'U') {
24                 /* Uvecavamo broj suglasnika */
25                 suglasnici++;
26             }
27         }
28     }
29     /* Vracamo izracunatu vrednost */
30     return suglasnici;
31 }
```

```

32 }

34 /* Funkcija koja sortira reci po zadatom kriterijumu.
   Informacija o duzini reci se mora proslediti zbog pravilnog
36 upravljanja memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_REC], int n)
38 {
    int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
40    duzina_j, duzina_min;
    char tmp[MAX_DUZINA_REC];
42    for (i = 0; i < n - 1; i++) {
        min = i;
44        for (j = i; j < n; j++) {
            /* Prvo uporedjujemo broj suglasnika */
46            broj_suglasnika_j = broj_suglasnika(reci[j]);
            broj_suglasnika_min = broj_suglasnika(reci[min]);
48            if (broj_suglasnika_j < broj_suglasnika_min)
                min = j;
50            else if (broj_suglasnika_j == broj_suglasnika_min) {
                /* Zatim, reci imaju isti broj suglasnika uporedjujemo
52                duzine */
                duzina_j = strlen(reci[j]);
54                duzina_min = strlen(reci[min]);

56                if (duzina_j < duzina_min)
                    min = j;
58                else
                    /* A ako reci imaju i isti broj suglasnika i iste
60                    duzine, uporedjujemo ih leksikografski */
                    if (duzina_j == duzina_min
62                        && strcmp(reci[j], reci[min]) < 0)
                        min = j;
64            }
        }
66        if (min != i) {
            strcpy(tmp, reci[min]);
68            strcpy(reci[min], reci[i]);
            strcpy(reci[i], tmp);
70        }
    }
72 }

74 int main()
{
76     FILE *ulaz;
78     int i = 0, n;

80     /* Niz u kojem ce biti smestane reci. Prvi broj oznacava broj
       reci, a drugi maksimalnu duzinu pojedinacne reci */
82     char reci[MAX_BR_REC][MAX_DUZINA_REC];

84     /* Otvaramo datoteku niske.txt za citanje */
    ulaz = fopen("niske.txt", "r");
86     if (ulaz == NULL) {
        fprintf(stderr,

```

### 3 Algoritmi pretrage i sortiranja

```
88         "Greska prilikom otvaranja datoteke niske.txt!\n");
89     return 0;
90 }
91
92 /* Sve dok mozemo da procitamo sledecu rec */
93 while (fscanf(ulaz, "%s", reci[i]) != EOF) {
94     /* Proveravamo da li smo ucitali najvise dozvoljenih reci i
95        ako jesmo, prekidamo ucitavanje */
96     if (i == MAX_BR_RECII)
97         break;
98     /* Pripremamo brojac za narednu iteraciju */
99     i++;
100 }
101
102 /* n je duzina naseg niza reci i predstavlja poslednju
103    vrednost koriscenog brojaca */
104 n = i;
105 /* Pozivamo funkciju za sortiranje reci - OPREZ: nacin
106    prosledjivanja niza reci */
107 sortiraj_reci(reci, n);
108
109 /* Ispisujemo sortirani niz reci */
110 for (i = 0; i < n; i++) {
111     printf("%s ", reci[i]);
112 }
113 printf("\n");
114
115 /* Zatvaramo datoteku */
116 fclose(ulaz);
117
118 /* Završavamo sa programom */
119 return 0;
120 }
```

#### Rešenje 3.23

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_ARTIKALA 100000
6
7 /* Struktura koja predstavlja jedan artikal */
8 typedef struct art {
9     long kod;
10     char naziv[20];
11     char proizvodjac[20];
12     float cena;
13 } Artikal;
14
15 /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj
16    sa traženim bar kodom */
17 int binarna_pretraga(Artikal a[], int n, long x)
18 {
19     int levi = 0;
```

```

21     int desni = n - 1;
22
23     /* Dokle god je indeks levi levo od indeksa desni */
24     while (levi <= desni) {
25         /* Racunamo sredisnji indeks */
26         int srednji = (levi + desni) / 2;
27         /* Ako je sredisnji element veci od x, tada se x mora
28            nalaziti u levoj polovini niza */
29         if (x < a[srednji].kod)
30             desni = srednji - 1;
31         /* Ako je sredisnji element manji od x, tada se x mora
32            nalaziti u desnoj polovini niza */
33         else if (x > a[srednji].kod)
34             levi = srednji + 1;
35         else
36             /* Ako je sredisnji element jednak x, tada smo pronasli x
37                na poziciji srednji */
38             return srednji;
39     }
40     /* Ako nije pronadjen vratamo -1 */
41     return -1;
42 }
43
44 /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
45 void selection_sort(Artikal a[], int n)
46 {
47     int i, j;
48     int min;
49     Artikal pom;
50
51     for (i = 0; i < n - 1; i++) {
52         min = i;
53         for (j = i + 1; j < n; j++)
54             if (a[j].kod < a[min].kod)
55                 min = j;
56         if (min != i) {
57             pom = a[i];
58             a[i] = a[min];
59             a[min] = pom;
60         }
61     }
62 }
63
64 int main()
65 {
66     Artikal asortiman[MAX_ARTIKALA];
67     long kod;
68     int i, n;
69     float racun;
70
71     FILE *fp = NULL;
72
73     /* Otvaranje datoteke */
74     if ((fp = fopen("artikli.txt", "r")) == NULL) {
75         fprintf(stderr,
76             "Neuspesno otvaranje datoteke artikli.txt.\n");

```

```
    exit(EXIT_FAILURE);
77 }

79 /* Ucitavanje artikala */
    i = 0;
81 while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
    asortiman[i].naziv, asortiman[i].proizvodjac,
83     &asortiman[i].cena) == 4)
    i++;
85

87 /* Zatvaranje datoteke */
    fclose(fp);

89 n = i;

91 /* Sortiracemo celokupan asortiman prodavnice prema kodovima
    jer ce pri kucanju racuna prodavac unositi kod artikla.
93 Prilikom kucanja svakog racuna pretrazuje se asortiman, da
    bi se utvrdila cena artikla. Kucanje racuna obuhvata vise
95 pretraga asortimana i u interesu nam je da ta operacija
    bude sto efikasnija. Zelimo da koristimo algoritam binarne
97 pretrage priliko pretrazivanje po kodu artikla. Iz tog
    razloga, potrebno je da nam asortiman bude sortirani po
99 kodovima i to cemo uraditi primenom selection sort
    algoritma. Sortiramo samo jednom na pocetku, ali zato posle
101 brzo mozemo da pretrazujemo prilikom kucanja proizvoljno
    puno racuna. Vreme koje se utrosi na sortiranje na pocetku
103 izvorsavanja programa, kasnije se isplati jer za brojna
    trazenja artikla mozemo umesto linearne da koristimo
105 efikasniju binarnu pretragu. */
    selection_sort(asortiman, n);
107

109 /* Ispis stanja u prodavnici */
    printf
        ("Asortiman:\nKOD          Naziv artikla      Ime
        proizvodjaca      Cena\n");
111 for (i = 0; i < n; i++)
    printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113     asortiman[i].naziv, asortiman[i].proizvodjac,
    asortiman[i].cena);
115

    kod = 0;
117 while (1) {
    printf("-----\n");
119    printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
    printf("- Za nov racun unesite kod artikla!\n\n");
121    /* Unos bar koda provog artikla sledeceg kupca */
    if (scanf("%ld", &kod) == EOF)
123        break;
    /* Trenutno racun novog kupca */
125    racun = 0;
    /* Za sve artikle trenutnog kupca */
127    while (1) {
        /* Nalazimo ih u nizu */
129        if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
            printf
```

```

131         ("\tGRESKA: Ne postoji proizvod sa traženim kodom!\n");
    } else {
133         printf("\tTrazili ste:\t%s %s %12.2f\n",
                asortiman[i].naziv, asortiman[i].proizvodjac,
135                asortiman[i].cena);
        /* I dodajemo na ukupan racun */
137        racun += asortiman[i].cena;
    }
139    /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0
        ako on nema vise artikla */
141    printf("Unesite kod artikla [ili 0 za prekid]: \t");
    scanf("%ld", &kod);
143    if (kod == 0)
        break;
145    }
    /* Stampanje ukupnog racuna trenutnog kupca */
147    printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
    }
149
151    printf("Kraj rada kase!\n");
153    exit(EXIT_SUCCESS);
}

```

### Rešenje 3.24

```

#include <stdio.h>
2  #include <stdlib.h>
#include <string.h>
4
#define MAX 500
6
/* Struktura koja nam je neophodna za sve informacije o
8  pojedinacnom studentu */
typedef struct {
10     char ime[20];
    char prezime[25];
12     int prisustvo;
    int zadaci;
14 } Student;

16 /* Funkcija kojom sortiramo niz struktura po prezimenu
    leksikografski rastuce */
18 void sort_ime_leksikografski(Student niz[], int n)
{
20     int i, j;
    int min;
22     Student pom;

24     for (i = 0; i < n - 1; i++) {
        min = i;
26         for (j = i + 1; j < n; j++)
            if (strcmp(niz[j].ime, niz[min].ime) < 0)
28             min = j;
    }
}

```

### 3 Algoritmi pretrage i sortiranja

```
30     if (min != i) {
31         pom = niz[min];
32         niz[min] = niz[i];
33         niz[i] = pom;
34     }
35 }
36 }

38 /* Funkcija kojom sortiramo niz struktura po ukupnom broju
39    uradjenih zadataka opadajuće, ukoliko neki studenti imaju
40    isti broj uradjenih zadataka sortiraju se po dužini imena
41    rastuće. */
42 void sort_zadatke_pa_imena(Student niz[], int n)
43 {
44     int i, j;
45     int max;
46     Student pom;
47     for (i = 0; i < n - 1; i++) {
48         max = i;
49         for (j = i + 1; j < n; j++)
50             if (niz[j].zadaci > niz[max].zadaci)
51                 max = j;
52             else if (niz[j].zadaci == niz[max].zadaci
53                     && strlen(niz[j].ime) < strlen(niz[max].ime))
54                 max = j;
55         if (max != i) {
56             pom = niz[max];
57             niz[max] = niz[i];
58             niz[i] = pom;
59         }
60     }
61 }

62 /* Funkcija kojom sortiramo niz struktura po broju casova na
63    kojima su bili opadajuće, a ukoliko * neki studenti imaju
64    isti broj casova, sortiraju se opadajuće po broju uradjenih
65    zadataka, * a ukoliko se i po broju zadataka poklapaju
66    sortirati ih po prezimenu opadajuće. */
67 void sort_prisustvo_pa_zadatke_pa_prezimena(Student niz[], int n)
68 {
69     int i, j;
70     int max;
71     Student pom;
72     for (i = 0; i < n - 1; i++) {
73         max = i;
74         for (j = i + 1; j < n; j++)
75             if (niz[j].prisustvo > niz[max].prisustvo)
76                 max = j;
77             else if (niz[j].prisustvo == niz[max].prisustvo
78                     && niz[j].zadaci > niz[max].zadaci)
79                 max = j;
80             else if (niz[j].prisustvo == niz[max].prisustvo
81                     && niz[j].zadaci == niz[max].zadaci
82                     && strcmp(niz[j].prezime, niz[max].prezime) > 0)
83                 max = j;
84         if (max != i) {
```



```

86     pom = niz[max];
      niz[max] = niz[i];
88     niz[i] = pom;
    }
90 }
}

92

94
96 int main(int argc, char *argv[])
97 {
    Student praktikum[MAX];
98     int i, br_studenata = 0;

100     FILE *fp = NULL;

102     /* Otvaranje datoteke za citanje */
    if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
104         fprintf(stderr,
            "Neuspesno otvaranje datoteke aktivnost.txt.\n");
106         exit(EXIT_FAILURE);
    }

108     /* Ucitavanje sadrzaja */
    for (i = 0;
        fscanf(fp, "%s%s%d%d", praktikum[i].ime,
112            praktikum[i].prezime, &praktikum[i].prisustvo,
            &praktikum[i].zadaci) != EOF; i++);

114     /* Zatvaranje datoteke */
    fclose(fp);
116     br_studenata = i;

118     /* Kreiramo prvi spisak studenata na kom su sortirani
        leksikografski po imenu rastuce */
120     sort_ime_leksikografski(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
122     if ((fp = fopen("dat1.txt", "w")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
124         exit(EXIT_FAILURE);
    }

126     /* Upis niza u datoteku */
    fprintf
128     (fp,
        "Studenti sortirani po imenu leksikografski rastuce:\n");
130     for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
132            praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);

134     /* Zatvaranje datoteke */
    fclose(fp);

136

138     /* Na drugom su sortirani po ukupnom broju uradjenih zadataka
        opadajuce, ukoliko neki studenti imaju isti broj uradjenih
        zadataka sortiraju se po duzini imena rastuce. */
140     sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */

```

### 3 Algoritmi pretrage i sortiranja

```
142 if ((fp = fopen("dat2.txt", "w")) == NULL) {
143     fprintf(stderr, "Neuspješno otvaranje datoteke dat2.txt.\n");
144     exit(EXIT_FAILURE);
145 }
146 /* Upis niza u datoteku */
147 fprintf(fp,
148         "Studenti sortirani po broju zadataka opadajuće,\n");
149 fprintf(fp, "pa po dužini imena raste:\n");
150 for (i = 0; i < br_studenata; i++)
151     fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
152             praktikum[i].prezime, praktikum[i].prisustvo,
153             praktikum[i].zadaci);
154 /* Zatvaranje datoteke */
155 fclose(fp);
156
157 /* Na trećem spisku su sortirani po broju časova na kojima su
158    bili opadajuće, a ukoliko neki studenti imaju isti broj
159    časova, sortiraju se opadajuće po broju urađenih zadataka,
160    a ukoliko se i po broju zadataka poklapaju sortirati ih po
161    prezimenu opadajuće. */
162 sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum,
163                                           br_studenata);
164 /* Otvaranje datoteke za pisanje */
165 if ((fp = fopen("dat3.txt", "w")) == NULL) {
166     fprintf(stderr, "Neuspješno otvaranje datoteke dat3.txt.\n");
167     exit(EXIT_FAILURE);
168 }
169 /* Upis niza u datoteku */
170 fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
171 fprintf(fp, "pa po broju zadataka,\n");
172 fprintf(fp, "pa po prezimenima leksikografski opadajuće,\n");
173 for (i = 0; i < br_studenata; i++)
174     fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
175             praktikum[i].prezime, praktikum[i].prisustvo,
176             praktikum[i].zadaci);
177 /* Zatvaranje datoteke */
178 fclose(fp);
179
180 return 0;
181 }
```

#### Rešenje 3.25

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define KORAK 10
5
6 /* Struktura koja opisuje jednu pesmu */
7 typedef struct {
8     char *izvodjac;
9     char *naslov;
10    int broj_gledanja;
11 } Pesma;
12
```

```

14  /* Funkcija za uporedjivanje pesama po broju gledanosti
    (potrebna za rad qsort funkcije) */
16  int uporedi_gledanost(const void *pp1, const void *pp2)
18  {
    Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

20     return p2->broj_gledanja - p1->broj_gledanja;
22 }

24  /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad
    qsort funkcije) */
26  int uporedi_naslove(const void *pp1, const void *pp2)
28  {
    Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

30     return strcmp(p1->naslov, p2->naslov);
32 }

34  /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za
    rad qsort funkcije) */
36  int uporedi_izvodjace(const void *pp1, const void *pp2)
38  {
    Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

40     return strcmp(p1->izvodjac, p2->izvodjac);
42 }

44  int main(int argc, char *argv[])
46  {
    FILE *ulaz;
    Pesma *pesme;                /* Pokazivac na deo memorije za
48                                   cuvanje pesama */
    int alocirano_za_pesme;      /* Broj mesta alociranih za
50                                   pesme */
    int i;                       /* Redni broj pesme cije se
52                                   informacije citaju */
    int n;                       /* Ukupan broj pesama */
54     int j, k;
    char c;
56     int alocirano;              /* Broj mesta alociranih za
                                   propratne informacije o
58                                   pesmama */

    int broj_gledanja;

60     /* Pripremamo datoteku za citanje */
62     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
    if (ulaz == NULL) {
64         printf("Greska pri otvaranju ulazne datoteke!\n");
        return 0;
66     }

68     /* Citamo informacije o pesmama */

```

```
pesme = NULL;
70 alocirano_za_pesme = 0;
i = 0;
72
74 while (1) {
    /* Proveravamo da li smo stigli do kraja datoteke */
76 c = fgetc(ulaz);
    if (c == EOF) {
78 /* Ako smo dobili kao rezultat EOF, jesmo, nema vise
        sadrzaja za citanje */
80 break;
    } else {
82 /* Ako nismo, vracamo procitani karakter nazad */
        ungetc(c, ulaz);
84 }

86
    /* Proveravamo da li imamo dovoljno memorije za citanje nove
88 pesme */
    if (alocirano_za_pesme == i) {
90
92 /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
        alociramo novih KORAK mesta */
        alocirano_za_pesme += KORAK;
94 pesme =
            (Pesma *) realloc(pesme,
96                             alocirano_za_pesme * sizeof(Pesma));

98 /* Proveravamo da li je nova memorija uspesno realocirana */
    if (pesme == NULL) {
100 /* Ako nije ... */
        /* Ispisujemo obavestenje */
102 printf("Problem sa alokacijom memorije!\n");

104 /* I oslobadjamo svu memoriju zauzetu do ovog koraka */
        for (k = 0; k < i; k++) {
106 free(pesme[k].izvodjac);
            free(pesme[k].naslov);
108 }
        free(pesme);
110 return 0;
    }
112 }

114
    /* Ako jeste, nastavljamo sa citanjem pesama ... */
116 /* Citamo ime izvodjaca */

118 j = 0; /* Oznacava poziciju na koju
            treba smestiti procitani
            karakter */
120
122 alocirano = 0; /* Oznacava broj alociranih
                    mesta */
124 pesme[i].izvodjac = NULL; /* Memorija koju mozemo
                                koristiti za smestanje
```

```

126                                     procitanih karaktera */
127
128 /* Sve dok ne stignemo do prve beline u liniji - beline koja
129    se nalazi nakon imena izvodjaca - citamo karaktere iz
130    datoteke */
131 while ((c = fgetc(ulaz)) != ' ') {
132
133     /* Proveravamo da li imamo dovoljno memorije za smestanje
134        procitanog karaktera */
135     if (j == alocirano) {
136
137         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
138            alociramo novih KORAK mesta */
139         alocirano += KORAK;
140         pesme[i].izvodjac =
141             (char *) realloc(pesme[i].izvodjac,
142                             alocirano * sizeof(char));
143
144         /* Proveravamo da li je nova alokacija uspesna */
145         if (pesme[i].izvodjac == NULL) {
146             /* Ako nije... */
147             /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
148             for (k = 0; k < i; k++) {
149                 free(pesme[k].izvodjac);
150                 free(pesme[k].naslov);
151             }
152             free(pesme);
153             /* I prekidamo sa izvršavanjem programa */
154             return 0;
155         }
156     }
157     /* Ako imamo dovoljno memorije, smestamo procitani
158        karakter */
159     pesme[i].izvodjac[j] = c;
160     j++;
161     /* I nastavljamo sa citanjem */
162 }
163
164 /* Upisujemo terminirajucu nulu na kraju reci */
165 pesme[i].izvodjac[j] = '\0';
166
167
168 /* Citamo - */
169 fgetc(ulaz);
170
171 /* Citamo razmak */
172 fgetc(ulaz);
173
174
175 /* Citamo naslov pesme */
176 j = 0;
177                                     /* Oznacava poziciju na koju
178                                     treba smestiti procitani
179                                     karakter */
180 alocirano = 0;
181                                     /* Oznacava broj alociranih
182                                     mesta */

```

```
182     pesme[i].naslov = NULL;          /* Memorija koju mozemo
                                         koristiti za smestanje
                                         procitanih karaktera */
184
186     /* Sve dok ne stignemo do zareza - zareza koji se nalazi
         nakon naslova pesme - citamo karaktere iz datoteke */
188
188     while ((c = fgetc(ulaz)) != ',') {
190         /* Proveravamo da li imamo dovoljno memorije za smestanje
             procitanog karaktera */
190         if (j == alocirano) {
192             /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
                 alociramo novih KORAK mesta */
194             alocirano += KORAK;
194             pesme[i].naslov =
196                 (char *) realloc(pesme[i].naslov,
                                     alocirano * sizeof(char));
198
198             /* Proveravamo da li je nova alokacija uspesna */
200             if (pesme[i].naslov == NULL) {
202                 /* Ako nije... */
202                 /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
204                 for (k = 0; k < i; k++) {
204                     free(pesme[k].izvodjac);
204                     free(pesme[k].naslov);
206                 }
206                 free(pesme[i].izvodjac);
208                 free(pesme);
210
210                 /* I prekidamo izvršavanje programa */
212                 return 0;
212             }
214
214             /* Ako imamo dovoljno memorije, smestamo procitani
                 karakter */
216             pesme[i].naslov[j] = c;
218             j++;
218             /* I nastavljamo dalje sa citanjem */
220         }
220         /* Upisujemo terminirajucu nulu na kraju reci */
222         pesme[i].naslov[j] = '\0';
224
224         /* Citamo razmak */
226         fgetc(ulaz);
228
228         /* Citamo broj gledanja */
230
230         broj_gledanja = 0;
232
232         /* Sve dok ne stignemo do znaka za novi red - kraja linije -
             citamo karaktere iz datoteke */
234         while ((c = fgetc(ulaz)) != '\n') {
234             broj_gledanja = broj_gledanja * 10 + (c - '0');
236         }
```

```

238     pesme[i].broj_gledanja = broj_gledanja;

240     /* Prelazimo na citanje sledece pesme */
    i++;

242 }

244 /* Cuvamo informaciju o broju pesama koje smo procitali */
    n = i;

246 /* Zatvaramo datoteku jer nam nece vise trebati */
    fclose(ulaz);

250 /* Analiziramo argumente komandne linije */
    if (argc == 1) {

252         /* Nema dodatnih opcija - sortiramo po broju gledanja */
        qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);

254     } else {

256         if (argc == 2 && strcmp(argv[1], "-n") == 0) {
            /* Sortiramo po naslovu */
            qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);

258         } else {
            if (argc == 2 && strcmp(argv[1], "-i") == 0) {
                /* Sortiramo po izvodjacu */
                qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);

260            } else {
                printf("Nedozvoljeni argumenti!\n");
                free(pesme);
                return 0;

262            }
        }

264     }

266 }

268 }

270 }

272 /* Ispisujemo rezultat */
    for (i = 0; i < n; i++) {
        printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
274             pesme[i].broj_gledanja);
    }

276 }

278 /* Oslobadjamo memoriju */
    for (i = 0; i < n; i++) {
        free(pesme[i].izvodjac);
        free(pesme[i].naslov);

280    }
    free(pesme);

282 }

284 /* Prekidamo izvršavanje programa */
    return 0;

286 }

```

### Rešenje 3.28

```

1 #include <stdio.h>

```

### 3 Algoritmi pretrage i sortiranja

---

```
1  #include <stdlib.h>
2  #include <math.h>
3  #include <search.h>
4
5  #define MAX 100
6
7  /* Funkcija poredi dva cela broja */
8
9  int compare_int(const void *a, const void *b)
10 {
11     /* Konvertujemo void pokazivace u int pokazivace koje zatim
12        dereferenciramo, dobijamo int-ove koje oduzimamo i razliku
13        vracamo. */
14
15     /* Zbog moguceg prekoracenja opsega celih brojeva necemo ih
16        oduzimati return *((int *)a) - *((int *)b); */
17
18     int b1 = *((int *) a);
19     int b2 = *((int *) b);
20
21     if (b1 > b2)
22         return 1;
23     else if (b1 < b2)
24         /* Ovo uredjenje favorizujemo jer zelimo rastuci poredak */
25         return -1;
26     else
27         return 0;
28 }
29
30 int compare_int_desc(const void *a, const void *b)
31 {
32     /* Za obrnuti poredak mozemo samo oduzimati a od b */
33     /* return *((int *)b) - *((int *)a); */
34
35     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
36        funkcija */
37     return -compare_int(a, b);
38 }
39
40 /* Test program */
41 int main()
42 {
43     size_t n;
44     int i, x;
45     int a[MAX], *p = NULL;
46
47     /* Unosimo dimenziju */
48     printf("Uneti dimenziju niza: ");
49     scanf("%ld", &n);
50     if (n > MAX)
51         n = MAX;
52
53     /* Unosimo elemente niza */
54     printf("Uneti elemente niza:\n");
55     for (i = 0; i < n; i++)
56         scanf("%d", &a[i]);
57 }
```



```

59  /* Sortiramo niz celih brojeva */
    qsort(a, n, sizeof(int), &compare_int);

61  /* Prikazujemo sortirani niz */
    printf("Sortirani niz u rastucem poretku:\n");
63  for (i = 0; i < n; i++)
        printf("%d ", a[i]);
65  putchar('\n');

67  /* Pretrazivanje niza */
    /* Vrednost koju cemo traziti u nizu */
69  printf("Uneti element koji se trazi u nizu: ");
    scanf("%d", &x);

71  /* Binarna pretraga */
73  printf("Binarna pretraga: \n");
    p = bsearch(&x, a, n, sizeof(int), &compare_int);
75  if (p == NULL)
        printf("Elementa nema u nizu!\n");
77  else
        printf("Element je nadjen na poziciji %ld\n", p - a);
79

    /* Linearna pretraga */
81  printf("Linearna pretraga (lfind): \n");
    p = lfind(&x, a, &n, sizeof(int), &compare_int);
83  if (p == NULL)
        printf("Elementa nema u nizu!\n");
85  else
        printf("Element je nadjen na poziciji %ld\n", p - a);
87

    return 0;
89 }

```

### Rešenje 3.29

```

#include <stdio.h>
2  #include <stdlib.h>
#include <math.h>
4  #include <search.h>

6  #define MAX 100

8  /* Funkcija racuna broj delilaca broja x */
int no_of_deviders(int x)
10 {
    int i;
12    int br;

14    /* Ako je argument negativan broj menjamo mu znak */
    if (x < 0)
16        x = -x;
    if (x == 0)
18        return 0;
    if (x == 1)
20        return 1;

```

### 3 Algoritmi pretrage i sortiranja

```
22  /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
    br = 2;
    /* Sve dok je */
24  for (i = 2; i < sqrt(x); i++)
        if (x % i == 0)
26      /* Ako i deli x onda su delioci: i, x/i */
            br += 2;
28  /* Ako je broj bas kvadrat, onda smo iz petlje izasli kada je
    i bilo bas jednako korenu od x, tada x ima jos jednog
30  delioca */
    if (i * i == x)
32      br++;
34  return br;
}
36
/* Funkcija poredjenja dva cela broja po broju delilaca */
38 int compare_no_deviders(const void *a, const void *b)
{
40     int ak = *(int *) a;
    int bk = *(int *) b;
42     int n_d_a = no_of_deviders(ak);
    int n_d_b = no_of_deviders(bk);
44
    if (n_d_a > n_d_b)
46         return 1;
    else if (n_d_a < n_d_b)
48         return -1;
    else
50         return 0;
}
52
/* Test program */
54 int main()
{
56     size_t n;
    int i;
58     int a[MAX];

60     /* Unosimo dimenziju */
    printf("Uneti dimenziju niza: ");
62     scanf("%ld", &n);
    if (n > MAX)
64         n = MAX;

66     /* Unosimo elemente niza */
    printf("Uneti elemente niza:\n");
68     for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
70
    /* Sortiramo niz celih brojeva prema broju delilaca */
72     qsort(a, n, sizeof(int), &compare_no_deviders);

74     /* Prikazujemo sortirani niz */
    printf
76     ("Sortirani niz u rastucem poretku prema broju delilaca:\n");
```

```

    for (i = 0; i < n; i++)
78     printf("%d ", a[i]);
    putchar('\n');
80
    return 0;
82 }

```

### Rešenje 3.30

```

#include <stdio.h>
2  #include <stdlib.h>
#include <string.h>
4  #include <search.h>
#define MAX_NISKI 1000
6  #define MAX_DUZINA 30

8  /******
   Niz nizova karaktera ovog potpisa
10 char niske[3][4];
   se moze graficki predstaviti ovako:
12 -----
   | a | b | c | \0 | | d | e | \0 |   | f | g | h | \0 |
14 -----

   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu.
16 Za svaku je rezervisano po 4 karaktera ukljucujuci \0.
   Druga rec sa nalazi na adresi koja je za 4 veka od prve reci,
18 a za 4 manja od adrese na kojoj se nalazi treca rec.
   Adresa i-te reci je niske[i] i ona je tipa char*.

20
   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
22 elemenata koji trebaju biti uporedjeni, (npr. pri porecenju
   prve i poslednje reci, pokazivac a ce pokazivati na slovo 'a',
24 a pokazivac b na slovo 'f') kastujemo ih na char*, i pozivamo
   funkciju strcmp nad njima.

26 *****/
int poredi_leksikografski(const void *a, const void *b)
28 {
    return strcmp((char *) a, (char *) b);
30 }

32 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
int poredi_duzine(const void *a, const void *b)
34 {
    return strlen((char *) a) - strlen((char *) b);
36 }

38
int main()
40 {
    int i;
42     size_t n;
    FILE *fp = NULL;
44     char niske[MAX_NISKI][MAX_DUZINA];
    char *p = NULL;
46     char x[MAX_DUZINA];

```

```
48  /* Otvaranje datoteke */
    if ((fp = fopen("niske.txt", "r")) == NULL) {
50      fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
      exit(EXIT_FAILURE);
52  }

54  /* Citanje sadrzaja datoteke */
    for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
56

    /* Zatvaranje datoteke */
58    fclose(fp);
    n = i;
60

    /* Sortiramo niske leksikografski, tako sto biblioteckoj
62     funkciji qsort prosledjujemo funkciju kojom se zadaje
        kriterijum poredjenja 2 niske po duzini */
64    qsort(niske, n, MAX_DUZINA * sizeof(char),
        &poredi_leksikografski);
66

    printf("Leksikografski sortirane niske:\n");
68    for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
70    printf("\n");

72    /* Unosimo trazeni nisku */
    printf("Uneti trazenu nisku: ");
74    scanf("%s", x);

76    /* Binarna pretraga */
    /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
78     jer nam je niz sortiran leksikografski. */
    p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
80               &poredi_leksikografski);

82    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
84              p, (p - (char *) niske) / MAX_DUZINA);
    else
86        printf("Niska nije pronadjena u nizu\n");

88    /* Linearna pretraga */
    /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
90     jer nam je niz sortiran leksikografski. */
    p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
92             &poredi_leksikografski);

94    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
96              p, (p - (char *) niske) / MAX_DUZINA);
    else
98        printf("Niska nije pronadjena u nizu\n");

100    /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
        funkciju poredjenja */
102    qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
```

```

104 printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
106     printf("%s ", niske[i]);
    printf("\n");
108
    exit(EXIT_SUCCESS);
110 }

```

### Rešenje 3.31

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <search.h>
#define MAX_NISKI 1000
#define MAX_DUZINA 30

/*****
Niz pokazivaca na karaktere ovog potpisa
char *niske[3];
posle alokacije u main-u se moze graficki predstaviti ovako:
12  ----->      | a | b | c | \0|
14  =====
    | Y | ----->      | d | e | \0|
16  =====
    | Z | ----->      | f | g | h | \0|
18  -----
Sa leve strane je vertikalno prikazan niz pokazivaca, gde
je i-ti njegov element pokazivac koji pokazuje na alocirane
karaktere i-te reci. Njegov tip je char*.

Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
elemenata koji trebaju biti uporedjeni (recimo adresu od X
i adresu od Z), i kako su X i Z tipa char*, onda a i b su
tipa char**, pa ih tako moramo i kastovati. Da bi uporedili
leksikografski elemente X i Z, moramo uporediti stringove
na koje oni pokazuju, pa zato u sledecoj funkciji pozivamo
strcmp() nad onim na sta pokazuju a i b, kastovani na
odgovarajuci tip.
*****/
32 int poredi_leksikografski(const void *a, const void *b)
    {
34     return strcmp(*(char **) a, *(char **) b);
    }

/* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
leksikografski, vec po duzini */
38 int poredi_duzine(const void *a, const void *b)
40 {
    return strlen(*(char **) a) - strlen(*(char **) b);
42 }

44 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje

```

```
na element u nizu sa kojim se poredi, pa njega treba
46  kastovati na char** i dereferencirati, (videti obrazlozenje
48  za prvu funkciju u ovom zadatku, a pokazivac a pokazuje na
element koji se trazi. U main funkciji je to x, koji je tipa
char*, tako da pokazivac a ovde samo kastujemo i ne
dereferenciramo. */
50  int poredi_leksikografski_b(const void *a, const void *b)
52  {
    return strcmp((char *) a, *(char **) b);
54  }

56  int main()
58  {
    int i;
    size_t n;
    FILE *fp = NULL;
    char *niske[MAX_NISKI];
    char **p = NULL;
    char x[MAX_DUZINA];

    /* Otvaranje datoteke */
    66  if ((fp = fopen("niske.txt", "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
    68  exit(EXIT_FAILURE);
    }

    70  /* Citanje sadrzaja datoteke */
    i = 0;
    while (fscanf(fp, "%s", x) != EOF) {
    72  /* Alociranje dovoljne memorije za i-tu nisku */
        if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
    74  fprintf(stderr, "Greska pri alociranju niske\n");
    76  exit(EXIT_FAILURE);
        }
        /* Kopiranje procitane niske na svoje mesto */
    80  strcpy(niske[i], x);
        i++;
    82  }

    84  /* Zatvaranje datoteke */
    fclose(fp);
    n = i;

    88  /* Sortiramo niske leksikografski, tako sto biblioteckoj
        funkciji qsort prosledjujemo funkciju kojom se zadaje
    90  kriterijum poredjenja 2 niske po duzini */
    qsort(niske, n, sizeof(char *), &poredi_leksikografski);

    92  printf("Leksikografski sortirane niske:\n");
    for (i = 0; i < n; i++)
    94  printf("%s ", niske[i]);
    96  printf("\n");

    98  /* Unosimo trazeni nisku */
    printf("Uneti trazenu nisku: ");
    100 scanf("%s", x);
```

```

102  /* Binarna pretraga */
103  /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
104     jer nam je niz sortiran leksikografski. */
105  p = bsearch(x, niske, n, sizeof(char *),
106             &poredi_leksikografski_b);

108  if (p != NULL)
109     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
110           *p, p - niske);
111  else
112     printf("Niska nije pronadjena u nizu\n");

114  /* Linearna pretraga */
115  /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
116     jer nam je niz sortiran leksikografski. */
117  p = lfind(x, niske, &n, sizeof(char *),
118           &poredi_leksikografski_b);

120  if (p != NULL)
121     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
122           *p, p - niske);
123  else
124     printf("Niska nije pronadjena u nizu\n");

126  /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
127     funkciju poredjenja */
128  qsort(niske, n, sizeof(char *), &poredi_duzine);

130  printf("Niske sortirane po duzini:\n");
131  for (i = 0; i < n; i++)
132     printf("%s ", niske[i]);
133  printf("\n");

134  /* Oslobadjanje zauzete memorije */
135  for (i = 0; i < n; i++)
136     free(niske[i]);

138  exit(EXIT_SUCCESS);
140 }

```

### Rešenje 3.32

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <search.h>

6  #define MAX 500

8  /* Struktura koja nam je neophodna za sve informacije o
   pojedinaenom studentu */
10 typedef struct {
11     char ime[21];
12     char prezime[21];

```

### 3 Algoritmi pretrage i sortiranja

---

```
14     int bodovi;
15 } Student;

16 /* Funkcija poredjenja koju cemo koristiti za sortiranje po
17    broju bodova, a studente sa istim brojevem bodova dodatno
18    sortiramo leksikografski po prezimenu */
19 int compare(const void *a, const void *b)
20 {
21     Student *prvi = (Student *) a;
22     Student *drugi = (Student *) b;

23
24     if (prvi->bodovi > drugi->bodovi)
25         return -1;
26     else if (prvi->bodovi < drugi->bodovi)
27         return 1;
28     else
29         /* Jednaki su po broju bodova, treba ih uporediti po
30            prezimenu */
31         return strcmp(prvi->prezime, drugi->prezime);
32 }

33
34 /* Funkcija za poredjenje koje ce porediti samo po broju bodova
35    Prvi parametar je ono sto trazimo u nizu, ovde je to broj
36    bodova, a drugi parametar ce biti element niza ciji se bodovi
37    porede. */
38 int compare_zabsearch(const void *a, const void *b)
39 {
40     int bodovi = *(int *) a;
41     Student *s = (Student *) b;
42     return s->bodovi - bodovi;
43 }

44
45 /* Funkcija za poredjenje koje ce porediti samo po prezimenu
46    Prvi parametar je ono sto trazimo u nizu, ovde je to prezime,
47    * a drugi parametar ce biti element niza cije se prezime
48    poredi. */
49 int compare_zalinearnaprezimena(const void *a, const void *b)
50 {
51     char *prezime = (char *) a;
52     Student *s = (Student *) b;
53     return strcmp(prezime, s->prezime);
54 }

55
56 int main(int argc, char *argv[])
57 {
58     Student kolokvijum[MAX];
59     int i;
60     size_t br_studenata = 0;
61     Student *nadjen = NULL;
62     FILE *fp = NULL;
63     int bodovi;
64     char prezime[21];

65
66     /* Ako je program pozvan sa nedovoljnim brojem argumenata
67        informisemo korisnika kako se program koristi i prekidamo
```



```

    izvrsavanje. */
70 if (argc < 2) {
    fprintf(stderr,
72         "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
            argv[0]);
74     exit(EXIT_FAILURE);
}
76
/* Otvaranje datoteke */
78 if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
80     exit(EXIT_FAILURE);
}
82
/* Ucitavanje sadrzaja */
84 for (i = 0;
    fscanf(fp, "%s%s%d", kolokvijum[i].ime,
86         kolokvijum[i].prezime,
            &kolokvijum[i].bodovi) != EOF; i++);
88
/* Zatvaranje datoteke */
90 fclose(fp);
br_studenata = i;
92
/* Sortiramo niz studenata po broju bodova, pa unutar grupe
94     studenata sa istim brojem bodova sortiranje se vrši po
    prezimenu */
96 qsort(kolokvijum, br_studenata, sizeof(Student), &compare);

98 printf("Studenti sortirani po broju poena opadajuće, ");
printf("pa po prezimenu rastuće:\n");
100 for (i = 0; i < br_studenata; i++)
    printf("%s %s %d\n", kolokvijum[i].ime,
102         kolokvijum[i].prezime, kolokvijum[i].bodovi);

104 /* Pretraživanje studenata po broju bodova se vrši binarnom
    pretragom jer je niz sortiran po broju bodova. */
106 printf("Unesite broj bodova: ");
scanf("%d", &bodovi);
108
nadjen =
110     bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
        &compare_za_bsearch);
112
if (nadjen != NULL)
114     printf
        ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
116         nadjen->ime, nadjen->prezime, nadjen->bodovi);
else
118     printf("Nema studenta sa unetim brojem bodova\n");

120 /* Pretraga po prezimenu se mora vršiti linearnom pretragom
    jer nam je niz sortiran po bodovima, globalno gledano. */
122 printf("Unesite prezime: ");
scanf("%s", prezime);
124

```

### 3 Algoritmi pretrage i sortiranja

```
126     nadjen =  
        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),  
              &compare_za_linearna_prezimana);  
128  
129     if (nadjen != NULL)  
130         printf  
131             ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",  
132              nadjen->ime, nadjen->prezime, nadjen->bodovi);  
133     else  
134         printf("Nema studenta sa unetim prezimenom\n");  
135  
136     return 0;  
}
```

#### Rešenje 3.33

```
1  #include<stdio.h>  
2  #include<string.h>  
3  #include <stdlib.h>  
4  
5  #define MAX 128  
6  
7  /* Funkcija poredi dva karaktera */  
8  int uporedi_char(const void *pa, const void *pb)  
9  {  
10     return *(char *) pa - *(char *) pb;  
11 }  
12  
13 /* Funkcija vraca: 1 - ako jesu anagrami 0 - inace */  
14 int anagrami(char s[], char t[], int n_s, int n_t)  
15 {  
16     /* Ako dve niske imaju razlicitu duzinu => nisu anagrami */  
17     if (n_s != n_t)  
18         return 0;  
19  
20     /* Sortiramo niske */  
21     qsort(s, strlen(t) / sizeof(char), sizeof(char),  
22           &uporedi_char);  
23     qsort(t, strlen(t) / sizeof(char), sizeof(char),  
24           &uporedi_char);  
25  
26     /* Ako su niske nakon sortiranja iste => jesu anagrami, u  
27        suprotnom, nisu */  
28     return !strcmp(s, t);  
29 }  
30  
31 int main()  
32 {  
33     char s[MAX], t[MAX];  
34  
35     /* Unose se dve niske sa ulaza */  
36     printf("Unesite dve niske: ");  
37     scanf("%s %s", s, t);  
38  
39     /* Ispituje se da li su niske anagrami */
```

```

    if (anagrami(s, t, strlen(s), strlen(t)))
41     printf("jesu\n");
    else
43     printf("nisu\n");
45     return 0;
}

```

### Rešenje 3.34

```

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

4
#define MAX 10
6  #define MAX_DUZINA 32

8  /* Funkcija poredi dve niske (stringove) */
  int uporedi_niske(const void *pa, const void *pb)
10 {
    return strcmp((char *) pa, (char *) pb);
12 }

14 int main()
  {
16     int i, n;
    char S[MAX][MAX_DUZINA];

18
    /* Unos broja niski */
20     printf("Unesite broj niski:");
    scanf("%d", &n);

22
    /* Unos niza niski */
24     printf("Unesite niske:\n");
    for (i = 0; i < n; i++)
26         scanf("%s", S[i]);

28
    /* Sortiramo niz niski */
    qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
30

    /******
32     Ovaj deo je iskomentarisano jer se u zadatku ne trazi ispis
    sortiranih niski. Koriscen je samo u fazi testiranja programa.

34
    printf("Sortirane niske su:\n");
    for(i = 0; i < n; i++)
36         printf("%s ", S[i]);
38     *****/

40
    /* Ako postoje dve iste niske u nizu, onda ce one nakon
    sortiranja niza biti jedna do druge */
42     for (i = 0; i < n - 1; i++)
        if (strcmp(S[i], S[i + 1]) == 0) {
44         printf("ima\n");
        return 0;
    }

```

### 3 Algoritmi pretrage i sortiranja

```
46     }  
48     printf("nema\n");  
49     return 0;  
50 }
```

#### Rešenje 3.35

```
1  #include<stdio.h>  
2  #include<stdlib.h>  
3  #include<string.h>  
  
5  /* Struktura koja predstavlja jednog studenta */  
6  typedef struct student {  
7      char nalog[8];  
8      char ime[21];  
9      char prezime[21];  
10     int poeni;  
11 } Student;  
  
13  
14 /* Funkcija poredi studente prema broju poena, rastuce */  
15 int uporedi_poeni(const void *a, const void *b)  
16 {  
17     Student s = *(Student *) a;  
18     Student t = *(Student *) b;  
19     return s.poeni - t.poeni;  
20 }  
  
23 /* Funkcija poredi studente prvo prema godini, zatim prema smeru  
24    i na kraju prema indeksu */  
25 int uporedi_nalog(const void *a, const void *b)  
26 {  
27     Student s = *(Student *) a;  
28     Student t = *(Student *) b;  
29     /* Za svakog studenta iz naloga izdvajamo godinu upisa, smer i  
30        broj indeksa */  
31     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';  
32     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';  
33     char smer1 = s.nalog[1];  
34     char smer2 = t.nalog[1];  
35     int indeks1 =  
36         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +  
37         s.nalog[6] - '0';  
38     int indeks2 =  
39         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +  
40         t.nalog[6] - '0';  
41     if (godina1 != godina2)  
42         return godina1 - godina2;  
43     else if (smer1 != smer2)  
44         return smer1 - smer2;  
45     else  
46         return indeks1 - indeks2;  
47 }
```

```

49 int uporedi_bsearch(const void *a, const void *b)
50 {
51     /* Nalog studenta koji se trazi */
52     char *nalog = (char *) a;
53     /* Kljuc pretrage */
54     Student s = *(Student *) b;
55
56     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
57     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
58     char smer1 = nalog[1];
59     char smer2 = s.nalog[1];
60     int indeks1 =
61         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] -
62         '0';
63     int indeks2 =
64         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
65         s.nalog[6] - '0';
66     if (godina1 != godina2)
67         return godina1 - godina2;
68     else if (smer1 != smer2)
69         return smer1 - smer2;
70     else
71         return indeks1 - indeks2;
72 }
73
74 int main(int argc, char **argv)
75 {
76     Student *nadjen = NULL;
77     char nalog_trazeni[8];
78     Student niz_studenata[100];
79     int i = 0, br_studenata = 0;
80     FILE *in = NULL, *out = NULL;
81
82     /* Ako je broj argumenata komandne linije razlicit i od 2 i od
83        3, korisnik nije ispravno pozvao program i prijavljujemo
84        gresku: */
85     if (argc != 2 && argc != 3) {
86         fprintf(stderr,
87             "Greska! Program se poziva sa: ./a.out -opcija (nalog)!\\
88             n");
89         exit(EXIT_FAILURE);
90     }
91
92     /* Otvaranje datoteke za citanje */
93     in = fopen("studenti.txt", "r");
94     if (in == NULL) {
95         fprintf(stderr,
96             "Greska prilikom otvarnja datoteke studenti.txt!\\n");
97         exit(EXIT_FAILURE);
98     }
99
100    /* Otvaranje datoteke za pisanje */
101    out = fopen("izlaz.txt", "w");
102    if (out == NULL) {
103        fprintf(stderr,

```

### 3 Algoritmi pretrage i sortiranja

```
103         "Greska prilikom otvaranja datoteke izlaz.txt!\n");
104     exit(EXIT_FAILURE);
105 }
106
107 /* Ucitavamo studente iz ulazne datoteke sve do njenog kraja */
108 while (fscanf
109     (in, "%s %s %s %d", niz_studenata[i].nalog,
110      niz_studenata[i].ime, niz_studenata[i].prezime,
111      &niz_studenata[i].poeni) != EOF)
112     i++;
113
114 br_studenata = i;
115
116 /* Ako je student uneo opciju -p, vrsi se sortiranje po
117    poenima */
118 if (strcmp(argv[1], "-p") == 0)
119     qsort(niz_studenata, br_studenata, sizeof(Student),
120          &uporedi_poeni);
121 /* A ako je uneo opciju -n, vrsi se sortiranje po nalogu */
122 else if (strcmp(argv[1], "-n") == 0)
123     qsort(niz_studenata, br_studenata, sizeof(Student),
124          &uporedi_nalog);
125
126 /* Sortirani studenti se ispisuju u izlaznu datoteku */
127 for (i = 0; i < br_studenata; i++)
128     fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
129          niz_studenata[i].ime, niz_studenata[i].prezime,
130          niz_studenata[i].poeni);
131
132 /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
133    studenta... */
134 if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
135     strcpy(nalog_trazeni, argv[2]);
136
137     /* ... pronalazi se student sa tim nalogom... */
138     nadjen =
139         (Student *) bsearch(nalog_trazeni, niz_studenata,
140                             br_studenata, sizeof(Student),
141                             &uporedi_bsearch);
142
143     if (nadjen == NULL)
144         printf("Nije nadjen!\n");
145     else
146         printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
147             nadjen->prezime, nadjen->poeni);
148 }
149
150 /* Zatvaranje datoteka */
151 fclose(in);
152 fclose(out);
153
154 return 0;
155 }
```

#### Rešenje 3.37

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

/* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
   standardnog ulaza */
void ucitaj_matricu(int **a, int n, int m)
{
    printf("Unesite elemente matrice po vrstama:\n");
    int i, j;

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

/* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
   kolona */
int zbir_vrste(int **a, int v, int m)
{
    int i, zbir = 0;

    for (i = 0; i < m; i++) {
        zbir += a[v][i];
    }
    return zbir;
}

/* Funkcija koja sortira vrste matrice na osnovu zbira
   koriscenjem selection algoritma */
void sortiraj_vrste(int **a, int n, int m)
{
    int i, j, min;

    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++) {
            if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
                min = j;
            }
        }
        if (min != i) {
            int *tmp;
            tmp = a[i];
            a[i] = a[min];
            a[min] = tmp;
        }
    }
}

/* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
   standardni izlaz */
void ispis_i_matricu(int **a, int n, int m)
{

```

### 3 Algoritmi pretrage i sortiranja

---

```
56     int i, j;

58     for (i = 0; i < n; i++) {
59         for (j = 0; j < m; j++) {
60             printf("%d ", a[i][j]);
61         }
62         printf("\n");
63     }
64 }

66 /* Funkcija koja alokira memoriju za matricu dimenzija nxm */
67 int **alociraj_memoriju(int n, int m)
68 {
69     int i, j;
70     int **a;

72     a = (int **) malloc(n * sizeof(int *));
73     if (a == NULL) {
74         fprintf(stderr, "Problem sa alokacijom memorije!\n");
75         exit(EXIT_FAILURE);
76     }
77     /* Za svaku vrstu ponaosob */
78     for (i = 0; i < n; i++) {
80         /* Alociramo memoriju */
81         a[i] = (int *) malloc(m * sizeof(int));
82         /* Proveravamo da li je doslo do greske prilikom alokacije */
83         if (a[i] == NULL) {
84             /* Ako jeste, ispisujemo poruku */
85             fprintf(stderr, "Problem sa alokacijom memorije!\n");
86             /* I oslobadjamo memoriju zauzetu do ovog koraka */
87             for (j = 0; j < i; j++) {
88                 free(a[j]);
89             }
90             free(a);
91             exit(EXIT_FAILURE);
92         }
93     }
94     return a;
95 }

96

97
98
99
100 /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije
101     nxm */
102 void oslobodi_memoriju(int **a, int n, int m)
103 {
104     int i;

105     for (i = 0; i < n; i++) {
106         free(a[i]);
107     }
108     free(a);
109 }
110 }
```



```
112 |
114 |
116 | int main(int argc, char *argv[])
118 | {
120 |     int **a;
122 |     int n, m;
124 |
126 |     /* Citamo dimenzije matrice */
128 |     printf("Unesite dimenzije matrice: ");
130 |     scanf("%d %d", &n, &m);
132 |
134 |     /* Alociramo memoriju */
136 |     a = alociraj_memoriju(n, m);
138 |
140 |     /* Ucitavamo elemente matrice */
142 |     ucitaj_matricu(a, n, m);
144 |
146 |     /* Pozivamo funkciju koja sortira vrste matrice prema zbiru */
148 |     sortiraj_vrste(a, n, m);
150 |
152 |     /* Ispisujemo rezultujuću matricu */
154 |     printf("Sortirana matrica je:\n")
156 |         ispisi_matricu(a, n, m);
158 |
160 |     /* Oslobadjamo memoriju */
162 |     oslobodi_memoriju(a, n, m);
164 |
166 |     /* I prekidamo sa izvršavanjem programa */
168 |     return 0;
170 | }
```



## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

**Zadatak 4.1** Napisati program koji koristi jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (a) Definirati strukturu `Cvor` koja predstavlja čvor liste.
- (b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vraća adresu novo kreiranog čvora.
- (c) Napisati funkciju koja dodaje novi element na početak liste.
- (d) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama `[ ]` i međusobno razdvojene zapetama.
- (e) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- (f) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- (g) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. **Ana:** Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i iterativno rešenje

### *Upotreba programa 1*

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3
    3 17 3 1 9 3
  Unosite element koji se trazi u listi: 17
  Unosite element koji se brise iz liste: 3
Izlaz:
  Lista: []
  Lista: [2]
  Lista: [3, 2]
  Lista: [14, 3, 2]
  Lista: [5, 14, 3, 2]
  Lista: [3, 5, 14, 3, 2]
  Lista: [3, 3, 5, 14, 3, 2]
  Lista: [17, 3, 3, 5, 14, 3, 2]
  Lista: [3, 17, 3, 3, 5, 14, 3, 2]
  Lista: [1, 3, 17, 3, 3, 5, 14, 3, 2]
  Lista: [9, 1, 3, 17, 3, 3, 5, 14, 3, 2]
  Lista: [3, 9, 1, 3, 17, 3, 3, 5, 14, 3, 2]

  Trazeni broj 17 je u listi!
  Lista nakon brisanja: [9, 1, 17, 5, 14, 2]
```

### *Upotreba programa 2*

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 23 14 35
  Unosite element koji se trazi u listi: 8
  Unosite element koji se brise iz liste: 3
Izlaz:
  Lista: []
  Lista: [23]
  Lista: [14, 23]
  Lista: [35, 14, 23]

  Element NIJE u listi!
  Lista nakon brisanja: [35, 14, 23]
```

### *Upotreba programa 3*

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
  Unosite element koji se trazi u listi: 1
  Unosite element koji se brise iz liste: 12
Izlaz:
  Lista: []

  Element NIJE u listi!
  Lista nakon brisanja: []
```

[Rešenje 4.1]

**Zadatak 4.2** Napisati program koji koristi jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (a) Definirati strukturu `Cvor` koja predstavlja čvor liste.
- (b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vraća adresu novo kreiranog čvora.
- (c) Napisati funkciju koja dodaje novi element na kraj liste.
- (d) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama `[, ]` i međusobno razdvojene zapetama.
- (e) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekursivno. **Ana:** Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekursivno i iterativno rešenje

#### Upotreba programa 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3
    3 17 3 1 9 3
Izlaz:
  Lista: []
  Lista: [2]
  Lista: [2, 15]
  Lista: [2, 15, 4]
  Lista: [2, 15, 4, 8]
  Lista: [2, 15, 4, 8, 3]
  Lista: [2, 15, 4, 8, 3, 24]
  Lista: [2, 15, 4, 8, 3, 24, 11]
  Lista: [2, 15, 4, 8, 3, 24, 11, 17]
  Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4]
  Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4, 3]
  Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4, 3, 7]
```

#### Upotreba programa 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
Izlaz:
  Lista: []
```

[Rešenje 4.2]

**Zadatak 4.3** Napisati program koji koristi jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (a) Definirati strukturu `Cvor` koja predstavlja čvor liste.
- (b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vraća adresu novo kreiranog čvora.
- (c) Napisati funkciju koja dodaje novi elemenat u listu tako da lista ostane rastuće sortirana.
- (d) Napisati funkciju koja oslobađa memoriju koju je zauzela lista.
- (e) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama  
,  
i međusobno razdvojene zapetama.
- (f) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- (g) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- (h) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. **Ana: Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i iterativno rešenje**

#### Upotreba programa 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3
    3 17 3 1 9 3
  Unesite element koji se trazi u listi: 5
  Unesite element koji se brise iz liste: 3
Izlaz:
  Lista: []
  Lista: [2]
  Lista: [2, 3]
  Lista: [2, 3, 14]
  Lista: [2, 3, 5, 14]
  Lista: [2, 3, 3, 5, 14]
  Lista: [2, 3, 3, 3, 5, 14]
  Lista: [2, 3, 3, 3, 5, 14, 17]
  Lista: [2, 3, 3, 3, 3, 5, 14, 17]
  Lista: [1, 2, 3, 3, 3, 3, 5, 14, 17]
  Lista: [1, 2, 3, 3, 3, 3, 5, 9, 14, 17]
  Lista: [1, 2, 3, 3, 3, 3, 3, 5, 9, 14, 17]

  Trazeni broj 5 je u listi!
  Lista nakon brisanja: [1, 2, 5, 9, 14, 17]
```

*Upotreba programa 2*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 23 14 35
  Unosite element koji se trazi u listi: 8
  Unosite element koji se brise iz liste: 3
Izlaz:
  Lista: []
        Lista: [23]
        Lista: [14, 23]
        Lista: [14, 23, 35]

        Element NIJE u listi!
        Lista nakon brisanja:  [14, 23, 35]

```

*Upotreba programa 3*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
  Unosite element koji se trazi u listi: 1
  Unosite element koji se brise iz liste: 12
Izlaz:
        Lista: []

        Element NIJE u listi!
        Lista nakon brisanja:  []

```

[Rešenje 4.3]

**Zadatak 4.4** Napisati program koji koristi dvostruko povezanu listu za čuvanje celih brojeva koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu se prekida učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. *I ovde isto mozda razdvojiti sortiranost od obične liste.*

- Napisati funkciju koja dodaje novi elemenat na početak liste.
- Napisati funkciju koja dodaje novi elemenat na kraj liste.
- Napisati funkciju koja dodaje novi elemenat u listu tako da lista ostane rastuće sortirana.
- Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom implementirati iterativno.

**Zadatak 4.5** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

*Test 1*

```
Datoteka: {[23 + 5344] * (24 - 234)} - 23
Izlaz:   Zagrade su ispravno uparene.
```

*Test 2*

```
Datoteka: {[23 + 5] * (9 * 2)} - {23}
Izlaz:   Zagrade su ispravno uparene.
```

*Test 3*

```
Datoteka: {[2 + 54] / (24 * 87)} + (234 + 23)
Izlaz:   Zagrade nisu ispravno uparene.
```

*Test 3*

```
Datoteka: {(2 - 14) / (23 + 11)} * (2 + 13)
Izlaz:   Zagrade nisu ispravno uparene.
```

*Test 4*

```
Datoteka je prazna.
Izlaz:   Zagrade su ispravno uparene.
```

*Test 5*

```
Datoteka ne postoji.
Izlaz:   Greska prilikom otvaranja datoteke izraz.txt!
```

**Zadatak 4.6** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije . **Milena: A sta ako se ne navede argument komandne linije?** Uputstvo: za rešavanje problema koristiti stek implementiran preko listi čiji su čvorovi HTML etikete.



*Test 1*

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    A sutra ce biti jos lepsi.
    <a link="http://www.google.com"> Link 1</a>
    <a link="http://www.math.rs"> Link 2</a>
  </body>
</html>
Izlaz: Ispravno uparene etikete.
```

*Test 2*

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
</html>
Izlaz: Neispravno uparene etikete.
```

*Test 3*

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
</body>
Izlaz: Neispravno uparene etikete.
```

*Test 4*

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva sa: ./a.out datoteka.html!
```

*Test 5*

```
Poziv: ./a.out datoteka.html
Datoteka.html ne postoji.
Izlaz: Greska prilikom otvaranja datoteke datoteka.html.
```

### Test 6

```
Poziv: ./a.out datoteka.html
Datoteka.html je prazna
Izlaz: Ispravno uparene etikete.
```

**Zadatak 4.7** Milena: Problem sa ovim zadatkom je sto je program najpre na usluzi korisnicima, a zatim na usluzi sluzbeniku i to nekako zbunjuje u formulaicji. Formulacija mi nije bila jasna bez citanja resenja, pokusala sam da je preciziran, u nastavku je izmenjena formulacija.

Medjutim, ja i dalje nisam bas zadovoljna i zato predlazem da se formulacija izmeni tako da je program stalno na usluzi sluzbeniku. Program ucitava podatke o prijavljenim korisnicima iz datoteke. Sluzbenik odlucuje da li ce da obradjuje redom korisnike, ili ce u nekim situacijama da odlozi rad sa korisnikom i stavi ga na kraj reda. Program ga uvek pita da na osnovu jmbg-a i zahteva odluci da li ce ga staviti na kraj reda, ako hoce, on ide na kraj reda, ako nece, onda sluzbenik daje odgovor na zahtev i jmbg, zahtev i odgovor se upisuju u izlaznu datoteku.

Napisati program kojim se simulira rad jednog šaltera na kojem se prvo zakazuju termini, a potom službenik uslužuje korisnike redom, kako su se prijavljivali.

Korisnik se prijavljuje unošenjem svog jmbg broja (niska koja sadrži 13 karaktera) i zahteva (niska koja sadrži najviše 999 karaktera). Prijavljivanje korisnika se prekida unošenjem karaktera za kraj ulaza (EOF).

Službenik redom proziva korisnike čitanjem njihovog jmbg broja, a zatim odlučuje da li korisnika vraća na kraj reda ili ga odmah uslužuje. Službeniku se postavlja pitanje **Da li korisnika vracate na kraj reda?** i ukoliko on da odgovor **Da**, korisnik se vraća na kraj reda. Ukoliko odgovor nije **Da**, tada službenik čita korisnikov zahtev. Posle svakog 10 usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu.

Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.

**Zadatak 4.8** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etiketete smeštati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
{
    unsigned broj_pojavljivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;
```

*Test 1*

```

Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    A sutra ce biti jos lepsi.
    <a link="http://www.google.com"> Link 1</a>
    <a link="http://www.math.rs"> Link 2</a>
  </body>
</html>

Izlaz:  a - 4
        br - 1
        h1 - 2
        body - 2
        title - 2
        head - 2
        html - 2

```

*Test 2*

```

Poziv: ./a.out
Izlaz: Greska! Program se poziva sa: ./a.out datoteka.html!

```

*Test 3*

```

Poziv: ./a.out datoteka.html
Datoteka.html ne postoji.
Izlaz: Greska prilikom otvaranja datoteke datoteka.html.

```

*Test 4*

```

Poziv: ./a.out datoteka.html
Datoteka.html je prazna.
Izlaz:

```

**Zadatak 4.9** Milena: malo me mucu u ovom zadatku sto nema neki smisao. Naime, ako se samo vrsi ucitavanje iz datoteka i ispisivanje, onda su ove liste zapravo visak jer isti rezultat moze da se dobije i bez koriscenja listi. Zato mi fali da program uradi nesto sto ne bi mogao da uradi bez koriscenja listi, npr da na osnovu unetog broja ispisuje svaki n-ti broj rezultujuce liste pa to u nekoj petlji da korisnik moze da ispisuje za razlicite unete n ili tako nesto...

Napisati program koji objedinjuje dve sortirane liste. Funkcija ne treba da kreira nove čvorove, već da samo postojeće čvorove preraspodeli. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije

## 4 Dinamičke strukture podataka

---

se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

### *Test 1*

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz: 2 4 5 6 6 10 11 12 14 15
       16
```

### *Test 2*

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva
      sa: ./a.out dat1.txt dat2.txt
      !
```

### *Test 3*

```
Poziv: ./a.out dat1.txt
Izlaz: Greska! Program se poziva
      sa: ./a.out dat1.txt dat2.txt
      !
```

### *Test 4*

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt ne postoji
Izlaz: Greska prilikom otvaranja
      datoteke dat2.txt.
```

### *Test 5*

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt ne postoji
dat2.txt: 2 4 6 10 15
Izlaz: Greska prilikom otvaranja
      datoteke dat1.txt.
```

### *Test 6*

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz: 2 4 6 10 15
```

**Zadatak 4.10** Napisati funkciju koja formira listu studenata tako što se podaci o studentima učitavaju iz datoteke čije se ime zadaje kao argument komandne linije. U svakom redu datoteke nalaze se podaci o studentu i to broj indeksa, ime i prezime. Napisati rekurzivnu funkciju koja određuje da li neki student pripada listi ili ne. Ispisati zatim odgovarajuću poruku i rekurzivno osloboditi memoriju koju je data lista zauzimala. Student se traži na osnovu broja indeksa, koji se zadaje sa standardnog ulaza.

*Test 1*

```

Poziv: ./a.out studenti.txt
Datoteka:      Ulaz:      Izlaz:
123/2014 Marko Lukic      3/2014      da: Ana Sokic
3/2014 Ana Sokic      235/2008      ne
43/2013 Jelena Ilic      41/2009      da: Marija Zaric
41/2009 Marija Zaric
13/2010 Milovan Lazic

```

*Test 2*

```

Poziv: ./a.out
Izlaz: Greska! Program se poziva
sa: ./a.out studenti.txt!

```

*Test 5*

```

Poziv: ./a.out studenti.txt
studenti.txt ne postoji
Izlaz: Greska prilikom otvaranja
datoteke studenti.txt

```

*Test 5*

```

Poziv: ./a.out studenti.txt
studenti.txt prazna
Izlaz: ??? videti sta ce tacno
biti

```

**Zadatak 4.11** Neka su date dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od tih lista formira novu listu L koja sadrži alternirajući rasporedene elemente lista L1 i L2 (prvi element iz L1, prvi element iz L2, drugi element L1, drugi element L2, itd). Ne formirati nove čvorove, već samo postojeće čvorove rasporediti u jednu listu. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. *Milena: Sta ako je neka lista duza? To precizirati. I ovde me muci sto nedostaje neki smisao zadatku, nesto sto ne bi moglo da se uradi da nismo koristili liste.*

### Test 1

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz:  2 5 4 6 6 11 10 12 15 14
        16
```

### Test 2

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva
       sa: ./a.out dat1.txt dat2.txt
          !
```

### Test 3

```
Poziv: ./a.out dat1.txt
Izlaz: Greska! Program se poziva
       sa: ./a.out dat1.txt dat2.txt
          !
```

### Test 4

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt ne postoji
Izlaz: Greska prilikom otvaranja
       datoteke dat2.txt.
```

### Test 5

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt ne postoji
dat2.txt: 2 4 6 10 15
Izlaz: Greska prilikom otvaranja
       datoteke dat1.txt.
```

### Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz: 2 4 6 10 15
```

**Zadatak 4.12** Data je datoteka brojevi.txt koja sadrži cele brojeve.

- (a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.

- (b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maximalan strogo rastući podniz.

Napisati program koji u datoteku `Rezultat.txt` upisuje nađeni strogo rastući podniz. Milena: I ovde me muci sto bi zadatak mogao da se resi i bez koriscenja listi...

Milena: Prirodni oblik testa ovde bi bio horizontalan, a ne ovako vertikalan.

#### Test 1

```
Poziv: ./a.out
brojevi.txt
      43 12 15 16 4 2 8
Izlaz:
Rezultat.txt : 12 15 16
```

#### Test 2

```
Poziv: ./a.out
brojevi.txt ne postoji
Izlaz:
Rezultat.txt : Greska prilikom
              otvaranja datoteke dat2.txt.
```

#### Test 3

```
Poziv: ./a.out
brojevi.txt prazna
Izlaz:
Rezultat.txt je prazna.
```

#### Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz: 2 4 6 10 15
```

**Zadatak 4.13** Grupa od  $n$  plesača na kostimima imaju brojeve od 1 do  $n$ , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi  $k$ -ti plesač. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza.

Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. Uputstvo: u implementaciji koristiti kružnu listu.

### Test 1

```
Ulaz: 5 3
Izlaz: 3 1 5 2 4
```

**Zadatak 4.14** Grupa od  $n$  plesača na kostimima imaju brojeve od 1 do  $n$ , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi  $k$ -ti plesač. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje počinje od sledećeg suseda prethodno izbačenog, uz promenu smeru. Ukoliko se prilikom prethodnog izbacivanja odbrojavalu u smeru kazaljke na satu sada će se obbrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza.

Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. Uputstvo: u implementaciji koristiti dvostruko povezanu kružnu listu.

### Test 1

```
Ulaz: 5 3
Izlaz: 3 5 4 2 1
```

### Test 1

```
Ulaz: 2 7
Izlaz: n mora biti
       uvek vece od k, a
       2 < 7!
```

## 4.2 Stabla

**Zadatak 4.15** Napisati program za rad sa binarnim pretraživačkim stablima.

- (a) Definisati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog stabla koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo<sup>1</sup>.
- (b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.
- (c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.

---

<sup>1</sup>U zadacima ove glave u kojima nije eksplicitno naglašen sadržaj čvorova stabla, podrazumevaće se ova struktura.



- (d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.
- (e) Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- (f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.
- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

Korišćenjem prethodnih funkcija, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

#### Test 1

```

Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa): 7 2 1 9 32 18
Izlaz:
  Infiksni ispis: 1 2 7 9 18 32
  Prefiksni ispis: 7 2 1 9 32 18
  Postfiksni ispis: 1 2 18 32 9 7
  Trazi se broj: 11
  Broj se ne nalazi u stablu!
  Brise se broj: 7
  Rezultujuce stablo: 1 2 9 18 32

```

### Test 2

```
Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa): 8 -2 6 13 24 -3
Izlaz:
  Infiksni ispis:  -3 -2 6 8 13 24
  Prefiksni ispis: 8 -2 -3 6 13 24
  Postfiksni ispis: -3 6 -2 24 13 8
  Trazi se broj: 6
  Broj se nalazi u stablu!
  Brise se broj: 14
  Rezultujuce stablo: -3 -2 6 8 13 24
```

[Rešenje 4.15]

**Zadatak 4.16** Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog stabla uređenog leksikografski prema rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku **Nedostaje ime ulazne datoteke!**. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

### Test 1

```
Poziv: ./a.out test.txt
Datoteka test.txt:
  Sunce utorak raCunar SUNCE
  programiranje jabuka
  PROGramiranje sunCE JABUka
Izlaz:
  jabuka: 2
  programiranje: 2
  racunar: 1
  sunce: 3
  utorak: 1

  Najcesca rec: sunce (pojavljuje
    se 3 puta)
```

*Test 2*

```

Poziv: ./a.out suma.txt
Datoteka suma.txt:
    lipa zova hrast ZOVA breza LIPA
Izlaz:
    breza: 1
    hrast: 1
    lipa: 2
    zova: 2

    Najcesca rec: lipa (pojavljuje
        se 2 puta)

```

*Test 3*

```

Poziv: ./a.out
Izlaz:
    Nedostaje ime ulazne datoteke!

```

[Rešenje 4.16]

**Zadatak 4.17** U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. **Pera Peric** 064/123-4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči **KRAJ**, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

*Upotreba programa 1*

```

Poziv: ./a.out
Datoteka imenik.txt:
    Pera Peric 011/3240-987
    Marko Maric 064/1234-987
    Mirko Maric 011/589-333
    Sanja Savkovic 063/321-098
    Zika Zikic 021/759-858
Ulaz:
    Unesite ime datoteke: imenik.txt
    Unesite ime i prezime: Pera Peric
Izlaz:
    Broj je: 011/3240-987
Ulaz:
    Unesite ime i prezime: Marko Markovic
Izlaz:
    Broj nije u imeniku!
Ulaz:
    Unesite ime i prezime: KRAJ

```

[Rešenje 4.17]

**Zadatak 4.18** U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niz najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

### Test 1

```
Poziv: ./a.out
Datoteka prijemni.txt:
Marko Markovic 45.4 12.3 11
Milan Jevremovic 35.2 1.3 9
Maja Agic 60 19 20
Nadica Zec 54.2 10 15.8
Jovana Milic 23.3 2 5.6
Izlaz:
1. Maja Agic 60 19 20 99
2. Nadica Zec 54.2 10 15.8 80
3. Marko Markovic 45.4 12.3 11 68.7
4. Milan Jevremovic 35.2 1.3 9 45.5
-----
5. Jovana Milic 23.3 2 5.6 30.9
```

\* **Zadatak 4.19** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.YYYY.`

*Upotreba programa 1*

```

Poziv: a.out
Datoteka rodjendani.txt:
Marko Markovic 12.12.1990.
Milan Jevremovic 04.06.1989.
Maja Agic 23.04.2000.
Nadica Zec 01.01.1993.
Jovana Milic 05.05.1990.
Ulaz:
Unesite datum: 23.04.
Izlaz:
Slavljenik: Maja Agic
Ulaz:
Unesite datum: 01.01.
Izlaz:
Slavljenik: Nadica Zec
Ulaz:
Unesite datum: 01.05.
Izlaz:
Slavljeni: Jovana Milic 05.05.
Ulaz:
Unesite datum: CTRL+D

```

**Zadatak 4.20** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0.

*Test 1*

```

Poziv: ./a.out
Ulaz:
Prvo stablo: 10 5 15 3 2 4 30 12 14 13 0
Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0
Izlaz:
Stabla jesu identicna.

```

*Test 2*

```

Poziv: ./a.out
Ulaz:
Prvo stablo: 10 5 15 4 3 2 30 12 14 13 0
Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0
Izlaz:
Stabla nisu identicna.

```

\* **Zadatak 4.21** Napisati program koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla

uračunata tačno po jednom. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

### Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 1 7 8 9 2 2
  Drugo stablo: 3 9 6 11 1
Izlaz:
  Unija: 1 2 3 6 7 8 9 11
  Presek: 1 9
  Razlika: 2 7 8
```

**Zadatak 4.22** Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

### Test 1

```
Poziv: ./a.out
Ulaz:
  n: 7
  a: 1 11 8 6 37 25 30
Izlaz:
  1 6 8 11 25 30 37
```

### Test 2

```
Poziv: ./a.out
Ulaz:
  n: 55
Izlaz:
  Greska: pogresna dimenzija niza!
```

**Zadatak 4.23** Dato je binarno pretraživačko stablo celih brojeva.

- (a) Napisati funkciju koja izračunava broj čvorova stabla.
- (b) Napisati funkciju koja izračunava broj listova stabla.
- (c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.
- (d) Napisati funkciju koja izračunava zbir čvorova stabla.
- (e) Napisati funkciju koja izračunava najveći element stabla.
- (f) Napisati funkciju koja izračunava dubinu stabla.
- (g) Napisati funkciju koja izračunava broj čvorova na  $i$ -tom nivou stabla.

- (h) Napisati funkciju koja ispisuje sve elemente na  $i$ -tom nivou stabla.
- (i) Napisati funkciju koja izračunava maksimalnu vrednost na  $i$ -tom nivou stabla.
- (j) Napisati funkciju koja izračunava zbir čvorova na  $i$ -tom nivou stabla.
- (k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti  $x$ .

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara  $i$  i  $x$  pročitati kao argumente komandne linije.

#### Test 2

```
Poziv: ./a.out 2 15
Ulaz:
 10 5 15 3 2 4 30 12 14 13
Izlaz:
 broj cvorova: 10
 broj listova: 4
 pozitivni listovi: 2 4 13 30
 zbir cvorova: 108
 najveći element: 30
 dubina stabla: 5
 broj cvorova na 2. nivou: 3
 elementi na 2. nivou: 3 12 30
 maksimalni na 2. nivou: 30
 zbir na 2. nivou: 45
 zbir elemenata manjih ili jednakih od 15: 7
```

**Zadatak 4.24** Dato je binarno pretraživačko stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa maksimalnim proizvodom vrednosti iz desnog podstabla.
- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjom sumom vrednosti iz levog podstabla.
- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gorenavedene funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza.

### Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  Cvor sa maksimalnim desnim proizvodom: 10
  Cvor sa najmanjom levom sumom: 2
  Putanja do najdubljeg cvora: 10 15 12 14 13
  Putanja do najmanjeg cvora: 10 5 3 2
```

**Zadatak 4.25** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima.

### Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  0.nivo: 10
  1.nivo: 5 15
  2.nivo: 3 12 30
  3.nivo: 2 4 14
  4.nivo: 13
```

\* **Zadatak 4.26** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

### Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 14 30 1 0
Izlaz:
  Stabla su slicna kao u ogledalu
  .
```



*Test 2*

```

Poziv: ./a.out
Ulaz:
    Prvo stablo: 11 20 5 3 0
    Drugo stablo: 8 20 15 0
Izlaz:
    Stabla nisu slicna kao u
    ogledalu.

```

**Zadatak 4.27** AVL-stablo je binarno stablo pretrage kod koga apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

*Test 1*

```

Poziv: ./a.out
Ulaz:
    10 5 15 2 11 16 1 13
Izlaz:
    7

```

*Test 2*

```

Poziv: ./a.out
Ulaz:
    16 30 40 24 10 18 45 22
Izlaz:
    6

```

**Zadatak 4.28** Binarno stablo se naziva HEAP ako je kompletno (svaki njegov čvor, izuzev listova, ima i levog i desnog potomka) i za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva HEAP. Napisati zatim i glavni program koji ispisuje rezultat `heap` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

*Test 1*

```

Poziv: ./a.out
Ulaz:
    100 19 36 17 3 25 1 2 7
Izlaz:
    Stablo je heap.

```

## 4.3 Rešenja

### Rešenje 4.1

```

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

```

```
4  /* NAPOMENA:
   Jednostruko povezana lista je struktura podataka
6  koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
   podatak (određenog tipa) i pokazivac na sledeci cvor u
8  sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
   liste (bez glave) je takodje lista, i naziva se rep liste.
10  Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
   baratanja listom mi cuvamo samo pokazivac na glavu liste.
12  Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
   sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
14  pristupimo, u njemu je sadržana adresa sledeceg elementa, pa
   preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
16  element u listi nema sledeci element: u tom slucaju se
   njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
18  lista se predstavlja NULL pokazivacem.

20  Prednost koriscenja povezanih lista u odnosu na dinamicki
   niz je u tome sto se elementi mogu efikasno umetati i brisati
22  sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
   premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
24  ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
   elementi moraju obradivati redom (iteracijom kroz listu).

26

28  Prilikom promene liste (dodavanje novog elementa, brisanje
   elementa,
   premestanje elemenata, itd.) postoji mogucnost da glava liste
30  bude
   promenjena, tj. da to postane neki drugi cvor (sa drugom adresom)
   .
   U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
32  promenu liste obavljamo u posebnoj funkciji onda je potrebno da
   se
   pozivajucoj funkciji vrati azurirana informacija o adresi glave
   liste.

34

36  Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
   pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
   cuva
   adresu glave i koju, eventualno, treba azurirati.
38  Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
   promeni promenljivu pozivajuce funkcije direktno. Npr:
40          funkcija_za_promenu(&pok, ...);
   */

42  /* Struktura koja predstavlja cvor liste */
44  typedef struct cvor {
   /* Podatak koji cvor sadrzi */
46      int vrednost;
   /* Pokazivac na sledeci cvor liste */
48      struct cvor *sledeci;
   } Cvor;

50

52
```

```

54  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
55     novog cvora inicijalizuje na broj, dok pokazivac na
56     sledeci cvor u novom cvoru postavlja na NULL.
57     Funkcija vraca pokazivac na novokreirani cvor ili NULL
58     ako alokacija nije uspesno izvorsena. */
59  Cvor * napravi_cvor(int broj) {
60      Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
61      if( novi == NULL )
62          return NULL;
63
64      /* Inicijalizacija polja u novom cvoru */
65      novi->vrednost = broj;
66      novi->sledeci = NULL;
67      return novi;
68  }
69
70
71  /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
72     ciji se pocetni cvor nalazi na adresi adresa_glave. */
73  void oslobodi_listu(Cvor ** adresa_glave) {
74      Cvor *pomocni = NULL;
75
76      /* Ako lista nije prazna, onda ima memorije koju treba
77         osloboditi */
78      while (*adresa_glave != NULL) {
79          /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
80             a tek onda osloboditi element koji predstavlja glavu liste */
81          pomocni = (*adresa_glave)->sledeci;
82          free(*adresa_glave);
83          /* Sledeci element je nova glava liste */
84          *adresa_glave = pomocni;
85      }
86  }
87
88
89  /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
90     ukoliko
91     alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
92     memorija
93     za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
94  void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
95      /* Ukoliko je novi NULL */
96      if ( novi == NULL ) {
97          fprintf(stderr, "Neuspela alokacija za nov cvor\n");
98          /* Oslobadjamo svu dinamički alociranu memoriju i prekidamo
99             program */
100          oslobodi_listu(adresa_glave);
101          exit(EXIT_FAILURE);
102      }
103  }
104

```

```
106 /* Funkcija dodaje novi cvor na pocetak liste.  
    Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga  
    na pocetak */  
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {  
    /* Kreiramo nov cvor i proveravamo da li je bilo greske pri  
    alokaciji */  
110    Cvor *novi = napravi_cvor(broj);  
    prover_i_alokaciju(adresa_glave, novi);  
112  
    /* Uvezujemo novi cvor na pocetak */  
114    novi->sledeci = *adresa_glave;  
    /* Nov cvor je sada nova glava liste */  
116    *adresa_glave = novi;  
118 }  
  
120 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,  
    ili NULL ukoliko je lista prazna */  
122 Cvor* pronadji_poslednji (Cvor* glava) {  
124    /* ako je lista prazna, nema ni poslednjeg cvor  
    i u tom slucaju vracamo NULL.*/  
126    if( glava == NULL)  
        return NULL;  
128  
    /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,  
    pomeramo pokazivac  
130    glava na taj sledeci element. Kada izadjemo iz petlje,  
    glava ce pokazivati na element liste koji nema sledeceg,  
132    tj. poslednji element liste je. Zato vracamo vrednost  
    pokazivaca glava.  
  
134    glava je argument funkcije i njegove promene nece se odraziti  
    na  
    vrednost pokazivaca glava u pozivajucoj funkciji. */  
136    while (glava->sledeci != NULL)  
        glava = glava->sledeci;  
138  
    return glava;  
140 }  
  
142 /* Funkcija trazi u listi element cija je vrednost jednaka datom  
    broju.  
144    Vraca pokazivac na cvor liste u kome je sadržan traženi broj  
    ili NULL u slucaju da takav element ne postoji u listi. */  
146 Cvor* pretrazi_listu(Cvor * glava, int broj) {  
    for ( ; glava != NULL; glava = glava->sledeci)  
148        /* Pronasli smo*/  
        if (glava->vrednost == broj)  
150            return glava;  
  
152    /* Nema traženog broja u listi i vracamo NULL*/  
    return NULL;  
154 }
```

```

156
158
160 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
    Funkcija azurira pokazivac na glavu liste (koji može biti
    promenjen u slučaju da se obriše stara glava) */
162 void obrisi_element(Cvor ** adresa_glave, int broj) {
    Cvor *tekuci = NULL;
164     Cvor *pomocni = NULL;

166     /* Brisemo sa pocetka liste sve eventualne cvorove
        koji su jednaki datom broju, i azuriramo pokazivac na glavu
        */
168     while (*adresa_glave != NULL && (*adresa_glave)->vrednost ==
    broj) {
        /* Sacuvamo adresu repa liste pre oslobadjanja glave */
170         pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
172         *adresa_glave = pomocni;
    }

174     /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
176     if (*adresa_glave == NULL)
        return;

178     /* Od ovog trenutka se u svakom koraku nalazimo
        na tekucem cvoru koji je razlicit od trazenog
        broja (kao i svi levo od njega). Poredimo
180         vrednost sledeceg cvora (ako postoji) sa trazenim
        brojem i brisemo ga ako je jednak, a prelazimo na
182         sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
        dok ne dodjemo do poslednjeg cvora. */
184     tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL)
186         if (tekuci->sledeci->vrednost == broj) {
            /* tekuci->sledeci treba obrisati,
            zbog toga sacuvamo njegovu adresu u pomocni */
188             pomocni = tekuci->sledeci;
            /* Tekucem preusmerimo pokazivac sledeci
            tako sto preskacemo njegovog trenutnog sledeceg.
            Njegov novi sledeci ce biti sledeci od ovog koga brisemo.
190             */
            tekuci->sledeci = tekuci->sledeci->sledeci;
192             /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
            */
            free(pomocni);
194             } else {
                /* Ne treba brisati sledeceg, prelazimo na sledeci */
                tekuci = tekuci->sledeci;
196             }
198         return;
200     }
202 }
204
206 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    Ne saljemo joj adresu promenljive koja cuva glavu liste, jer

```

```
    ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
    pokazivac
208    na glavu liste iz pozivajuće funkcije. */
void ispisi_listu(Cvor * glava)
210 {
    putchar('[');
212    for ( ; glava != NULL; glava = glava->sledeci)
    {
214        printf("%d", glava->vrednost);
    if( glava->sledeci != NULL )
216        printf(", ");
    }

218    printf("]\n");
220 }

222
224
/* Glavni program u kome testiramo sve funkcije za rad sa listama */
226 int main() {
    Cvor *glava = NULL; /* na pocetku imamo praznu listu */
228    Cvor *trazeni = NULL;
    int broj;

230
    /* Testiramo dodavanje na pocetak*/
232    printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+
D)\n");
    printf("\n\tLista: ");
234    ispisi_listu(glava);

236    while(scanf("%d",&broj)>0)
    {
238        dodaj_na_pocetak_liste(&glava, broj);
        printf("\n\tLista: ");
240        ispisi_listu(glava);
    }

242
    printf("\nUnesite element koji se trazi u listi: ");
244    scanf("%d", &broj);

246    trazeni=pretrazi_listu(glava, broj);
    if(trazeni==NULL)
248        printf("Element NIJE u listi!\n");
    else
250        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

252
    /* brisemo elemente iz liste cije polje vrednost je jednako
    broju procitanom sa ulaza */
254    printf("\nUnesite element koji se brise iz liste: ");
    scanf("%d", &broj);

256
    obrisi_element(&glava, broj);

258
260    printf("Lista nakon brisanja: ");
```

```

262     ispisi_listu(glava);
264     oslobodi_listu(&glava);
266     return 0;
}

```

## Rešenje 4.2

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   /* NAPOMENA:
5   Jednostruko povezana lista je struktura podataka
   koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
7   podatak (odredjenog tipa) i pokazivac na sledeci cvor u
   sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
9   liste (bez glave) je takodje lista, i naziva se rep liste.
   Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
11  baratanja listom mi cuvamo samo pokazivac na glavu liste.
   Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
13  sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
   pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
15  preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
   element u listi nema sledeci element: u tom slucaju se
17  njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
   lista se predstavlja NULL pokazivacem.
19
21  Prednost koriscenja povezanih lista u odnosu na dinamicki
   niz je u tome sto se elementi mogu efikasno umetati i brisati
   sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
23  premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
   ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
25  elementi moraju obradivati redom (iteracijom kroz listu).
27
29  Prilikom promene liste (dodavanje novog elementa, brisanje
   elementa,
   premestanje elemenata, itd.) postoji mogucnost da glava liste
   bude
   promenjena, tj. da to postane neki drugi cvor (sa drugom adresom)
31  .
   U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
   promenu liste obavljamo u posebnoj funkciji onda je potrebno da
   se
33  pozivajucoj funkciji vrati azurirana informacija o adresi glave
   liste.
35
   Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
   pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
   cuva
37  adresu glave i koju, eventualno, treba azurirati.
   Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
   promeni promenljivu pozivajuce funkcije direktno. Npr:
39  funkcija_za_promenu(&pok, ...);

```

```
41 */
43 /* Struktura koja predstavlja cvor liste */
typedef struct cvor {
45     /* Podatak koji cvor sadrzi */
    int vrednost;
47     /* Pokazivac na sledeci cvor liste */
    struct cvor *sledeci;
49 } Cvor;

51

53 /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
    novog cvora inicijalizuje na broj, dok pokazivac na
55     sledeci cvor u novom cvoru postavlja na NULL.
    Funkcija vraća pokazivac na novokreirani cvor ili NULL
57     ako alokacija nije uspesno izvršena. */
Cvor * napravi_cvor(int broj) {
59     Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
    if( novi == NULL )
61         return NULL;

63     /* Inicijalizacija polja u novom cvoru */
    novi->vrednost = broj;
65     novi->sledeci = NULL;
    return novi;
67 }

69

71 /* Funkcija oslobadja dinamičku memoriju zauzetu za elemente liste
73     čiji se početni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave) {
75     Cvor *pomocni = NULL;

77     /* Ako lista nije prazna, onda ima memorije koju treba
    osloboditi */
    while (*adresa_glave != NULL) {
79         /* Potrebno je najpre zapamtiti adresu sledećeg elementa,
        a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
81         free(*adresa_glave);
        /* Sledeći element je nova glava liste */
83         *adresa_glave = pomocni;
85     }
    }

87

89 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
91     alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
    za listu čiji početni cvor se nalazi na adresi adresa_glave. */
93 void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
```



```

105     /* Ukoliko je novi NULL */
106     if ( novi == NULL ) {
107         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
108     }
109     /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo
110     program */
111
112     oslobodi_listu(adresa_glave);
113     exit(EXIT_FAILURE);
114 }
115
116 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
117    ili NULL ukoliko je lista prazna */
118 Cvor* pronadji_poslednji (Cvor* glava) {
119     /* ako je lista prazna, nema ni poslednjeg cvor
120     i u tom slucaju vracamo NULL.*/
121     if( glava == NULL)
122         return NULL;
123
124     /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
125     pomeramo pokazivac
126     glava na taj sledeci element. Kada izadjemo iz petlje,
127     glava ce pokazivati na element liste
128     koji nema sledeceg, tj, poslednji element liste je. Zato vracamo
129     vrednost pokazivaca glava.
130
131     glava je argument funkcije i njegove promene nece se odraziti na
132     vrednost pokazivaca glava u pozivajucoj funkciji. */
133     while (glava->sledeci != NULL)
134         glava = glava->sledeci;
135
136     return glava;
137 }
138
139 /* Funkcija dodaje novi cvor na kraj liste. */
140 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj) {
141     Cvor *novi = napravi_cvor(broj);
142     /* Proveravamo da li je doslo do greske prilikom alokacije
143     memorije */
144     prover_i_alokaciju(adresa_glave, novi);
145
146     /* U slucaju prazne liste */
147     if (*adresa_glave == NULL) {
148         /* Glava nove liste je upravo novi cvor i ujedno i cela
149         lista.
150         * Azuriramo vrednost na koju pokazuje adresa_glave i tako
151         azuriramo
152         * i pokazivacku promenljivu u pozivajucoj funkciji. */
153         *adresa_glave = novi;
154     }
155     /* Vracamo se iz funkcije */
156     return;
157 }

```

```
145      /* Ako lista nije prazna, pronalazimo poslednji element liste */
146      Cvor* poslednji = pronadji_poslednji(*adresa_glave);
147
148      /* Dodajemo novi cvor na kraj preusmeravanjem pokazivaca */
149      poslednji->sledeci = novi;
150  }
151
152
153  /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
154  Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
155  ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
156  pokazivac
157  na glavu liste iz pozivajuće funkcije. */
158  void ispisi_listu(Cvor * glava)
159  {
160      putchar('[');
161      for ( ; glava != NULL; glava = glava->sledeci)
162      {
163          printf("%d", glava->vrednost);
164          if( glava->sledeci != NULL )
165              printf(", ");
166      }
167
168      printf("]\n");
169  }
170
171
172
173
174
175  /* Glavni program u kome testiramo sve funkcije za rad sa listama */
176  int main() {
177      Cvor *glava = NULL; /* na pocetku imamo praznu listu */
178      int broj;
179
180      /* Testiramo dodavanje na kraj liste */
181      printf("\nUnesite elemente liste! (za kraj unesite EOF tj. CTRL+
182      D)\n");
183      printf("\n\tLista: ");
184      ispisi_listu(glava);
185
186      while(scanf("%d",&broj)>0) {
187          dodaj_na_kraj_liste(&glava, broj);
188          printf("\n\tLista: ");
189          ispisi_listu(glava);
190      }
191
192      oslobodi_listu(&glava);
193
194      return 0;
195  }
```

### Rešenje 4.3

```

1  #include <stdio.h>
2  #include <stdlib.h>

4  /* NAPOMENA:
   Jednostruko povezana lista je struktura podataka
6  koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
   podatak (odredjenog tipa) i pokazivac na sledeci cvor u
8  sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
   liste (bez glave) je takodje lista, i naziva se rep liste.
10  Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
   baratanja listom mi cuvamo samo pokazivac na glavu liste.
12  Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
   sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
14  pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
   preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
16  element u listi nema sledeci element: u tom slucaju se
   njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
18  lista se predstavlja NULL pokazivacem.

20  Prednost koriscenja povezanih lista u odnosu na dinamicki
   niz je u tome sto se elementi mogu efikasno umetati i brisati
22  sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
   premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
24  ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
   elementi moraju obradivati redom (iteracijom kroz listu).

26

28  Prilikom promene liste (dodavanje novog elementa, brisanje
   elementa,
   premestanje elemenata, itd.) postoji mogucnost da glava liste
   bude
30  promenjena, tj. da to postane neki drugi cvor (sa drugom adresom)
   .
   U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
32  promenu liste obavljamo u posebnoj funkciji onda je potrebno da
   se
   pozivajucoj funkciji vrati azurirana informacija o adresi glave
   liste.

34

   Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
36  pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
   cuva
   adresu glave i koju, eventualno, treba azurirati.
38  Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
   promeni promenljivu pozivajuce funkcije direktno. Npr:
40  funkcija_za_promenu(&pok, ...);

42  */

44  /* Struktura koja predstavlja cvor liste */
   typedef struct cvor {
       /* Podatak koji cvor sadrzi */
46       int vrednost;
       /* Pokazivac na sledeci cvor liste */
48       struct cvor *sledeci;
   } Cvor;

```

```
50 |
52 |
53 | /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
54 | novog cvora inicijalizuje na broj, dok pokazivac na
55 | sledeci cvor u novom cvoru postavlja na NULL.
56 | Funkcija vraća pokazivac na novokreirani cvor ili NULL
57 | ako alokacija nije uspesno izvršena. */
58 | Cvor * napravi_cvor(int broj) {
59 |     Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60 |     if( novi == NULL )
61 |         return NULL;
62 |
63 |     /* Inicijalizacija polja u novom cvoru */
64 |     novi->vrednost = broj;
65 |     novi->sledeci = NULL;
66 |     return novi;
67 | }
68 |
69 |
70 |
71 | /* Funkcija oslobadja dinamičku memoriju zauzetu za elemente liste
72 | ciji se pocetni cvor nalazi na adresi adresa_glave. */
73 | void oslobodi_listu(Cvor ** adresa_glave) {
74 |     Cvor *pomocni = NULL;
75 |
76 |     /* Ako lista nije prazna, onda ima memorije koju treba
77 | osloboditi */
78 |     while (*adresa_glave != NULL) {
79 |         /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
80 | a tek onda osloboditi element koji predstavlja glavu liste */
81 |         pomocni = (*adresa_glave)->sledeci;
82 |         free(*adresa_glave);
83 |         /* Sledeci element je nova glava liste */
84 |         *adresa_glave = pomocni;
85 |     }
86 | }
87 |
88 |
89 |
90 | /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
91 | ukoliko
92 | alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
93 | memorija
94 | za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
95 | void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
96 |     /* Ukoliko je novi NULL */
97 |     if ( novi == NULL ) {
98 |         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
99 |         /* Oslobadjamo svu dinamički alociranu memoriju i prekidamo
100 | program */
101 |
102 |         oslobodi_listu(adresa_glave);
103 |         exit(EXIT_FAILURE);
104 |     }
105 | }
```

```

102 }
104
106 /* Funkcija dodaje novi cvor na pocetak liste.
   Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga
   na pocetak */
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
110     /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
       alokaciji */
112     Cvor *novi = napravi_cvor(broj);
114     prover_i_alokaciju(adresa_glave, novi);
116     /* Uvezujemo novi cvor na pocetak */
118     novi->sledeci = *adresa_glave;
120     /* Nov cvor je sada nova glava liste */
122     *adresa_glave = novi;
124 }
126
128 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov element sa vrednoscu broj.*/
130 Cvor * pronadji_mesto_umetanja(Cvor* glava, int broj ) {
132     /*Ako je lista prazna onda nema takvog mesta i vracamo NULL */
134     if(glava == NULL)
136         return NULL;
138
140     /* Krecemo se kroz listu sve dok se ne dodje do elementa
       ciji je sledeci element veci ili jednak od novog elementa,
       ili dok se ne dodje do poslednjeg elementa.
       Zbog lenjog izracunavanja izraza u C-u prvi deo konjukcije
       mora biti provera da li smo dosli do poslednjeg elementa liste
       pre nego sto proverimo vrednost njegovog sledeceg elementa,
       jer u slucaju poslednjeg, sledeci ne postoji, pa ni vrednost.*/
142     while (glava->sledeci != NULL && glava->sledeci->vrednost <
       broj)
144         glava = glava->sledeci;
146
148     /* Iz petlje smo mogli izaci jer smo dosli do poslednjeg
       elementa
       ili smo se zaustavili ranije na elementu ciji sledeci ima
       vrednost vecu od broj */
150     return glava;
152 }
154
156 void dodaj_iza(Cvor* tekuci, Cvor* novi) {
158     /* Novi element dodajemo iza tekuceg elementa */
160     novi->sledeci = tekuci->sledeci;
162     tekuci->sledeci = novi;
164 }
166
168 /* Funkcija dodaje novi element u sortiranu listu
   tako da nova lista ostane sortirana.*/
170 void dodaj_sortirano(Cvor ** adresa_glave, int broj) {

```

```

154  /* U slucaju prazne liste glava nove liste je upravo novi cvor
    */
156  if ( *adresa_glave == NULL ) {
    Cvor *novi = napravi_cvor(broj);
    /* Proveravamo da li je doslo do greske prilikom alokacije
    memorije */
158  proveri_alokaciju(adresa_glave, novi);
    *adresa_glave = novi;
160  return;
    }

162
    /* Lista nije prazna*/
164  /* Ako je broj manji ili jednak vrednosti u glavi liste,
    onda ga dodajemo na pocetak liste */
166  if ( (*adresa_glave)->vrednost >= broj ) {
    dodaj_na_pocetak_liste(adresa_glave, broj);
168  return;
    }

170
    /* U slucaju da je glava liste element manji od novog elementa,
    tada pronalazimo element liste iza koga treba da se umetne
    nov broj */
172  Cvor* pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);

174
    Cvor *novi = napravi_cvor(broj);
176  /* Proveravamo da li je doslo do greske prilikom alokacije
    memorije */
    proveri_alokaciju(adresa_glave, novi);
178
    /* Uvezujemo novi cvor iza pomocnog */
180  dodaj_iza(pomocni, novi);
    }

182
184
    /* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju.
    Funkcija se u pretrazi oslanja na cinjenicu da je lista
    koja se pretrazuje rastuce sortirana.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj
    ili NULL u slucaju da takav element ne postoji u listi. */
190  Cvor* pretrazi_listu(Cvor * glava, int broj) {
    /* U konjukciji koja cini uslov ostanka u petlji, bitan je
    redosled! */
192  for ( ; glava != NULL && glava->vrednost <= broj ; glava = glava
    ->sledeci)
        /* Pronasli smo*/
194  if (glava->vrednost == broj)
    return glava;

196
    /* Nema traženog broja u listi i vracamo NULL*/
198  return NULL;
    }

200
202

```

```

204  /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
    rastuce.
    Funkcija azurira pokazivac na glavu liste, koji moze biti
206  promenjen u slucaju da se obrise stara glava liste. */
void obrisi_element(Cvor ** adresa_glave, int broj) {
208      Cvor *tekuci = NULL ;
      Cvor *pomocni = NULL ;
210
      /* Brisemo sa pocetka liste sve eventualne cvorove
212      koji su jednaki datom broju, i azuriramo pokazivac na glavu
      */
      while (*adresa_glave != NULL && (*adresa_glave)->vrednost ==
214      broj) {
          /* Sacuvamo adresu repa liste pre oslobadjanja glave */
          pomocni = (*adresa_glave)->sledeci;
216          free(*adresa_glave);
          *adresa_glave = pomocni;
218      }

220      /* Ako je nakon toga lista ostala prazna ili glava liste sadrzi
      vrednost
      koja je veca od broja, kako je lista sortirana rastuce nema
      potrebe
222      broj traziti u repu liste i zato prekidamo funkciju */
      if ( *adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
224          return;

226      /* Od ovog trenutka se u svakom koraku nalazimo u tekucem cvoru
      cija
      vrednost je manja od trazenog broja (kao i svi levo od njega)
      .
228      Poredimo vrednost sledeceg cvora (ako postoji) sa trazenim
      brojem
      i brisemo ga ako je jednak, a prelazimo na sledeci cvor ako
      je
230      razlicit. Ovaj postupak ponavljamo dok ne dodjemo do
      poslednjeg cvora
      ili prvog cvora cija vrednost je veca od trazenog broja. */
232      tekuci = *adresa_glave;
      while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <=
      broj)
234          if (tekuci->sledeci->vrednost == broj) {
              /* tekuci->sledeci treba obrisati,
236              zbog toga sacuvamo njegovu adresu u pomocni */
              pomocni = tekuci->sledeci;
238              /* Tekucem preusmerimo pokazivac sledeci
              tako sto preskacemo njegovog trenutnog sledeceg.
              Njegov novi sledeci ce biti sledeci od ovog koga brisemo.
240          */
              tekuci->sledeci = tekuci->sledeci->sledeci;
242          /* Sada mozemo
              * slobodno i da oslobodimo cvor sa vrednoscu broj */
              free(pomocni);
244          } else {
246          /* Ne treba brisati sledeceg, jer je manji od trazenog

```

```

    i prelazimo na sledeci */
248     tekuci = tekuci->sledeci;
    }
250     return;
}
252
/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
254   Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
   ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
   pokazivac
256   na glavu liste iz pozivajuće funkcije. */
void ispisi_listu(Cvor * glava)
258 {
    putchar('[');
260     for ( ; glava != NULL; glava = glava->sledeci)
    {
262         printf("%d", glava->vrednost);
        if( glava->sledeci != NULL )
264             printf(", ");
    }

266     printf("]\n");
268 }

270
272
274 /* Glavni program u kome testiramo sve funkcije za rad sa listama */
int main() {
276     Cvor *glava = NULL; /* na pocetku imamo praznu listu */
    Cvor *trazeni = NULL;
278     int broj;

280     /* Testiramo dodavanje u listu tako da ona bude rastuce
   sortirana */
    printf("\nUnosite elemente liste! (za kraj unesite EOF tj. CTRL+
282     D)\n");
    printf("\n\tLista: ");
    ispisi_listu(glava);
284
    while(scanf("%d",&broj)>0)
286     {
        dodaj_sortirano(&glava, broj);
288         printf("\n\tLista: ");
        ispisi_listu(glava);
290     }

292     printf("\nUnesite element koji se trazi u listi: ");
    scanf("%d", &broj);
294
    trazeni = pretrazi_listu(glava, broj);
296     if(trazeni == NULL)
        printf("Element NIJE u listi!\n");
298     else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
}
```



```

300
302     /* Brisemo elemente iz liste cije polje vrednost je jednako
    broju
        procitanom sa ulaza */
304     printf("\nUnesite element koji se brise iz liste: ");
    scanf("%d", &broj);
306
    obrisi_element(&glava, broj);
308
    printf("Lista nakon brisanja: ");
310    ispisi_listu(glava);
312
    oslobodi_listu(&glava);
314
    return 0;
}

```

Rešenje 4.4

Rešenje 4.5

Rešenje 4.6

Rešenje 4.7

Rešenje 4.8

Rešenje 4.9

```

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

4  /* NAPOMENA:
    Jednostruko povezana lista je struktura podataka
6   koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
    podatak (odredjenog tipa) i pokazivac na sledeci cvor u
8   sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
    liste (bez glave) je takodje lista, i naziva se rep liste.
10  Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
    baratanja listom mi cuvamo samo pokazivac na glavu liste.
12  Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
    sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
14  pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
    preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
16  element u listi nema sledeci element: u tom slucaju se
    njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
18  lista se predstavlja NULL pokazivacem.

20  Prednost koriscenja povezanih lista u odnosu na dinamicki
    niz je u tome sto se elementi mogu efikasno umetati i brisati
22  sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili

```

```
24 premestanjem elemenata. Nedostatak ovakvog pristupa je to što
26 ne možemo nasumično pristupiti proizvoljnom elementu, već se
    elementi moraju obradivati redom (iteracijom kroz listu).

28 Prilikom promene liste (dodavanje novog elementa, brisanje
    elementa,
    premestanje elemenata, itd.) postoji mogućnost da glava liste
    bude
30 promenjena, tj. da to postane neki drugi cvor (sa drugom adresom)
    .
    U tom slučaju se pokazivac na glavu liste mora azurirati. Kada
32 promenu liste obavljamo u posebnoj funkciji onda je potrebno da
    se
    pozivajućoj funkciji vrati azurirana informacija o adresi glave
    liste.

34 Pozvana funkcija koja vrši promenu na listi prihvata kao argument
36 pokazivac na pokazivačku promenljivu koja u pozivajućoj funkciji
    čuva
    adresu glave i koju, eventualno, treba azurirati.
38 Sada pozvana funkcija može interno da preko dobijenog pokazivaca
    promeniti promenljivu pozivajuće funkcije direktno. Npr:
40         funkcija_za_promenu(&pok, ...);
42 */

44 /* Struktura koja predstavlja cvor liste */
46 typedef struct cvor {
48     /* Podatak koji cvor sadrži */
49     int vrednost;
50     /* Pokazivac na sledeći cvor liste */
51     struct cvor *sledeci;
52 } Cvor;

54 /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
    novog cvora inicijalizuje na broj, dok pokazivac na
    sledeći cvor u novom cvoru postavlja na NULL.
56 Funkcija vraća pokazivac na novokreirani cvor ili NULL
    ako alokacija nije uspešno izvršena. */
58 Cvor * napravi_cvor(int broj) {
59     Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60     if( novi == NULL )
61         return NULL;

62     /* Inicijalizacija polja u novom cvoru */
63     novi->vrednost = broj;
64     novi->sledeci = NULL;
65     return novi;
66 }

68

70

72 /* Funkcija oslobadja dinamičku memoriju zauzetu za elemente liste
```

```

    ciji se pocetni cvor nalazi na adresi adresa_glave. */
74 void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;
76
    /* Ako lista nije prazna, onda ima memorije koju treba
    osloboditi */
78    while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
        a tek onda osloboditi element koji predstavlja glavu liste */
80        pomocni = (*adresa_glave)->sledeci;
82        free(*adresa_glave);
        /* Sledeci element je nova glava liste */
84        *adresa_glave = pomocni;
    }
86 }
88
90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
    alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
    za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
92 void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94     /* Ukoliko je novi NULL */
96     if ( novi == NULL ) {
        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo
        program */
98
        oslobodi_listu(adresa_glave);
100        exit(EXIT_FAILURE);
    }
102 }
104
106 /* Funkcija dodaje novi cvor na pocetak liste.
    Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga
    na pocetak */
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
    /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
    alokaciji */
110    Cvor *novi = napravi_cvor(broj);
    prover_i_alokaciju(adresa_glave, novi);
112
    /* Uvezujemo novi cvor na pocetak */
114    novi->sledeci = *adresa_glave;
    /* Nov cvor je sada nova glava liste */
116    *adresa_glave = novi;
    }
118
120
122 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL ukoliko je lista prazna */

```

```

124 Cvor* pronadji_poslednji (Cvor* glava) {
125     /* ako je lista prazna, nema ni poslednjeg cvor
126     i u tom slucaju vracamo NULL.*/
127     if( glava == NULL)
128         return NULL;

129     /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
130     pomeramo pokazivac
131     glava na taj sledeci element. Kada izadjemo iz petlje,
132     glava ce pokazivati na element liste koji nema sledeceg,
133     tj. poslednji element liste je. Zato vracamo vrednost
134     pokazivaca glava.

135     glava je argument funkcije i njegove promene nece se odraziti
136     na
137     vrednost pokazivaca glava u pozivajucoj funkciji. */
138     while (glava->sledeci != NULL)
139         glava = glava->sledeci;

140     return glava;
141 }

142
143 /* Funkcija trazi u listi element cija je vrednost jednaka datom
144    broju.
145    Vraca pokazivac na cvor liste u kome je sadržan traženi broj
146    ili NULL u slučaju da takav element ne postoji u listi. */
147 Cvor* pretrazi_listu(Cvor * glava, int broj) {
148     for ( ; glava != NULL; glava = glava->sledeci)
149         /* Pronasli smo*/
150         if (glava->vrednost == broj)
151             return glava;

152     /* Nema traženog broja u listi i vracamo NULL*/
153     return NULL;
154 }

155
156
157
158 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
159 Funkcija azurira pokazivac na glavu liste (koji može biti
160 promenjen u slučaju da se obriše stara glava) */
161 void obrisi_element(Cvor ** adresa_glave, int broj) {
162     Cvor *tekuci = NULL;
163     Cvor *pomocni = NULL;

164
165     /* Brisemo sa pocetka liste sve eventualne cvorove
166     koji su jednaki datom broju, i azuriramo pokazivac na glavu
167     */
168     while (*adresa_glave != NULL && (*adresa_glave)->vrednost ==
169 broj) {
170         /* Sacuvamo adresu repa liste pre oslobadjanja glave */
171         pomocni = (*adresa_glave)->sledeci;
172         free(*adresa_glave);
173         *adresa_glave = pomocni;

```

```

174     }
176     /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
177     if ( *adresa_glave == NULL)
178         return;
179
180     /* Od ovog trenutka se u svakom koraku nalazimo
181        na tekucem cvoru koji je razlicit od trazenog
182        broja (kao i svi levo od njega). Poredimo
183        vrednost sledeceg cvora (ako postoji) sa trazenim
184        brojem i brisemo ga ako je jednak, a prelazimo na
185        sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
186        dok ne dodjemo do poslednjeg cvora. */
187     tekuci = *adresa_glave;
188     while (tekuci->sledeci != NULL)
189         if (tekuci->sledeci->vrednost == broj) {
190             /* tekuci->sledeci treba obrisati,
191                zbog toga sacuvamo njegovu adresu u pomocni */
192             pomocni = tekuci->sledeci;
193             /* Tekucem preusmerimo pokazivac sledeci
194                tako sto preskacemo njegovog trenutnog sledeceg.
195                Njegov novi sledeci ce biti sledeci od ovog koga brisemo.
196             */
197             tekuci->sledeci = tekuci->sledeci->sledeci;
198             /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
199             */
200             free(pomocni);
201         } else {
202             /* Ne treba brisati sledeceg, prelazimo na sledeci */
203             tekuci = tekuci->sledeci;
204         }
205     return;
206 }
207
208 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
209    Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
210    ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
211    pokazivac
212    na glavu liste iz pozivajuce funkcije. */
213 void ispisi_listu(Cvor * glava)
214 {
215     putchar('[');
216     for ( ; glava != NULL; glava = glava->sledeci)
217     {
218         printf("%d", glava->vrednost);
219         if( glava->sledeci != NULL )
220             printf(", ");
221     }
222     printf("]\n");
223 }
224
225 /* Glavni program u kome testiramo sve funkcije za rad sa listama */

```

```
226 int main() {
    Cvor *glava = NULL; /* na pocetku imamo praznu listu */
228 Cvor *trazeni = NULL;
    int broj;

230
    /* Testiramo dodavanje na pocetak*/
232 printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+
D)\n");
    printf("\n\tLista: ");
234 ispisi_listu(glava);

    while(scanf("%d",&broj)>0)
    {
238         dodaj_na_pocetak_liste(&glava, broj);
        printf("\n\tLista: ");
240         ispisi_listu(glava);
    }

242
    printf("\nUnesite element koji se trazi u listi: ");
244 scanf("%d", &broj);

    trazeni=pretrazi_listu(glava, broj);
    if(trazeni==NULL)
248         printf("Element NIJE u listi!\n");
    else
250         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

252
    /* brisemo elemente iz liste cije polje vrednost je jednako
254        broju procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
256 scanf("%d", &broj);

    obrisi_element(&glava, broj);

258
    printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

260
    oslobodi_listu(&glava);

262
    return 0;
264
266 }
```

Rešenje 4.10

Rešenje 4.11

Rešenje 4.12

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

4 /* NAPOMENA:
    Jednostruko povezana lista je struktura podataka
```

```

6   koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
8   podatak (odredjenog tipa) i pokazivac na sledeci cvor u
10  sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
12  liste (bez glave) je takodje lista, i naziva se rep liste.
14  Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
16  baratanja listom mi cuvamo samo pokazivac na glavu liste.
18  Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
20  sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
22  pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
24  preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
26  element u listi nema sledeci element: u tom slucaju se
28  njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
30  lista se predstavlja NULL pokazivacem.

32  Prednost koriscenja povezanih lista u odnosu na dinamicki
34  niz je u tome sto se elementi mogu efikasno umetati i brisati
36  sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
38  premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
40  ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
42  elementi moraju obradivati redom (iteracijom kroz listu).

44  Prilikom promene liste (dodavanje novog elementa, brisanje
46  elementa,
48  premestanje elemenata, itd.) postoji mogucnost da glava liste
50  bude
52  promenjena, tj. da to postane neki drugi cvor (sa drugom adresom)
54  .
56  U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
58  promenu liste obavljamo u posebnoj funkciji onda je potrebno da
60  se
62  pozivajucoj funkciji vrati azurirana informacija o adresi glave
64  liste.

66  Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
68  pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
70  cuva
72  adresu glave i koju, eventualno, treba azurirati.
74  Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
76  promeni promenljivu pozivajuce funkcije direktno. Npr:
78      funkcija_za_promenu(&pok, ...);
80  */

82  /* Struktura koja predstavlja cvor liste */
84  typedef struct cvor {
86      /* Podatak koji cvor sadrzi */
88      int vrednost;
90      /* Pokazivac na sledeci cvor liste */
92      struct cvor *sledeci;
94  } Cvor;

96  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
98  novog cvora inicijalizuje na broj, dok pokazivac na
100  sledeci cvor u novom cvoru postavlja na NULL.

```

```
56     Funkcija vraća pokazivac na novokreirani cvor ili NULL
    ako alokacija nije uspesno izvršena. */
58 Cvor * napravi_cvor(int broj) {
    Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60     if( novi == NULL )
        return NULL;

62     /* Inicijalizacija polja u novom cvoru */
64     novi->vrednost = broj;
    novi->sledeci = NULL;
66     return novi;
}

68
70
72 /* Funkcija oslobadja dinamičku memoriju zauzetu za elemente liste
    čiji se početni cvor nalazi na adresi adresa_glave. */
74 void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;

76     /* Ako lista nije prazna, onda ima memorije koju treba
    osloboditi */
78     while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledećeg elementa,
80         a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
82         free(*adresa_glave);
        /* Sledeći element je nova glava liste */
84         *adresa_glave = pomocni;
    }
86 }

88
90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
    alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
92     za listu čiji početni cvor se nalazi na adresi adresa_glave. */
void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94     /* Ukoliko je novi NULL */
    if ( novi == NULL ) {
96         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        /* Oslobadjamo svu dinamički alociranu memoriju i prekidamo
        program */
98         oslobodi_listu(adresa_glave);
100        exit(EXIT_FAILURE);
    }
102 }

104
106 /* Funkcija dodaje novi cvor na pocetak liste.
    Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga
```



```

108     na pocetak */
void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
    /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
    alokaciji */
110    Cvor *novi = napravi_cvor(broj);
    prover_i_alokaciju(adresa_glave, novi);
112
    /* Uvezujemo novi cvor na pocetak */
114    novi->sledeci = *adresa_glave;
    /* Nov cvor je sada nova glava liste */
116    *adresa_glave = novi;
}
118
120
/* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
122 ili NULL ukoliko je lista prazna */
Cvor* pronadji_poslednji (Cvor* glava) {
124    /* ako je lista prazna, nema ni poslednjeg cvor
    i u tom slucaju vracamo NULL.*/
126    if( glava == NULL)
        return NULL;
128
    /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
    pomeramo pokazivac
130    glava na taj sledeci element. Kada izadjemo iz petlje,
    glava ce pokazivati na element liste koji nema sledeceg,
132    tj. poslednji element liste je. Zato vracamo vrednost
    pokazivaca glava.

    glava je argument funkcije i njegove promene nece se odraziti
    na
    vrednost pokazivaca glava u pozivajucoj funkciji. */
136    while (glava->sledeci != NULL)
        glava = glava->sledeci;
138
    return glava;
140 }
142
/* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju.
144    Vraca pokazivac na cvor liste u kome je sadržan traženi broj
    ili NULL u slucaju da takav element ne postoji u listi. */
146 Cvor* pretrazi_listu(Cvor * glava, int broj) {
    for ( ; glava != NULL; glava = glava->sledeci)
148        /* Pronasli smo */
        if (glava->vrednost == broj)
150            return glava;

    /* Nema traženog broja u listi i vracamo NULL*/
152    return NULL;
154 }
156

```

```
158
160 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
161 Funkcija azurira pokazivac na glavu liste (koji moze biti
162 promenjen u slucaju da se obrise stara glava) */
163 void obrisi_element(Cvor ** adresa_glave, int broj) {
164     Cvor *tekuci = NULL;
165     Cvor *pomocni = NULL;
166
167     /* Brisemo sa pocetka liste sve eventualne cvorove
168     koji su jednaki datom broju, i azuriramo pokazivac na glavu
169     */
170     while (*adresa_glave != NULL && (*adresa_glave)->vrednost ==
171 broj) {
172         /* Sacuvamo adresu repa liste pre oslobadjanja glave */
173         pomocni = (*adresa_glave)->sledeci;
174         free(*adresa_glave);
175         *adresa_glave = pomocni;
176     }
177
178     /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
179     if (*adresa_glave == NULL)
180         return;
181
182     /* Od ovog trenutka se u svakom koraku nalazimo
183     na tekucem cvoru koji je razlicit od trazienog
184     broja (kao i svi levo od njega). Poredimo
185     vrednost sledeceg cvora (ako postoji) sa trazienim
186     brojem i brisemo ga ako je jednak, a prelazimo na
187     sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
188     dok ne dodjemo do poslednjeg cvora. */
189     tekuci = *adresa_glave;
190     while (tekuci->sledeci != NULL)
191         if (tekuci->sledeci->vrednost == broj) {
192             /* tekuci->sledeci treba obrisati,
193             zbog toga sacuvamo njegovu adresu u pomocni */
194             pomocni = tekuci->sledeci;
195             /* Tekucem preusmerimo pokazivac sledeci
196             tako sto preskakemo njegovog trenutnog sledeceg.
197             Njegov novi sledeci ce biti sledeci od ovog koga brisemo.
198             */
199             tekuci->sledeci = tekuci->sledeci->sledeci;
200             /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
201             */
202             free(pomocni);
203         } else {
204             /* Ne treba brisati sledeceg, prelazimo na sledeci */
205             tekuci = tekuci->sledeci;
206         }
207     return;
208 }
209
210 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
211 Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
212 ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
213 pokazivac
214 na glavu liste iz pozivajuce funkcije. */
```

```

210 void ispisi_listu(Cvor * glava)
{
    putchar('[');
212     for ( ; glava != NULL; glava = glava->sledeci)
    {
214         printf("%d", glava->vrednost);
        if( glava->sledeci != NULL )
216             printf(", ");
    }

218     printf("]\n");
220 }

222
224
/* Glavni program u kome testiramo sve funkcije za rad sa listama */
226 int main() {
    Cvor *glava = NULL; /* na pocetku imamo praznu listu */
228     Cvor *trazeni = NULL;
    int broj;

230
    /* Testiramo dodavanje na pocetak*/
232     printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+
D)\n");
    printf("\n\tLista: ");
234     ispisi_listu(glava);

    while(scanf("%d",&broj)>0)
    {
238         dodaj_na_pocetak_liste(&glava, broj);
        printf("\n\tLista: ");
240         ispisi_listu(glava);
    }

242
    printf("\nUnesite element koji se trazi u listi: ");
244     scanf("%d", &broj);

    trazeni=pretrazi_listu(glava, broj);
    if(trazeni==NULL)
248         printf("Element NIJE u listi!\n");
    else
250         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

252
    /* brisemo elemente iz liste cije polje vrednost je jednako
    broju procitanom sa ulaza */
254     printf("\nUnesite element koji se brise iz liste: ");
    scanf("%d", &broj);
256

    obrisi_element(&glava, broj);
258

    printf("Lista nakon brisanja: ");
    ispisi_listu(glava);
260

    oslobodi_listu(&glava);
262

```

```
264     return 0;
266 }
```

Rešenje 4.13

Rešenje 4.14

Rešenje 4.15

Rešenje 4.16

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX 50
7
8  /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
9  pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
11     char *rec;
12     int brojac;
13     struct cvor *levo;
14     struct cvor *desno;
15 } Cvor;
16
17 /* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec) {
19
20     /* Alociramo memoriju za novi cvor */
21     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
22     if (novi_cvor == NULL)
23         return NULL;
24
25     /* Alociramo memoriju za zadatu rec: potrebno je rezervisati
26 memoriju za svaki karakter reci ukljucujuci i terminirajucu nulu */
27     novi_cvor->rec = (char*)malloc((strlen(rec)+1)*sizeof(char));
28     if (novi_cvor->rec == NULL) {
29         free(novi_cvor);
30         return NULL;
31     }
32
33     /* Inicijalizacija polja u novom cvoru */
34     strcpy(novi_cvor->rec, rec);
35     novi_cvor->brojac = 1;
36     novi_cvor->levo = NULL;
37     novi_cvor->desno = NULL;
38
39     /* Vracamo adresu novog cvora */
40     return novi_cvor;
41 }
```

```
43 /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla*/
void prover_i_alokaciju(Cvor* novi_cvor) {
45
46     /* Ukoliko je cvor neuspesno kreiran */
47     if( novi_cvor == NULL) {
48
49         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
programa */
51         fprintf(stderr, "Malloc greska za nov cvor!\n");
52         exit(EXIT_FAILURE);
53     }
54 }
55
56 /* Funkcija koja dodaje novu rec u stablo. */
57 void dodaj_u_stablo(Cvor** adresa_korena, char *rec) {
58
59     /* Ako je stablo prazno */
60     if( *adresa_korena == NULL) {
61         /* Kreiramo novi cvor */
62         Cvor* novi = napravi_cvor(rec);
63         prover_i_alokaciju(novi);
64         /* i proglašavamo ga korenom stabla */
65         *adresa_korena = novi;
66         return;
67     }
68
69     /* U suprotnom trazimo odgovarajucu poziciju za novu rec */
70
71     /* Ako je rec leksikografski manju od reci u korenu ubacujemo
je u levo podstablo */
73     if (strcmp(rec, (*adresa_korena)->rec) < 0)
74         dodaj_u_stablo(&(*adresa_korena)->levo, rec);
75     else
76         /* Ako je rec leksikografski veca od reci u korenu
ubacujemo je u desno podstablo */
77         if (strcmp(rec, (*adresa_korena)->rec) > 0)
78             dodaj_u_stablo(&(*adresa_korena)->desno, rec);
79         else
80             /* Ako je rec jednaka reci u korenu, uvecavamo njen
broj pojavljivanja */
81             (*adresa_korena)->brojac++;
82     }
83 }
84
85
86
87 /* Funkcija koja oslobadja memoriju zauzetu stablom */
88 void oslobodi_stablo(Cvor** adresa_korena) {
89
90     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
91     if(*adresa_korena == NULL)
92         return;
93
94     /* Inace ...*/
95     /* Oslobadjamo memoriju zauzetu levim podstablom */
96     oslobodi_stablo(&(*adresa_korena)->levo);
97 }
```

```
100      /* Oslobadjamo memoriju zauzetu desnim podstablom */
101      oslobodi_stablo(&(*adresa_korena)->desno);
102
103      /* Oslobadjamo memoriju zauzetu korenom */
104      free((*adresa_korena)->rec);
105      free(*adresa_korena);
106
107      /* Proglasavamo stablo praznim */
108      *adresa_korena = NULL;
109  }
110
111  /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
112   sa najvećim brojem pojavljivanja) */
113  Cvor *nadj_i_najfrekventniju_rec(Cvor * koren) {
114
115      Cvor *max, *max_levo, *max_desno;
116
117      /* Ako je stablo prazno, prekidamo sa pretragom */
118      if (koren == NULL)
119          return NULL;
120
121      /* Pronalazimo najfrekventniju reci u levom podstablu */
122      max_levo = nadj_i_najfrekventniju_rec(koren->levo);
123
124      /* Pronalazimo najfrekventniju reci u desnom podstablu */
125      max_desno = nadj_i_najfrekventniju_rec(koren->desno);
126
127      /* Trazimo maksimum vrednosti pojavljivanja reci iz
128      levog podstabla, korena i desnog podstabla */
129      max = koren;
130      if (max_levo != NULL && max_levo->brojac > max->brojac)
131          max = max_levo;
132      if (max_desno != NULL && max_desno->brojac > max->brojac)
133          max = max_desno;
134
135      /* Vracamo adresu cvora sa najvećim brojacem */
136      return max;
137  }
138
139  /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
140  pracene brojem pojavljivanja */
141  void prikazi_stablo(Cvor * koren) {
142
143      /* Ako je stablo prazno, završavamo sa ispisom */
144      if (koren == NULL)
145          return;
146
147      /* Zbog leksikografskog poretka, prvo ispisujemo sve reci iz
148      levog podstabla */
149      prikazi_stablo(koren->levo);
150      /* Zatim ispisujemo rec iz korena */
151      printf("%s: %d\n", koren->rec, koren->brojac);
152      /* I nastavljamo sa ispisom reci iz desnog podstabla */
153      prikazi_stablo(koren->desno);
154  }
```

```

    prikazi_stablo(koren->desno);
155 }

157

159 /* Funkcija ucitava sledecu rec iz zadate datoteke i upisuje je
    u niz rec. Maksimalna duzina reci je odredjena argumentom max.
161 Funkcija vraca EOF ako nema vise reci ili 0 u suprotnom.
    Rec je niz malih ili velikih slova.*/
163 int procitaj_rec(FILE * f, char rec[], int max) {
    /* karakter koji citamo */
165     int c;
    /* indeks pozicije na koju se smesta procitani karakter */
167     int i = 0;

169     /* Sve dok ima mesta za jos jedan karakter u nizu
        i dokle god nismo stigli do kraja datoteke... */
171     while (i < max - 1 && (c = fgetc(f)) != EOF) {
        /* Proveravamo da li je procitani karakter slovo */
173         if (isalpha(c))
            /* Ako jeste, smestamo ga u niz - pritom vrsimo
175 konverziju u mala slova jer program treba da bude neosetljiv na
            razliku izmedju malih i velikih slova */
177             rec[i++] = tolower(c);
            else
179                 /* Ako nije, proveravamo da li smo procitali barem jedno
                slovo nove rece */
181                 /* Ako jesmo prekidamo sa citanjem */
                if (i > 0)
183                     break;

185                 /* U suprotnom idemo na sledecu iteraciju */
            }

187     /* Dodajemo na rec terminirajucu nulu */
189     rec[i] = '\0';

191     /* Vracamo 0 ako smo procitali rec, EOF u suprotnom */
    return i > 0 ? 0 : EOF;
193 }

195
197 int main(int argc, char **argv) {
    Cvor *koren = NULL, *max;
    FILE *f;
199     char rec[MAX];

201     /* Proveravamo da li je navedeno ime datoteke prilikom
    pokretanja programa */
203     if (argc < 2) {
        fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
205         exit(EXIT_FAILURE);
    }

207     /* Otvaramo datoteku iz koje citamo reci */
209     if ((f = fopen(argv[1], "r")) == NULL) {

```

```

    fprintf(stderr, "fopen() greska pri otvaranju %s\n", argv
211 [1]);
    exit(EXIT_FAILURE);
213 }

/* Ucitavamo reci iz datoteke i smestamo u binarno stablo
pretrage. */
215 while (procitaj_rec(f, rec, MAX) != EOF)
    dodaj_u_stablo(&koren, rec);
217

/* Posto smo zavrшили sa citanjem reci zatvaramo datoteku */
219 fclose(f);

/* Prikazujemo sve reci iz teksta i brojeve njihovih
pojavlјivanja. */
221 prikazi_stablo(koren);
223

/* Pronalazimo najfrekventniju rec */
225 max = najdi_najfrekventniju_rec(koren);
227

/* Ako takve reci nema... */
229 if (max== NULL)
    /* Ispisujemo odgovarajuće obavestenje */
231 printf("U tekstu nema reci!\n");
    else
233     /* Inace, ispisujemo broj pojavlјivanja reci */
    printf("Najcesca rec: %s (pojavlјuje se %d puta)\n", max->
rec,
235 max->brojac);
237

/* Oslobadjamo dinamički alociran prostor za stablo */
239 oslobodi_stablo(&koren);

/* Završavamo sa programom */
241 return 0;
243 }
```

Rešenje 4.17

Rešenje 4.18

Rešenje 4.19

Rešenje 4.20

Rešenje 4.21

Rešenje 4.22



Rešenje [4.23](#)

Rešenje [4.24](#)

Rešenje [4.25](#)

Rešenje [4.26](#)

Rešenje [4.27](#)

Rešenje [4.28](#)



# Glava 5

## Ispitni rokovi

### 5.1 Programiranje 2, praktični deo ispita, jun 2015.

#### Zadatak 5.1

Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost -1 i prekinuti izvršavanje programa.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>ž Sadraj datoteke: 5 Programiranje Matematika 12345 dInAmiCnArEc Ispit Izlaz: Ispit Matematika Programiranje</pre>	<pre>ž Sadraj datoteke: 2 maksimalano poena Izlaz:</pre>	<pre>Problem: datoteka ne postoji Izlaz: -1</pre>

[Rešenje [5.1](#)]

#### Zadatak 5.2

Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sa-  
drže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja  
izračunava zbir svih čvorova koji se nalaze na  $n$ -tom nivou stabla (koren se nalazi  
na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane  
funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim  
stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj  $n$ , a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati  $-1$ .

*Test 1*

```
|| Ulaz:
|| 2 8 10 3 6 14 13 7 4 0
|| Izlaz:
|| 20
```

*Test 2*

```
|| Ulaz:
|| 0 50 14 5 2 4 56 8 52 7 1 0
|| Izlaz:
|| 50
```

[Rešenje 5.2]

**Zadatak 5.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost  $-1$  na standardni izlaz za greške.

*Test 1*

```
|| Ulaz:
|| 4
|| 5
|| 1 2 3 4 5
|| -1 2 -3 4 -5
|| -5 -4 -3 -2 1
|| -1 0 0 0 0
|| Izlaz:
|| 0
```

*Test 2*

```
|| Ulaz:
|| 2
|| 3
|| 0 0 -5
|| 1 2 -4
|| Izlaz:
```

*Test 3*

```
|| Ulaz:
|| -2
|| Izlaz (na stderr):
|| -1
```

[Rešenje 5.3]

## 5.2 Programiranje 2, praktični deo ispita, jul 2015.

### Zadatak 5.4

Napisati program koji kao prvi argument komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati  $-1$  na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

Test 1

```
|| Poziv: ./a.out fajl.txt test
|| Datoteka: Ovo je test primer.
||           U njemu se rec test
||           javlja
||           vise puta. testtesttest
|| Izlaz: 5
```

Test 2

```
|| Poziv: ./a.out
|| Izlaz (na stderr): -1
```

Test 3

```
|| Poziv: ./a.out fajl.txt foo
|| Datoteka: (ne postoji)
|| Izlaz (na stderr): -1
```

Test 4

```
|| Poziv: ./a.out fajl.txt .
|| Datoteka: (prazna)
|| Izlaz: 0
```

[Rešenje 5.4]

### Zadatak 5.5

Na početku datoteke "trouglovi.txt" nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

Test 1

```
|| Datoteka: 4
||           0 0 0 1.2 1 0
||           0.3 0.3 0.5 0.5 0.9 1
||           -2 0 0 0 0 1
||           2 0 2 2 -1 -1
|| Izlaz:    2 0 2 2 -1 -1
||           -2 0 0 0 0 1
||           0 0 0 1.2 1 0
||           0.3 0.3 0.5 0.5 0.9 1
```

Test 2

```
|| Datoteka: 3
||           1.2 3.2 1.1
||           4.3
|| Izlaz:    -1
```

Test 3

```
|| Datoteka: (nema datoteke)
|| Izlaz:    -1
```

Test 4

```
|| Datoteka: 0
|| Izlaz:
```

[Rešenje 5.5]

**Zadatak 5.6** Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeba. Napisati funkciju

```
int f3(Cvor *koren, int n)
```

koja u datom stablu prebrojava čvorove na  $n$ -tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou 0. Ispravnost napisane funkcije testirati na osnovu zadate main funkcije i biblioteke za rad sa stablima.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Ulaz:</code> 1 5 3 6 1 4 7 9 <code>Izlaz:</code> 1	<code>Ulaz:</code> 2 5 3 6 1 0 4 7 9 <code>Izlaz:</code> 2	<code>Ulaz:</code> 0 4 2 5 <code>Izlaz:</code> 0
<i>Test 4</i>	<i>Test 5</i>	
<code>Ulaz:</code> 3 <code>Izlaz:</code> 0	<code>Ulaz:</code> -1 4 5 1 7 <code>Izlaz:</code> 0	

[Rešenje 5.6]

### 5.3 Rešenja

#### Rešenje 5.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define MAX 50
5
6 void greska(){
7     printf("-1\n");
8     exit(EXIT_FAILURE);
9 }
10
11 int main(int argc, char* argv[]){
12
13     FILE* ulaz;
14     char** linije;
15     int i, j, n;
16
17     /* Proveravamo argumente komandne linije.
18     */
19     if(argc!=2){
20         greska();
21     }
22
23     /* Otvaramo datoteku cije ime je navedeno kao argument komandne
24     linije neposredno nakon imena programa koji se poziva. */
25     ulaz=fopen(argv[1], "r");
26     if(ulaz==NULL){
27         greska();
28     }
29
30     /* Ucitavamo broj linija. */
31     fscanf(ulaz, "%d", &n);
32
33     /* Alociramo memoriju na osnovu ucitanog broja linija.*/
```

```

33     linije=(char**)malloc(n*sizeof(char*));
34     if(linije==NULL){
35         greska();
36     }
37     for(i=0; i<n; i++){
38         linije[i]=malloc(MAX*sizeof(char));
39         if(linije[i]==NULL){
40             for(j=0; j<i; j++){
41                 free(linije[j]);
42             }
43             free(linije);
44             greska();
45         }
46     }
47
48     /* Ucitavamo svih n linija iz datoteke. */
49     for(i=0; i<n; i++){
50         fscanf(ulaz, "%s", linije[i]);
51     }
52
53     /* Ispisujemo u odgovarajucem poretku ucitane linije koje
54     zadovoljavaju kriterijum. */
55     for(i=n-1; i>=0; i--){
56         if(isupper(linije[i][0])){
57             printf("%s\n", linije[i]);
58         }
59     }
60
61     /* Oslobadjamo memoriju koju smo dinamicki alocirali. */
62     for(i=0; i<n; i++){
63         free(linije[i]);
64     }
65
66     free(linije);
67
68     /* Zatvaramo datoteku. */
69     fclose(ulaz);
70
71     /* Završavamo sa programom. */
72     return 0;
73 }

```

## Rešenje 5.2

```

1  #include <stdio.h>
2  #include "stabla.h"
3
4
5  int sumirajN (Cvor * koren, int n){
6      if(koren==NULL){
7          return 0;
8      }
9
10     if(n==0){

```

```

12         return koren->broj;
13     }
14     return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
15 }
16
17
18 int main(){
19     Cvor* koren=NULL;
20     int n;
21     int nivo;
22
23     /* Citamo vrednost nivoa */
24     scanf("%d", &nivo);
25
26
27     while(1){
28
29         /* Citamo broj sa standardnog ulaza */
30         scanf("%d", &n);
31
32         /* Ukoliko je korisnik uneo 0, prekidamo dalje citanje. */
33         if(n==0){
34             break;
35         }
36
37         /* A ako nije, dodajemo procitani broj u stablo. */
38         dodaj_u_stablo(&koren, n);
39
40     }
41
42     /* Ispisujemo rezultat rada trazene funkcije */
43     printf("%d\n", sumirajN(koren,nivo));
44
45     /* Oslobadjamo memoriju */
46     oslobodi_stablo(&koren);
47
48
49     /* Prekidamo izvršavanje programa */
50     return 0;
51 }

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"
4
5  Cvor* napravi_cvor(int b ) {
6      Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
7      if( novi == NULL)
8          return NULL;
9
10     /* Inicijalizacija polja novog Cvora */
11     novi->broj = b;
12     novi->levo = NULL;
13     novi->desno = NULL;

```



```

15     return novi;
16 }
17
19 void oslobodi_stablo(Cvor** adresa_korena) {
20     /* Prazno stablo i nema sta da se oslobadja */
21     if( *adresa_korena == NULL)
22         return;
23
24     /* Rekurzivno oslobadjamo najpre levo, a onda i desno podstablo */
25     if( (*adresa_korena)->levo )
26         oslobodi_stablo(&(*adresa_korena)->levo);
27     if( (*adresa_korena)->desno)
28         oslobodi_stablo(&(*adresa_korena)->desno);
29
30     free(*adresa_korena);
31     *adresa_korena = NULL;
32 }
33
35 void prover_i_alokaciju( Cvor* novi) {
36     if( novi == NULL) {
37         fprintf(stderr, "Malloc greska za nov cvor!\n");
38         exit(EXIT_FAILURE);
39     }
40 }
41
43 void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
44     /* Postojece stablo je prazno*/
45     if( *adresa_korena == NULL){
46         Cvor* novi = napravi_cvor(broj);
47         prover_i_alokaciju(novi);
48         *adresa_korena = novi; /* Kreirani Cvor novi ce biti od
49 sada koren stabla*/
50         return;
51     }
52
53     /* Brojeve smestamo u uredjeno binarno stablo, pa
54 ako je broj koji ubacujemo manji od broja koji je u korenu */
55     if( broj < (*adresa_korena)->broj)
56         /* Dodajemo u levo podstablo */
57         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
58     /* Ako je broj manji ili jednak od broja koji je u korenu stabla
59 , dodajemo nov Cvor desno od korena */
60     else
61         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
62 }
63
64 #ifndef __STABLA_H__
65 #define __STABLA_H__ 1
66
67 /* Struktura kojom se predstavlja Cvor drвета */
68 typedef struct dcvor{
69     int broj;
70     struct dcvor* levo, *desno;

```

```
8 } Cvor;

10 /* Funkcija alokira prostor za novi Cvor drveta, inicijalizuje polja
    strukture i vraca pokazivac na nov Cvor */
12 Cvor* napravi_cvor(int b );

14 /* Oslobadjamo dinamički alokirani prostor za stablo
    * Nakon oslobadjanja se u pozivajućoj funkciji koren
16 * postavlja NULL, jer je stablo prazno */
void oslobodi_stablo(Cvor** adresa_korena);

18

20 /* Funkcija proverava da li je novi Cvor ispravno alokiran,
    * i nakon toga prekida program */
22 void prover_i_alokaciju( Cvor* novi);

24

26 /* Funkcija dodaje nov Cvor u stablo i
    * azurira vrednost korena stabla u pozivajućoj funkciji.
    */
28 void dodaj_u_stablo(Cvor** adresa_korena, int broj);

30 #endif
```

### Rešenje 5.3

```
#include <stdio.h>
2 #define MAX 50

4

int main(){
6     int m[MAX][MAX];
    int v, k;
8     int i, j;
    int max_broj_negativnih, max_indeks_kolone;
10    int broj_negativnih;

12    /* Ucitavamo dimenzije matrice */
    scanf("%d", &v);
14    scanf("%d", &k);

16    if(v<0 || v>MAX || k<0 || k>MAX){
        fprintf(stderr, "-1\n");
18        return 0;
    }

20    /* Ucitavamo elemente matrice */
22    for(i=0; i<v; i++){
        for(j=0; j<k; j++){
24            scanf("%d", &m[i][j]);
        }
26    }

28    /*Pronalazimo kolonu koja sadrzi najveći broj negativnih
    elemenata */
```

```

max_indeks_kolone=0;
30
max_broj_negativnih=0;
32 for(i=0; i<v; i++){
    if(m[i][0]<0){
34         max_broj_negativnih++;
    }
36
    }
38
    for(j=0; j<k; j++){
        broj_negativnih=0;
        40 for(i=0; i<v; i++){
            if(m[i][j]<0){
                broj_negativnih++;
            }
            44 if(broj_negativnih>max_broj_negativnih){
                max_indeks_kolone=j;
            }
            46
        }
        48
    }
    50
    /* Ispisujemo trazeni rezultat */
    52 printf("%d\n", max_indeks_kolone);
    54
    /* Završavamo program */
    56 return 0;
}

```

### Rešenje 5.4

```

1 #include <stdio.h>
  #include <stdlib.h>
3 #include <string.h>
  #define MAX 128
5
int main(int argc, char **argv) {
7     FILE *f;
    int brojac = 0;
9     char linija[MAX], *p;
11
    if (argc != 3) {
        fprintf(stderr, "-1\n");
13        exit(EXIT_FAILURE);
    }
15
    if ((f = fopen(argv[1], "r")) == NULL) {
17        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
19    }
21
    while (fgets(linija, MAX, f) != NULL) {
        p = linija;
23        while (1) {

```

```
    p = strstr(p, argv[2]);
25     if (p == NULL)
        break;
27     brojac++;
    p = p + strlen(argv[2]);
29 }
}
31
fclose(f);
33
printf("%d\n", brojac);
35
return 0;
37 }
```

### Rešenje 5.5

```
#include <stdio.h>
2  #include <stdlib.h>
    #include <math.h>
4
typedef struct _trougao {
6     double xa, ya, xb, yb, xc, yc;
} trougao;
8
double duzina(double x1, double y1, double x2, double y2) {
10     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}
12
double povrsina(trougao t) {
14     double a = duzina(t.xb, t.yb, t.xc, t.yc);
    double b = duzina(t.xa, t.ya, t.xc, t.yc);
16     double c = duzina(t.xa, t.ya, t.xb, t.yb);
    double s = (a + b + c) / 2;
18     return sqrt(s * (s - a) * (s - b) * (s - c));
}
20
int poredi(const void *a, const void *b) {
22     trougao x = *(trougao*)a;
    trougao y = *(trougao*)b;
24     double xp = povrsina(x);
    double yp = povrsina(y);
26     if (xp < yp)
        return 1;
28     if (xp > yp)
        return -1;
30     return 0;
}
32
int main() {
34     FILE *f;
    int n, i;
36     trougao *niz;

38     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
```

```

    fprintf(stderr, "-1\n");
40    exit(EXIT_FAILURE);
}

42
if (fscanf(f, "%d", &n) != 1) {
44    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
46 }

48 if ((niz = malloc(n * sizeof(trougao))) == NULL) {
    fprintf(stderr, "-1\n");
50    exit(EXIT_FAILURE);
}

52
for (i = 0; i < n; i++) {
54     if (fscanf(f, "%lf%lf%lf%lf%lf%lf",
                &niz[i].xa, &niz[i].ya,
56                &niz[i].xb, &niz[i].yb,
                &niz[i].xc, &niz[i].yc) != 6) {
58         fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
60     }
}

62
qsort(niz, n, sizeof(trougao), &poredi);

64
for (i = 0; i < n; i++)
66     printf("%g %g %g %g %g %g\n",
            niz[i].xa, niz[i].ya,
68            niz[i].xb, niz[i].yb,
            niz[i].xc, niz[i].yc);
70

72 free(niz);
    fclose(f);

74 return 0;
}

```

```

1  #include <stdio.h>
    #include <stdlib.h>
3  #include "stabla.h"

5  Cvor *napravi_cvor(int broj)
    {
7
    /* Dinamicki kreiramo cvor */
9     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

11    /* U slucaju greske ... */
        if (novi == NULL) {
13        fprintf(stderr, "-1\n");
        exit(1);
15    }

17    /* Inicijalizacija */
        novi->vrednost = broj;

```

```

19     novi->levi = NULL;
    novi->desni = NULL;

21

22     /* Vracamo adresu novog cvora */
23     return novi;
24 }
25
26 void dodaj_u_stablo(Cvor **koren, int broj)
27 {
28
29     /* Izlaz iz rekurzije: ako je stablo bilo prazno,
    novi koren je upravo novi cvor */
30     if (*koren == NULL) {
31         *koren = napravi_cvor(broj);
32         return;
33     }
34
35     /* Ako je stablo neprazno, i koren sadrzi manju vrednost
    od datog broja, broj se umece u desno podstablo,
    rekursivnim pozivom */
36     if ((*koren)->vrednost < broj)
37         dodaj_u_stablo(&(*koren)->desni, broj);
38     /* Ako je stablo neprazno, i koren sadrzi vecu vrednost
    od datog broja, broj se umece u levo podstablo,
    rekursivnim pozivom */
39     else if ((*koren)->vrednost > broj)
40         dodaj_u_stablo(&(*koren)->levi, broj);
41
42 }
43
44 void prikazi_stablo(Cvor * koren)
45 {
46     /* Izlaz iz rekurzije */
47     if (koren == NULL)
48         return;
49
50     prikazi_stablo(koren->levi);
51     printf("%d ", koren->vrednost);
52     prikazi_stablo(koren->desni);
53 }
54
55 Cvor* ucitaj_stablo() {
56     Cvor *koren = NULL;
57     int x;
58     while (scanf("%d", &x) == 1)
59         dodaj_u_stablo(&koren, x);
60     return koren;
61 }
62
63 void oslobodi_stablo(Cvor **koren)
64 {
65     /* Izlaz iz rekurzije */
66     if (*koren == NULL)
67         return;
68 }

```

```

75     oslobodi_stablo(&(*koren)->levi);
       oslobodi_stablo(&(*koren)->desni);
77     free(*koren);

79     *koren = NULL;
   }

```

```

1  #ifndef __STABLA_H__
   #define __STABLA_H__ 1

3
   /* Struktura koja predstavlja cvor stabla */
5  typedef struct cvor {
       int vrednost; /* Vrednost koja se cuva */
       struct cvor *levi; /* Pokazivac na levo podstablo */
       struct cvor *desni; /* Pokazivac na desno podstablo */
9  } Cvor;

11 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira
    dinamički, funkcijom malloc(). U slučaju greske program
13 se prekida i ispisuje se poruka o gresci. U slučaju
    uspeha inicijalizuje se vrednost datim brojem, a pokazivaci
15 na podstabla se inicijalizuju na NULL. Funkcija vraća
    adresu novokreiranog cvora */
17 Cvor *napravi_cvor(int broj);

19 /* Funkcija dodaje novi cvor u stablo sa datim korenom.
    Ukoliko broj već postoji u stablu, ne radi ništa.
    Cvor se kreira funkcijom napravi_cvor(). */
21 void dodaj_u_stablo(Cvor **koren, int broj);

23
   /* Funkcija prikazuje stablo s leva u desno (tj.
25 prikazuje elemente u rastućem poretku) */
   void prikazi_stablo(Cvor * koren);

27
   /* Funkcija učitava stablo sa standardnog ulaza do kraja ulaza i
       vraća
29 pokazican na njegov koren */
   Cvor* ucitaj_stablo();

31
   /* Funkcija oslobadja prostor koji je alociran za
33 cvorove stabla. */
   void oslobodi_stablo(Cvor **koren);

35
   #endif

```

### Rešenje 5.6

```

   #include <stdio.h>
2  #include "stabla.h"

4  int f3(Cvor *koren, int n) {
       if (koren == NULL || n < 0)
6         return 0;
       if (n == 0) {
8         if (koren -> levi == NULL && koren -> desni != NULL)

```

```
10     return 1;
11     if (koren -> levi != NULL && koren -> desni == NULL)
12         return 1;
13     return 0;
14 }
15 return f3(koren -> levi, n - 1) + f3(koren -> desni, n - 1);
16 }
17
18 int main() {
19     Cvor *koren;
20     int n;
21
22     scanf("%d", &n);
23     koren = ucitaj_stablo();
24
25     printf("%d\n", f3(koren, n));
26
27     oslobodi_stablo(&koren);
28
29     return 0;
30 }
```