

Univerzitet u Beogradu  
Matematički fakultet

Milena Vujošević Jančić, Jelena Graovac, Ana Spasić,  
Mirko Spasić, Anđelka Zečević, Nina Radojičić

## 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, ispravnom radu sa pokazivačima i dinamički alociranom memorijom, osnovnim algoritmima pretraživanja i sortiranja, kao i radu 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 . . . . .	12
1.4	Rešenja . . . . .	19
<b>2</b>	<b>Pokazivači</b>	<b>61</b>
2.1	Pokazivačka aritmetika . . . . .	61
2.2	Višedimenzioni nizovi . . . . .	65
2.3	Dinamička alokacija memorije . . . . .	70
2.4	Pokazivači na funkcije . . . . .	73
2.5	Rešenja . . . . .	75
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>113</b>
3.1	Pretraživanje . . . . .	113
3.2	Sortiranje . . . . .	118
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	127
3.4	Rešenja . . . . .	132
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>205</b>
4.1	Liste . . . . .	205
4.2	Stabla . . . . .	219
4.3	Rešenja . . . . .	227
<b>5</b>	<b>Ispitni rokovi</b>	<b>319</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	319
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	321
5.3	Rešenja . . . . .	323



# 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 imaginaran 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 odgovarajućem 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 `imaginaran_deo` koja računa vrednosti imaginarnog dela broja.
- (f) Napisati funkciju `modulo` koja računa modulo 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.

## 1 Uvodni zadaci

---

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

### Test 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. Test program treba da koristi tu biblioteku da za kompleksan broj unet sa standardnog ulaza ispiše polarni oblik unetog broja.

### Test Test 1

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

[Rešenje 1.2]

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

- (a) Definisati strukturu `Polinom` koja predstavlja polinom (stepena najviše 20). Struktura sadrži stepen i niz koeficijenata. Redosled navođenja koeficijenata u nizu treba da bude takav da na nultoj poziciji u nizu bude koeficijent uz slobodan član, na prvoj koeficijent uz prvi stepen, itd.



- (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 koristeći Hornerov algoritam.
- (e) Napisati funkciju koja sabira dva polinoma.
- (f) Napisati funkciju koja množi dva polinoma.

Napisati program koji testira prethodno napisane funkcije tako što se najpre unosi polinom  $p$  (stepen polinoma, a zatim i koeficijenti) i ispisuje na standardan izlaz u odgovarajućem obliku. Nakon toga se od korisnika traži da unese tačku u kojoj se računa vrednost tog polinoma a zatim se ispisuje izračunata vrednost zaokružena na dve decimale. Nakon toga se unosi polinom  $q$ , a potom se ispisuju zbir i proizvod polinoma  $p$  i  $q$ . Na kraju se sa standardnog ulaza unosi broj  $n$ , a potom se ispisuje  $n$ -ti izvod polinoma  $p$ .

### Test Upotreba programa 1

```
Unesite polinom (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
Unesite tačku u kojoj racunate vrednost polinoma
5
Vrednost polinoma u tacki je 194.00
Unesite drugi polinom (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
2 1 0 1
Zbir polinoma je: 1.00x3+3.00x2+3.00x+5.00
Prozvod polinoma je: 1.00x5+2.00x4+4.00x3+6.00x2+3.00x+4.00
Unesite izvod polinoma koji zelite:
2
2. izvod prvog polinoma je: 6.00x+4.00
```

[Rešenje 1.3]

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

- (a) Definisati 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 skraćene vrednosti razlomaka koji su dobijeni kao zbir, razlika, proizvod i količnik razlomka **r1** i recipročne vrednosti razlomka **r2**.

*Test Test 1*

```

Ulaz:   Unesite imenilac i brojilac prvog razlomka: 1 2
        Unesite imenilac i brojilac drugog razlomka: 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 neoznačenog 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:
    00000000000000000000000001111111
```

*Test Test 2*

```
| Ulaz:    0x80  
| Izlaz:   000000000000000000000000000000000000000000000000000
```

*Test Test 3*

```
Ulaz: 0x00FF00FF
Izlaz: 00000000111111110000000011111111
```

*Test Test 4*

```
Ulaz:      0xFFFFFFFF
Izlaz:     111111111111111111111111111111111111
```

*Test Test*

```
Ulaz: 0xABCDE123
Izlaz: 10101011110011011110000100100011
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkciju koja broji bitove postavljene na 1 u zapisu celog broja  $x$ , tako što se pomeranje vrši nad

- (a) maskom,
- (b) brojem  $x$ .

Napisati program koji testira tu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

*Test Test 1*

```
|| Ulaz: 0x7F
|| Izlaz:
|| Broj jedinica u zapisu je 7
```

*Test Test 2*

```
|| Ulaz: 0x80
|| Izlaz:
|| Broj jedinica u zapisu je 1
```

*Test Test 3*

```
|| Ulaz: 0x00FF00FF
|| Izlaz:
|| Broj jedinica u zapisu je 16
```

*Test Test 4*

```
|| Ulaz: 0xFFFFFFFF
|| Izlaz:
|| Broj jedinica u zapisu je 32
```

*Test Test 4*

```
|| Ulaz: 0xABCDE123
|| Izlaz:
|| Broj jedinica u zapisu je 17
```

[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 Test 1*

```
|| Ulaz: 0x7F
|| Izlaz:
|| Najveci:
|| 11111110000000000000000000000000
|| Najmanji:
|| 0000000000000000000000000000111111
```

*Test Test 2*

```
|| Ulaz: 0x80
|| Izlaz:
|| Najveci:
|| 10000000000000000000000000000000
|| Najmanji:
|| 00000000000000000000000000000001
```

## 1 Uvodni zadaci

---

### Test Test 3

```
|| Ulaz:  0x00FF00FF
|| Izlaz:
|| Najveci:
|| 11111111111111111000000000000000
|| Najmanji:
|| 00000000000000000111111111111111
```

### Test Test 4

```
|| Ulaz:  0xFFFFFFFF
|| Izlaz:
|| Najveci:
|| 11111111111111111111111111111111
|| Najmanji:
|| 11111111111111111111111111111111
```

### Test Test 4

```
|| Ulaz:  0xABCDE123
|| Izlaz:
|| Najveci:
|| 11111111111111111000000000000000
|| Najmanji:
|| 00000000000000000111111111111111
```

[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 od  $n$  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 bita najmanje težine, pri čemu je pozicija bita najmanje težine nula.*

```

Ulaz:      235 5 10 127
Izlaz:
  Broj      235          = 000000000000000000000000000011101011
  reset(    235,      5,    10) = 000000000000000000000000000001010111
  set(      235,      5,    10) = 00000000000000000000000000001111101011
  get_bits( 235,      5,    10) = 0000000000000000000000000000000000011
  y =                                127 = 000000000000000000000000000001111111
  set_n_bits( 235,      5,    10, 127) = 00000000000000000000000000001111101011
  invert(    235,      5,    10) = 00000000000000000000000000001100101011

```

```

Ulaz:      B10011A7 5
Izlaz:
x           = 10110001000000000001000110100111
rotate_left(2969571751,      5) = 001000000000000100011010011110110
rotate_right(2969571751,      5) = 001111011000100000000000010001101
rotate_right_signed(2969571751, 5) = 001111011000100000000000010001101

```

## 1 Uvodni zadaci

---

a potom i binarnu reprezentaciju broja dobijenog nakon poziva funkcije `mirror` za uneti broj.

*Test Test 1*

```
|| Ulaz: 255
|| Izlaz:
||      0000000000000000000000001001010101
||      10101010010000000000000000000000
```

[Rešenje 1.10]

**Zadatak 1.11** Napisati funkciju `int Broj01(unsigned int n)` koja za dati broj `n` vraća 1 ako u njegovom binarnom zapisu ima više jedinica nego nula, a inače vraća 0. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

*Test Test 1*

```
|| Ulaz: 10
|| Izlaz: 0
```

*Test Test 2*

```
|| Ulaz: 1024
|| Izlaz: 0
```

*Test Test 3*

```
|| Ulaz: 2147377146
|| Izlaz: 1
```

*Test Test 4*

```
|| Ulaz: 1111111115
|| Izlaz: 0
```

[Rešenje 1.11]

**Zadatak 1.12** Napisati funkciju koja broji koliko se puta kombinacija 11 (dve uzastopne jedinice) pojavljuje u binarnom zapisu celog neoznačenog broja `x`. Tri uzastopne jedinice se broje dva puta. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

*Test Test 1*

```
|| Ulaz: 11
|| Izlaz: 1
```

*Test Test 2*

```
|| Ulaz: 1024
|| Izlaz: 0
```

*Test Test 3*

```
|| Ulaz: 2147377146
|| Izlaz: 22
```

*Test Test 4*

```
|| Ulaz: 1111111115
|| Izlaz: 6
```

[Rešenje 1.12]

**Zadatak 1.13** ++ Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama  $i$ ,  $j$ . Pozicije  $i$ ,  $j$  se učitavaju kao parametri komandne linije. Smatrati da je krajnji desni bit binarne reprezentacije 0-ti bit. Pri rešavanju nije dozvoljeno koristiti pomoćni niz niti aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ .

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

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

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

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

### 1.3 Rekurzija

**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.

```
Test Test 1
|| Ulaz: 2 10
|| Izlaz: 1024
```

[Rešenje 1.17]

**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.

<i>Test Test 1</i>	<i>Test Test 2</i>
<pre>   Ulaz: 11    Izlaz: Uneti broj ima paran broj cifara</pre>	<pre>   Ulaz: 123    Izlaz: Uneti broj ima neparan            broj cifara</pre>

[Rešenje 1.18]

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



*Test Test 1*

```
|| Ulaz:   Unesite n (<= 12): 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 Test 1*

```
|| Ulaz:   Unesi koeficijente
||         2 3
||         Unesi koji clan niza racunamo
||         5
|| Izlaz:  F(5) = 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 Test 1*

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

*Test Test 2*

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

*Test Test 3*

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

*Test Test 4*

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

*Test Test 5*

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

[Rešenje 1.21]

**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 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 Test 1*

```
|| Ulaz:  3 2 1 4 21
|| Izlaz: 21
```

*Test Test 2*

```
|| Ulaz:  2 -1 0 -5 -10
|| Izlaz: 2
```

*Test Test 3*

```
|| Ulaz:  1 11 3 5 8 1
|| Izlaz: 11
```

*Test Test 4*

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

[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.

*Test Test 1*

```
|| Ulaz:  3 1 2 3 1 2 3
|| Izlaz: 14
```

*Test Test 2*

```
|| Ulaz:  2 3 5 2 6
|| Izlaz: 36
```

*Test 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.

*Test Test 1*

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

*Test Test 2*

```
|| Ulaz: 11 3 2 11 14 11 43 1
|| Izlaz: 2
```

*Test 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.

*Test Test 1*

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

*Test Test 2*

```
|| Ulaz: 1 2 3 11 1 2 4 3 6
|| Izlaz: ne
```

*Test Test 3*

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

[Rešenje 1.26]

**Zadatak 1.27** Napisati rekurzivnu funkciju koja vraća broj bitova koji su postavljeni na 1, u binarnoj reprezentaciji njenog celobrojnog argumenta. Napisati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza u heksadekadnom formatu.

*Test Test 1*

```
|| Ulaz: 0x7F
|| Izlaz: 7
```

*Test Test 2*

```
|| Ulaz: 0x80
|| Izlaz: 1
```

*Test Test 3*

```
|| Ulaz: 0x00FF00FF
|| Izlaz: 16
```



*Test Test 4*

```

|| Ulaz: 165
|| Izlaz: 10

```

[Rešenje 1.30]

**Zadatak 1.31** 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 Test 1*

```

|| Ulaz:  programiranje
|| Izlaz:  ne

```

*Test Test 2*

```

|| Ulaz:  anavolimilovana
|| Izlaz:  da

```

*Test Test 3*

```

|| Ulaz:  a
|| Izlaz:  da

```

*Test Test 4*

```

|| Ulaz:  aba
|| Izlaz:  da

```

*Test Test*

```

|| Ulaz:  aa
|| Izlaz:  da

```

[Rešenje 1.31]

\* **Zadatak 1.32** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za proizvoljan prirodan broj  $n$  ( $n \leq 50$ ) unet sa standardnog ulaza.

*Test Test 1*

```

|| Ulaz:  Unesite duzinu permutacije: 3
|| Izlaz:  1 2 3
||          1 3 2
||          2 1 3
||          2 3 1
||          3 1 2
||          3 2 1

```

[Rešenje 1.32]

\* **Zadatak 1.33** 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 Uvodni zadaci

```

      1   4   6   4   1
     1   5  10  10   5   1

```

- (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 za unetu veličinu Paskalovog trougla i hipotenuzu najpre iscrtava Paskalov trougao a zatim sumu elemenata hipotenuze.

*Test 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 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 1.33]

**Zadatak 1.34** 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 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 1.35** *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 1.36** *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.

## 1.4 Rešenja

### Rešenje 1.1

```

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

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

10
11 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
12 kompleksnog broja i smesta ih u strukturu cija adresa je
13 argument funkcije */
14 void ucitaj_kompleksan_broj(KompleksanBroj * z)
15 {
16     printf("Unesite realan i imaginaran deo kompleksnog broja: ");
17     scanf("%f", &z->real);
18     scanf("%f", &z->imag);
19 }

20
21 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji
22 joj se salje kao argument u obliku (x + y i) Ovoj funkciji se
23 kompleksan broj prenosi po vrednosti, jer za ispis nam nije
24 neophodno da imamo adresu */
25 void ispisi_kompleksan_broj(KompleksanBroj z)
26 {

```

```
printf("(");
28 if (z.real != 0) {
    printf("%.2f", z.real);
30
    if (z.imag > 0)
32         printf(" + %.2f i", z.imag);
    else if (z.imag < 0)
34         printf(" - %.2f i", -z.imag);
    } else
36         printf("%.2f i", z.imag);

38 if (z.imag == 0 && z.real == 0)
    printf("0");
40
42 printf(")");
}

44 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
float realan_deo(KompleksanBroj z)
46 {
    return z.real;
48 }

50 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
float imaginaran_deo(KompleksanBroj z)
52 {
    return z.imag;
54 }

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

62
64 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
    odgovara kompleksnom broju poslatom kao argument */
KompleksanBroj konjugovan(KompleksanBroj z)
66 {
    KompleksanBroj z1 = z;
68     z1.imag *= -1;
    return z1;
70 }

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

78     z.real += z2.real;
```



```
    z.imag += z2.imag;
80
    return z;
82 }

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

90     z.real -= z2.real;
    z.imag -= z2.imag;

92     return z;
94 }

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

102     z.real = z1.real * z2.real - z1.imag * z2.imag;
    z.imag = z1.real * z2.imag + z1.imag * z2.real;

104     return z;
106 }

108 /* Funkcija vraca argument kompleksnog broja koji je funkciji
    poslat kao argument */
110 float argument(KompleksanBroj z)
{
112     return atan2(z.imag, z.real);
    }
114

116 /* U main() funckiji testiramo sve funckije koje smo definisali */
int main()
{
118     /* deklariseemo promenljive tipa KompleksanBroj */
120     KompleksanBroj z1, z2;

122     /* Ucitavamo prvi kompleksan broj */
    ucitaj_kompleksan_broj(&z1);

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

128     /* Ispisujemo prvi kompleksan broj, a zatim i njegov realan i
    imaginaran deo, kao i moduo kompleksnog broja z1 */
130     ispisi_kompleksan_broj(z1);
```

```

132 printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
    realan_deo(z1), imaginaran_deo(z1), moduo(z1));
134 printf("\n");
136 /* Ispisujemo drugi kompleksan broj, a zatim i racunamo i
    ispisujemo konjugovano kompleksan broj od z2 */
138 ispisi_kompleksan_broj(z2);
139 printf("\nNjegov konjugovano kompleksan broj: ");
140 ispisi_kompleksan_broj(konjugovan(z2));
142 /* Testiramo funkciju koja racuna argument kompleksnih brojeva
    */
143 printf("\nArgument kompleksnog broja: %.f\n", argument(z2));
144 printf("\n");
146 /* Testiramo sabiranje kompleksnih brojeva */
147 ispisi_kompleksan_broj(z1);
148 printf(" + ");
149 ispisi_kompleksan_broj(z2);
150 printf(" = ");
151 ispisi_kompleksan_broj(saberi(z1, z2));
152 printf("\n");
154 /* Testiramo oduzimanje kompleksnih brojeva */
155 printf("\n");
156 ispisi_kompleksan_broj(z1);
157 printf(" - ");
158 ispisi_kompleksan_broj(z2);
159 printf(" = ");
160 ispisi_kompleksan_broj(oduzmi(z1, z2));
161 printf("\n");
162 /* Testiramo mnozenje kompleksnih brojeva */
163 printf("\n");
164 ispisi_kompleksan_broj(z1);
165 printf(" * ");
166 ispisi_kompleksan_broj(z2);
167 printf(" = ");
168 ispisi_kompleksan_broj(mnozi(z1, z2));
169 printf("\n");
170 /* Program se zavrшава uspesno, tj, bez greske */
172 return 0;
}

```

### Rešenje 1.2

```

2 /* Uključujemo zaglavlje neophodno za rad sa kompleksnim brojevima
   * Ovdje je to neophodno jer nam je neophodno da bude poznata
   definicija tipa KompleksanBroj
   * i da budu uključena zaglavlja standardne biblioteke koja smo već
   naveli u complex.h

```

```
4  */
   #include "complex.h"
6
   /* Funkcija ucitava sa standardnog ulaza realan i imaginaran deo
      kompleksnog broja i smesta ih u strukturu cija adresa je argument
      funkcije */
8  void ucitaj_kompleksan_broj(KompleksanBroj* z) {
   printf("Unesite realan i imaginaran deo kompleksnog broja: ");
10  scanf("%f", &z->real);
   scanf("%f", &z->imag);
12  }

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

22         if(z.imag > 0)
             printf(" + %.2f i", z.imag);
24         else if(z.imag < 0)
             printf(" - %.2f i", -z.imag);
26         }
     else
28         printf("%.2f i", z.imag);

30     if(z.imag == 0 && z.real == 0 )
         printf("0");
32
     printf(")");
34  }

36  /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
   float realan_deo(KompleksanBroj z) {
38     return z.real;
   }
40
   /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
42  float imaginaran_deo(KompleksanBroj z) {
     return z.imag;
44  }

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

## 1 Uvodni zadaci

---

```
/* Funkcija vraća vrednost konjugovano kompleksnog broja koji
   odgovara kompleksnom broju poslatom kao argument */
52 KompleksanBroj konjugovan(KompleksanBroj z) {
    KompleksanBroj z1 = z;
54     z1.imag *= -1;
    return z1;
56 }

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

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

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

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

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

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

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

1 /*
   Zaglavlje complex.h sadrži definiciju tipa KompleksanBroj i
   deklaracije funkcija za rad sa kompleksnim brojevima.
3   Zaglavlje nikada ne treba da sadrži definicije funkcija.
```

```

    Bilo koji program koji bi hteo da koristi ove brojeve i funkcije iz
    ove biblioteke, neophodno je da ukljuci ovo zaglavlje
5  */

7  /* Ovim pretprocesorskim direktivama zakljucavamo zaglavlje i time
    onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci.
    Niska posle kljucne reci ifndef je proizvoljna ali treba da se
    ponovi u narednoj pretprocesorskoj define direktivi
9  */
11 #ifndef _COMPLEX_H
13 #define _COMPLEX_H

13 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
    koje se koriste u definicijama funkcija koje smo naveli u complex
    .c */
15 #include <stdio.h>
15 #include <math.h>

17 /* struktura kojom predstavljamo kompleksan broj, cuvajuci njegov
    realan i imaginaran deo */
19 typedef struct {
19     float real;
19     float imag;
21 } KompleksanBroj;

23 /* Deklaracije funkcija za rad sa kompleksnim brojevima.
    Sve one su definisane u complex.c */
25 void ucitaj_kompleksan_broj(KompleksanBroj* z) ;

27 void ispisi_kompleksan_broj(KompleksanBroj z) ;

29 float realan_deo(KompleksanBroj z) ;

31 float imaginaran_deo(KompleksanBroj z);

33 float moduo(KompleksanBroj z);

35 KompleksanBroj konjugovan(KompleksanBroj z) ;

37 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) ;

39 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) ;

41 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) ;

43 float argument(KompleksanBroj z) ;

45 /* Kraj zakljucanog dela */
    #endif

2  /*
    * Koristimo korektno definisanu biblioteku kompleksnih brojeva.

```

## 1 Uvodni zadaci

```

2  * U zaglavlju complex.h nalazi se definicija kompleksnog broja i
3  * 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  *
8  * I dalje kompilacija programa se najjednostavnije postize naredbom
9  * gcc -Wall -lm -o izvrsni complex.c main.c
10
11  Kompilaciju mozemo uraditi i na sledeci nacin:
12  gcc -Wall -c -o complex.o complex.c
13  gcc -Wall -c -o main.o main.c
14  gcc -lm -o complex complex.o main.o
15  */
16
17
18  #include <stdio.h>
19  /* Ukljuujemo zaglavlje neophodno za rad sa kompleksnim brojevima */
20  #include "complex.h"
21
22  /* U main funkciji za uneti kompleksan broj ispisujemo njegov polarni
23  * oblik */
24  int main() {
25      KompleksanBroj z;
26
27      /* Ucitavamo kompleksan broj */
28      ucitaj_kompleksan_broj(&z);
29
30      printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f\n",
31             moduo(z), argument(z));
32
33      return 0;
34  }
```

### Rešenje 1.3

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "polinom.h"
4
5
6  /* Funkcija koja ispisuje polinom na standardan izlaz u citljivom
7  * obliku.
8  * Kako bi uštedeli kopiranje cele strukture, polinom prenosimo po
9  * adresi */
10 void ispisi(const Polinom * p)
11 {
12     int i;
13     for (i = p->stepen; i >= 0; i--) {
14         if (p->koeff[i]) {
15             if (p->koeff[i] >= 0 && i != p->stepen)
16                 printf("%d", p->koeff[i]);
17             else
18                 printf("%d", -p->koeff[i]);
19             printf("x^%d", i);
20             if (i > 0)
21                 printf(" + ");
22         }
23     }
24     printf("\n");
25 }
```

```

15     putchar('+');
16     if (i > 1)
17         printf("%.2fx^%d", p->koef[i], i);
18     else if (i == 1)
19         printf("%.2fx", p->koef[i]);
20     else
21         printf("%.2f", p->koef[i]);
22 }
23 }
24 putchar('\n');
25 }
26
27 /* Funkcija koja ucitava polinom sa tastature */
28 Polinom ucitaj()
29 {
30     int i;
31     Polinom p;
32
33     /* Ucitavamo stepen polinoma */
34     scanf("%d", &p.stepen);
35
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
43     /* Unosimo koeficijente polinoma */
44     for (i = p.stepen; i >= 0; i--)
45         scanf("%lf", &p.koef[i]);
46     return p;
47 }
48
49 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
50 algoritmom */
51 /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
52 double izracunaj(const Polinom * p, double x)
53 {
54     double rezultat = 0;
55     int i = p->stepen;
56     for (; i >= 0; i--)
57         rezultat = rezultat * x + p->koef[i];
58     return rezultat;
59 }
60
61 /* Funkcija koja sabira dva polinoma */
62 Polinom saberi(const Polinom * p, const Polinom * q)
63 {
64     Polinom rez;
65     int i;

```

```
        rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
65
        for (i = 0; i <= rez.stepen; i++)
67            rez.koef[i] =
                (i > p->stepen ? 0 : p->koef[i]) + (i >
69                    q->stepen ? 0 : q->
                        koef[i]);
71        return rez;
73    }

75    /* Funkcija mnozi dva polinoma p i q */
    Polinom pomnozi(const Polinom * p, const Polinom * q)
77    {
        int i, j;
79        Polinom r;

81        r.stepen = p->stepen + q->stepen;
        if (r.stepen > MAX_STEPEN) {
83            fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
            exit(EXIT_FAILURE);
85        }

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

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

93        return r;
95    }

97    /* Funkcija racuna izvod polinoma p */
    Polinom izvod(const Polinom * p)
99    {
        int i;
101        Polinom r;

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

111        return r;
    }

113
115    /* Funkcija racuna n-ti izvod polinoma p */
    Polinom nIzvod(const Polinom * p, int n)
```



```

{
117     int i;
        Polinom r;
119
        if (n < 0) {
121             fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
            exit(EXIT_FAILURE);
123         }

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

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

131         return r;
133     }

```

```

1
/* Ovim preprocesorskim 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 koja cuva
   koeficijente (koef[i] je koeficijent uz clan x^i)
   i stepen polinoma */
17 typedef struct {
19     double koef[MAX_STEPEN + 1];
    int stepen;
21 } Polinom;

23 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku
   Polinom prenosimo po adresi, da bi ustedeli 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();

```

## 1 Uvodni zadaci

---

```
31 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
    algoritmom */
    /*  $x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 1$  */
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 /*
    Prevodjenje:
6 gcc -o test-polinom polinom.c main.c

8 ili:
    gcc -c polinom.c
10 gcc -c main.c
    gcc -o test-polinom polinom.o main.o
12 */

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

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

28         printf("Unesite tacku u kojoj racunate vrednost polinoma\n");
            scanf("%lf", &x);
30
            /* Ispisujemo vrednost polinoma u toj tacki */
32         printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&p, x));
```

```

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

38     /* Sabiramno polinome i ispisujemo zbir ta dva polinoma */
    r = saberi(&p, &q);
    printf("Zbir polinoma je: ");
    ispisi(&r);

42     /* Mnozimo polinome i ispisujemo proizvod ta dva polinoma */
    r = pomnozi(&p, &q);
    printf("Prozvod polinoma je: ");
    ispisi(&r);

44     /* Izvod polinoma */
    printf("Unesite izvod polinoma koji zelite:\n");
    scanf("%d", &n);
    r = nIzvod(&p, n);
    printf("%d. izvod prvog polinoma je: ", n);
    ispisi(&r);

46     /* Uspesno završavamo program */
    return 0;
58 }

```

## Rešenje 1.5

```

#include <stdio.h>

2
/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
4  celog broja u memoriji. Bitove u zapisu broja
    ispisujemo sa leva na desno, tj. od bita najvece tezine ka
6  bitu najmanje tezine. */
void print_bits(unsigned x)
8 {

10     /* broj bitova celog broja */
    unsigned velicina = sizeof(unsigned) * 8;
12     /* maska koja se koristi za "ocitavanje" bitova */
    unsigned maska;

14     /* Pocetna vrednost maske se postavlja na broj ciji binarni
        zapis na mestu bita najvece tezine sadrzi jedinicu, a na
16     svim ostalim mestima sadrzi nulu. U svakoj iteraciji ova
        jedinica se pomera u desno, kako bi se ocitao naredni bit.
        Odgovarajuci
18     karakter, ('0' ili '1'), ispisuje se na standardnom izlazu. Zbog
        siftovanja maske u desno, koja na pocetku ima najvisi bit
20     postavljen na 1, neophodno je da maska bude neoznaceni ceo

```

## 1 Uvodni zadaci

---

```
22     broj kako bi se siftovanjem u desno vrsilo logicko siftovanje
    (popunjavanje nulama) a ne aritmeticko siftovanje (popunjavanje
24     znakom broja). */
    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
26         putchar(x & maska ? '1' : '0');

28     putchar('\n');
    }

30

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

38     return 0;
    }
```

### Rešenje 1.6

```
1  #include <stdio.h>

3  /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
5  void print_bits(int x)
    {
7      unsigned velicina = sizeof(int) * 8;
        unsigned maska;

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

13     putchar('\n');
    }

15

17 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
    x pomeranjem broja x */
17 int count_bits(int x)
    {
19     int br = 0;
21     unsigned wl = sizeof(unsigned) * 8 - 1;

23     /* Formiramo masku 100000...0000000, koja služi za ocitavanje
        bita najveće težine. U svakoj iteraciji maska se pomera u
25     desno za 1 mesto, i ocitavamo sledeći bit. Petlja se
        završava kada više nema jedinica tj. kada maska postane
27     nula. */
        unsigned maska = 1 << wl;
29     for (; maska != 0; maska >>= 1)
        {
            x & maska ? br++ : 1;
        }
    }
```

```

31     return br;
33 }

35
37 int main()
38 {
39     int x;
40     scanf("%x", &x);
41     printf("Broj jedinica u zapisu je %d.\n", count_bits(x));
42
43     return 0;
44 }

```

```

1  #include <stdio.h>

3  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   * celog broja u memoriji */
5  void print_bits(int x)
6  {
7      unsigned velicina = sizeof(int) * 8;
8      unsigned maska;

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

12     putchar('\n');
13 }

15 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
   * x pomeranjem broja x */
17 int count_bits1(int x)
18 {
19     int br = 0;
20     unsigned wl = sizeof(int) * 8 - 1;

22     /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1
   * vrsila
   * bi aritmeticko pomeranje u desno, tj. popunjavanje bita najvece
   * tezine bitom znaka. U tom slucaju nikad ne bi bio ispunjen
   * uslov x!=0 i program bi bio zarobljen u beskonacnoj petlji.
   * Zbog toga se koristi pomeranj broja x ulevo i maska koja
   * ocitava bit najvece tezine. */

24     unsigned maska = 1 << wl;
25     for (; x != 0; x <<= 1)
26         x & maska ? br++ : 1;

28     return br;
29 }

31
33
35
37

```

## 1 Uvodni zadaci

---

```
int main()
39 {
    int x;
41     scanf("%x", &x);
    printf("Broj jedinica u zapisu je %d.\n", count_bits1(x));
43
    return 0;
45 }
```

### Rešenje 1.7

```
#include <stdio.h>

2
/* Funkcija vraca najveći neoznačeni broj sastavljen iz istih
4   bitova kao i x */
unsigned najveći(unsigned x)
6 {
    unsigned velicina = sizeof(unsigned) * 8;
8
    /* Formiramo masku 100000...00000000 */
10    unsigned maska = 1 << (velicina - 1);

12    /* Inicijalizujemo rezultat na 0 */
    unsigned rezultat = 0;
14

16    /* Dokle god postoje jedinice u binarnoj reprezentaciji broja
    x (tj. dokle god je x različit od nule) pomeramo ga ulevo. */
    for (; x != 0; x <= 1) {
18        /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
        leva u rezultat */
20        if (x & maska) {
            rezultat >>= 1;
22            rezultat |= maska;
        }
24    }

26    return rezultat;
}

28
/* Funkcija vraca najmanji neoznačen broj sa istim binarnim
30   ciframa kao i x */
unsigned najmanji(unsigned x)
32 {
    /* Inicijalizujemo rezultat na 0 */
34    unsigned rezultat = 0;

36    /* Dokle god imamo jedinice u broju x, pomeramo ga udesno. */
    for (; x != 0; x >= 1) {
38        /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
        desna u rezultat */
40        if (x & 1) {
```

```

    rezultat <<= 1;
    rezultat |= 1;
}
}

return rezultat;
}

/* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
celog broja u memoriji */
void print_bits(int x)
{
    unsigned velicina = sizeof(int) * 8;
    unsigned maska;

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

    putchar('\n');
}

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

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

    printf("Najmanji:\n");
    print_bits(najmanji(broj));

    return 0;
}

```

### 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 */
7 unsigned reset(unsigned x, unsigned n, unsigned p)
{
9     /* 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, dok su svi ostali
   postavljeni na 1.

13     Na primer, za n=5 i p=10 formiramo masku oblika 1111 1111

```

## 1 Uvodni zadaci

```
15      1111 1111 1111 1000 0011 1111 To postizemo na sledeci
      nacin: ~0 1111 1111 1111 1111 1111 1111 1111 (~0 << n)
17      1111 1111 1111 1111 1111 1111 1110 0000 ~(~0 << n) 0000
      0000 0000 0000 0000 0000 0001 1111 (~(~0 << n) << ( p-n+1))
19      0000 0000 0000 0000 0000 0111 1100 0000 ~(~(~0 << n) << (
      p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111 */
21      unsigned maska = ~(~(~0 << n) << (p - n + 1));

23      return x & maska;
25  }

27  /* 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
29      1010 1110 0111 1010 1011 1100 1101 1110 1111 1110 0111 */
      unsigned set(unsigned x, unsigned n, unsigned p)
31  {
      /* Kako zelimo da samo odredjenih n bitova postavimo na 1, dok
33      ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
      formiramo masku oblika 0000 0000 0000 0000 0000 0111 1100
35      0000 prateci vrlo slican postupak kao za prethodnu funkciju
      */
37      unsigned maska = ~(~0 << n) << (p - n + 1);

39      return x | maska;
41  }

43  /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
      predstavlja n bitova pocev od pozicije p u binarnoj
      reprezentaciji broja x, pri cemu se pozicija broji sa desna
45      ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
      i broj 1010 1011 1100 1101 1110 1010 1110 0111 0000 0000 0000
47      0000 0000 0000 0000 1011 */
      unsigned get_bits(unsigned x, unsigned n, unsigned p)
49  {
      /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a
51      ostali su 0. Na primer za n=5 0000 0000 0000 0000 0000 0000
      0001 1111 */
53      unsigned maska = ~(~0 << n);

55      /* Pomeramo sadrzaj u desno tako da trazeno polje bude uz
      desni kraj. Zatim maskiramo ostale bitove, sem zeljenih n i
57      vracamo vrednost */
      return maska & (x >> (p - n + 1));
59  }

61  /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
      postavljeni na vrednosti n bitova najnize tezine binarne
      reprezentacije broja y */
63      unsigned set_n_bits(unsigned x, unsigned n, unsigned p,
65                          unsigned y)
```



```

67 {
68     /* Kreiramo masku kod koje su poslednjih n bitova 1, a
69        ostali su 0. Na primer za n=5 0000 0000 0000 0000 0000 0000
        0001 1111 */
70     unsigned last_n_1 = ~(~0 << n);
71
72     /* Kao ranije u funkciji reset, kreiramo masku koja ima n
73        bitova postavljenih na 0 pocevsi od pozicije p, dok su
74        ostali bitovi 1. Na primer za n=5 i p =10 1111 1111 1111
75        1111 1111 1000 0011 1111 */
76     unsigned middle_n_0 = ~(~0 << n) << (p - n + 1);
77
78     /* x sa resetovanih n bita na pozicijama pocev od p */
79     unsigned x_reset = x & middle_n_0;
80
81     /* y cijih je n bitova najnize tezine pomereni tako da stoje
82        pocev od pozicije p. Ostali bitovi su nule. (y & last_n_1)
83        resetuje sve bitove osim najnižih n */
84     unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);
85
86     return x_reset ^ y_shift_middle;
87 }
88
89
90 /* Funkcija invertuje bitove u zapisu broja x pocevsi od
91    pozicije p njih n */
92 unsigned invert(unsigned x, unsigned n, unsigned p)
93 {
94     /* Formiramo masku sa n jedinica pocev od pozicije p Na primer
95        za n=5 i p=10 0000 0000 0000 0000 0000 0111 1100 0000 */
96     unsigned maska = ~(~0 << n) << (p - n + 1);
97
98     /* Operator ekskluzivno ili invertuje sve bitove gde je
99        odgovarajuci bit maske 1. Ostali bitovi ostaju
100        nepromenjeni. */
101     return maska ^ x;
102 }
103
104
105 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
106    celog broja u memoriji */
107 void print_bits(int x)
108 {
109     unsigned velicina = sizeof(int) * 8;
110     unsigned maska;
111
112     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
113         putchar(x & maska ? '1' : '0');
114
115     putchar('\n');
116 }
117

```

## 1 Uvodni zadaci

```
119
121 int main()
122 {
123     unsigned broj, p, n, y;
124     scanf("%u%u%u%u", &broj, &n, &p, &y);
125     printf("Broj %5u %25s= ", broj, "");
126     print_bits(broj);
127
128     printf("reset(%5u,%5u,%5u)%11s = ", broj, n, p, "");
129     print_bits(reset(broj, n, p));
130
131     printf("set(%5u,%5u,%5u)%13s = ", broj, n, p, "");
132     print_bits(set(broj, n, p));
133
134     printf("get_bits(%5u,%5u,%5u)%8s = ", broj, n, p, "");
135     print_bits(get_bits(broj, n, p));
136
137     printf("y = %31u = ", y);
138     print_bits(y);
139     printf("set_n_bits(%5u,%5u,%5u,%5u) = ", broj, n, p, y);
140     print_bits(set_n_bits(broj, n, p, y));
141
142     printf("invert(%5u,%5u,%5u)%10s = ", broj, n, p, "");
143     print_bits(invert(broj, n, p));
144
145     return 0;
146 }
147
```

### Rešenje 1.9

```
#include <stdio.h>

2
/* Funkcija broj x rotira u levo za n mesta Na primer za n =5 i
4   x cija je interna reprezentacija 1010 1011 1100 1101 1110
   0001 0010 0011 0111 1001 1011 1100 0010 0100 0111 0101 */
6 unsigned rotate_left(int x, unsigned n)
7 {
8     unsigned first_bit;
9     /* Maska koja ima samo najvisi bit postavljen na 1 neophodna
10      da bismo pre siftovanja u levo za 1 sacuvali najvisi bit. */
11     unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
12     int i;

13
14     /* n puta vrsimo rotaciju za jedan bit u levo. U svakoj
15      iteraciji odredimo prvi bit, a potom pomeramo sadrzaj broja
16      x u levo za 1 i potom najnizi bit postavljamo na vrednost
17      koju je imao prvi bit koji smo istisnuli siftovanjem */
18     for (i = 0; i < n; i++) {
19         first_bit = x & first_bit_mask;
```

```

20     x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
21 }
22 return x;
23 }
24
25 /* Funkcija neoznaceni broj x rotira u desno za n Na primer za n
26    =5 i x cija je interna reprezentacija 1010 1011 1100 1101
27    1110 0001 0010 0011 0001 1101 0101 1110 0110 1111 0000 1001 */
28 unsigned rotate_right(unsigned x, unsigned n)
29 {
30     unsigned last_bit;
31     int i;
32
33     /* n puta ponavljamo rotaciju u desno za jedan bit. U svakoj
34        iteraciji odredjujemo bit najmanje tezine broja x, zatm
35        tako odredjeni bit siftujemo u levo tako da najnizi bit
36        dođe do pozicije najviseg bita i nakon siftovanja x za 1 u
37        desno postavljamo x-ov najvisi bit na vrednost najnižeg
38        bita. */
39     for (i = 0; i < n; i++) {
40         last_bit = x & 1;
41         x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
42     }
43
44     return x;
45 }
46
47 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde
48    je x oznaceni broj */
49 int rotate_right_signed(int x, unsigned n)
50 {
51     unsigned last_bit;
52     int i;
53
54     /* U svakoj iteraciji odredjujemo bit najmanje tezine tj.
55        last_bit. Kako je x oznacen ceo broj, tada se prilikom
56        siftovanja u desno vrši aritmeticki sift i cuva se znak
57        broja. Iza tog razloga imamo dva slucaja u zavisnosti od
58        znaka od x. Nije dovoljno da se ova provera izvrši pre
59        petlje, jer rotiranjem u desno na mesto najviseg bita može
60        doći i 0 i 1, nezavisno od pocetnog znaka x. */
61     for (i = 0; i < n; i++) {
62         last_bit = x & 1;
63
64         if (x < 0)
65             /* Siftovanjem u desno broja koji je negativan dobijamo 1
66                na najvisoj poziciji. Na primer ako je x 1010 1011
67                1100 1101 1110 0001 0010 001b (sa b oznacavamo u
68                primeru 1 ili 0 na najnižoj poziciji) last_bit je 0000
69                0000 0000 0000 0000 0000 000b nakon Siftovanja za
70                1 u desno 1101 0101 1110 0110 1111 0000 1001 0001 da

```

## 1 Uvodni zadaci

---

```
72         bismo najvisu 1 u x postavili na b nije dovoljno da ga
73         siftujemo na najvisu poziciju jer bi se time dobile 0,
74         a nama su potrebne 1 zbog bitovskog & zato prvo
75         komplementiramo, pa tek onda siftujemo ~last_bit <<
76         (sizeof(int)*8 -1) B000 0000 0000 0000 0000 0000 0000
77         0000 (B oznacava ~b ) i ponovo komplementiramo da bismo
78         imali b na najvisoj poziciji i sve 1 na ostalim
79         pozicijama ~(~last_bit << (sizeof(int)*8 -1)) b111 1111
80         1111 1111 1111 1111 1111 1111 */
81         x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
82     else
83         x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
84     }
85
86     return x;
87 }
88
89 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
90    celog broja u memoriji */
91 void print_bits(int x)
92 {
93     unsigned velicina = sizeof(int) * 8;
94     unsigned maska;
95     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
96         putchar(x & maska ? '1' : '0');
97
98     putchar('\n');
99 }
100
101 int main()
102 {
103     unsigned x, k;
104     scanf("%x%x", &x, &k);
105     printf("x %36s = ", "");
106     print_bits(x);
107     printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
108     print_bits(rotate_left(x, k));
109
110     printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
111     print_bits(rotate_right(x, k));
112
113     printf("rotate_right_signed(%7u,%6u) = ", x, k);
114     print_bits(rotate_right_signed(x, k));
115
116     return 0;
117 }
```

### Rešenje 1.10

---

```

1  #include <stdio.h>
2
3  /* Funkcija vraća vrednost čija je binarna reprezentacija slika
4     u ogledalu binarne reprezentacije broja x. Na primer za x
5     čija binarna reprezentacija izgleda ovako
6     101010111100110111100100100100011 funkcija treba da vrati
7     broj čija binarna reprezentacija izgleda:
8     11000100100001111011001111010101 */
9  unsigned mirror(unsigned x)
10 {
11     unsigned najnizi_bit;
12     unsigned rezultat = 0;
13
14     int i;
15     /* Krecemo od najnižeg bita u zapisu broja x i dodajemo ga u
16        rezultat */
17     for (i = 0; i < sizeof(x) * 8; i++) {
18         najnizi_bit = x & 1;
19         x >>= 1;
20         /* Potiskujemo trenutni rezultat ka levom kraju. Tako svi
21            prethodno postavljeni bitovi dobijaju veću poziciju. Novi
22            bit postavljamo na najnižu poziciju */
23         rezultat <<= 1;
24         rezultat |= najnizi_bit;
25     }
26     return rezultat;
27 }
28
29 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
30    celog broja u memoriji */
31 void print_bits(int x)
32 {
33     unsigned velicina = sizeof(int) * 8;
34     unsigned maska;
35     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
36         putchar(x & maska ? '1' : '0');
37
38     putchar('\n');
39 }
40
41 int main()
42 {
43     int broj;
44     scanf("%x", &broj);
45
46     /* Ispisujemo binarnu reprezentaciju unetog broja */
47     print_bits(broj);
48
49     /* Ispisujemo binarnu reprezentaciju broja dobijenog pozivom
50        funkcije mirror */
51     print_bits(mirror(broj));
52 }

```

```
54     return 0;
    }
```

### Rešenje 1.11

```
#include <stdio.h>

2
/* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n
4   broj jedinica veci od broja nula. U suprotnom funkcija vraca
   0 */
6 int Broj01(unsigned int n)
{
8
   int broj_nula, broj_jedinica;
10  unsigned int maska;

12  broj_nula = 0;
   broj_jedinica = 0;
14
   /* Postavljamo masku tako da pocinjemo sa analiziranjem bita
16     najvece tezine */
   maska = 1 << (sizeof(unsigned int) * 4 - 1);
18
   /* Dok ne obidjemo sve bitove u zapisu broj n */
20  while (maska != 0) {

22     /* Proveravamo da li se na poziciji koju odredjuje maska
       nalazi 0 ili 1 i uvecavamo odgovarajuci brojac */
24     if (n & maska) {
       broj_jedinica++;
26     } else {
       broj_nula++;
28     }

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

34
   /* Ako je broj jedinica veci od broja nula vracamo 1, u
36     suprotnom vracamo 0 */
   return (broj_jedinica > broj_nula) ? 1 : 0;
38
}

40
int main()
42 {
   unsigned int n;
44
   /* Ucitavamo broj */
```

```
46 scanf("%u", &n);
48 /* Ispisujemo vrednost funkcije */
printf("%d\n", Broj01(n));
50 return 0;
52 }
```

### Rešenje 1.12

```
#include <stdio.h>
2
int broj_parova(unsigned int x)
4 {
6     int broj_parova;
    unsigned int maska;
8
    /* Postavljamo broj parova na 0 */
10    broj_parova = 0;
12
    /* Postavljamo masku tako da mozemo da procitamo da li su dva
        najmanja bita u zapisu broja x 11 */
14    /* broj 3 je binarno 000....00011 */
    maska = 3;
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 */
22        if ((x & maska) == maska) {
            broj_parova++;
24        }
26
        /* Pomeramo broj u desnu stranu tako da mozemo da očitamo
            vrednost sledeceg para bitova. Pomeranjem u desno
            bit najvece tezine se popunjava nulom jer je x
            neoznaceni broj. */
28        x = x >> 1;
30    }
32
    return broj_parova;
34 }
36
int main()
38 {
    unsigned int x;
40
    /* Ucitavamo broj */
```

## 1 Uvodni zadaci

---

```
42 | scanf("%u", &x);  
  
44 | /* Ispisujemo vrednost funkcije */  
    | printf("%d\n", broj_parova(x));  
  
46 | return 0;  
48 | }
```

### Rešenje 1.13

### Rešenje 1.14

```
#include <stdio.h>  
  
2 |  
/*  
4 | Niska koju formiramo je duzine (sizeof(unsigned int)*8)/4 +1  
   | jer za svaku heksadekadnu cifru nam trebaju 4 binarne cifre i  
6 | 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 */  
  
12 | #define MAX_DUZINA sizeof(unsigned int)*2 +1  
14 |  
  
16 | void prevod(unsigned int x, char s[])  
   | {  
18 |     int i;  
20 |     unsigned int maska;  
   |     int vrednost;  
22 |  
   | /* Heksadekadni zapis broja 15 je 000...0001111 - ovo nam  
24 |    odgovara ako hocemo da citamo 4 uzastopne cifre */  
   | maska = 15;  
26 |  
   | /*  
28 |    Broj cemo citati od pozicije najmanje tezine ka poziciji  
   |    najvece tezine; npr. za broj  
30 |    0000000001101000100001111010101 u prvom koraku cemo  
   |    procitati bitove: 000000000110100010000111101<0101>  
32 |    (bitove izdvojene sa <...>) u drugom koraku cemo procitati:  
   |    00000000011010001000011<1101>0101 u trecem koraku cemo  
34 |    procitati: 0000000001101000100<0011>11010101 i tako redom  
  
36 |    indeks i oznacava poziciju na koju smestamo vrednost  
  
38 |    */
```



```

40  for (i = MAX_DUZINA - 2; i >= 0; i--) {
41      /* Vrednost izdvojene cifre */
42      vrednost = x & maska;
43
44      /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter
45      dobijamo dodavanjem ASCII koda '0' Ako je vrednost iz
46      opsega od 10 do 15 odgovarajuci karakter dobijamo tako
47      sto prvo oduzmemo 10 (dobijamo vrednosti od 0 do 5) pa
48      dodamo ASCII kod 'A' (time dobijamo slova 'A', 'B', ...
49      'F') */
50      if (vrednost < 10) {
51          s[i] = vrednost + '0';
52      } else {
53          s[i] = vrednost - 10 + 'A';
54      }
55
56      /* Broj pomeramo za 4 bita u desnu stranu tako da mozemo da
57      procitamo sledecu cifru */
58      x = x >> 4;
59  }
60
61  s[MAX_DUZINA - 1] = '\0';
62  }
63
64  int main()
65  {
66      unsigned int x;
67      char s[MAX_DUZINA];
68
69      /* Ucitavamo broj */
70      scanf("%u", &x);
71
72      /* Pozivamo funkciju */
73      prevod(x, s);
74
75      /* Ispisujemo dobijenu nisku */
76      printf("%s\n", s);
77
78      return 0;
79  }

```

### Rešenje 1.17

```

1  #include <stdio.h>
2
3  /* Iskomentarisan je deo koji se ispisuje svaki put kad se udje
4  u funkciju. Odkomentarisati pozive printf funkcije u obe
5  funkcije da uocite razliku u broju rekurzivnih poziva obe
6  verzije. */

```

## 1 Uvodni zadaci

---

```
8  /* Linearno resenje se zasniva na cinjenici:  $x^0 = 1$   $x^k = x * x^{(k-1)}$  */
10 int stepen(int x, int k)
11 {
12     // printf("Racunam stepen (%d, %d)\n", x, k);
13     if (k == 0)
14         return 1;
15
16     return x * stepen(x, k - 1);
17
18     /* Celo telo funkcije se moze ovako kratko zapisati return k
19        == 0 ? 1 : x * stepen(x,k-1); */
20 }
21
22 /* Druga verzija prethodne funkcije. Obratiti paznju na
23    efikasnost u odnosu na prvu verziju! */
24
25 /* Logaritamsko resenje je zasnovano na cinjenicama: -  $x^0 = 1$ ; -
26     $x^k = x * (x^2)^{(k/2)}$ , za neparno k -  $x^k = (x^2)^{(k/2)}$ ,
27    za parno k
28
29    Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
30    doslo do rezultata, i stoga je efikasnije. */
31 int stepen2(int x, int k)
32 {
33     // printf("Racunam stepen2 (%d, %d)\n",x,k);
34     if (k == 0)
35         return 1;
36
37     /* Ako je stepen paran */
38     if ((k % 2) == 0)
39         return stepen2(x * x, k / 2);
40     /* Inace (ukoliko je stepen neparan) */
41     return x * stepen2(x * x, k / 2);
42 }
43
44 int main()
45 {
46     int x, k;
47     scanf("%d%d", &x, &k);
48
49     printf("%d", stepen(2, 10));
50     // printf("\n-----\n");
51     // printf("%d\n",stepen2(2,10));
52     return 0;
53 }
54 }
```

### Rešenje 1.18

---

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 100
5
   /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
7    (posrednu) rekurziju. */

9   /* Deklaracija funkcije neparan mora da bude navedena jer se ta
   funkcija koristi u telu funkcije paran, tj. koristi se pre
11  svoje definicije. Funkcija je mogla biti deklarirana i u telu
   funkcije paran. */
13
   unsigned neparan(unsigned n);
15
   /* Funkcija vraca 1 ako broj n ima paran broj cifara inace
17  vraca 0. */
   unsigned paran(unsigned n)
19 {
   if (n >= 0 && n <= 9)
21     return 0;
   else
23     return neparan(n / 10);
   }
25
   /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace
27  vraca 0. */
   unsigned neparan(unsigned n)
29 {
   if (n >= 0 && n <= 9)
31     return 1;
   else
33     return paran(n / 10);
   }
35
   /* Glavna funkcija za testiranje */
37 int main()
   {
39     int n;
   scanf("%d", &n);
41     printf("Uneti broj ima %s paran broj cifara\n",
           (paran(n) == 1 ? "" : "ne"));
43     return 0;
   }

```

### Rešenje 1.19

```

   #include <stdio.h>
2   /* Repno-rekurzivna (eng. tail recursive) je ona funkcija
   Cije se telo završava rekurzivnim pozivom, pri čemu taj
4   rekurzivni poziv ne učestvuje u nekom izrazu.

```

```
6      Kod ovih funkcija se po završetku za tekuci rekurzivni
8      poziv umesto skoka na adresu povratka skace na adresu
      povratka za prethodni poziv, odnosno za poziv na manjoj
      dubini. Time se stedi i prostor i vreme.

10
      Ovakve funkcije se mogu lako zameniti odgovarajucom
12      iterativnom funkcijom, cime se smanjuje prostorna
      slozenost algoritma. */
14      /* Pomoćna funkcija koja izracunava n! * result. Koristi
      repnu rekurziju. */
16      /* Result je argument u kom cemo akumulirati do tada
      izracunatu vrednost faktoriijela. Kada završimo, tj. kada
18      dodjemo do izlaza iz rekurzije potrebno je da vratimo
      result. */
20 int faktoriijelRepna(int n, int result)
21 {
22     if (n == 0)
23         return result;
24
25     return faktoriijelRepna(n - 1, n * result);
26 }

28 /* Sada zelimo da se oslobodimo repne rekurzije koja postoji u
      funkciji faktoriijelRepna, koristeći algoritam sa predavanja.

30
      Najpre cemo vrednost argumenta funkcije postaviti na vrednost
32      koja bi se prosledjivala rekurzivnom pozivu i pomocu goto
      naredbe vratiti se na pocetak tela funkcije. */

34
35 int faktoriijelRepna_v1(int n, int result)
36 {
37     pocetak:
38     if (n == 0)
39         return result;
40
41     result = n * result;
42     n = n - 1;
43     goto pocetak;
44 }

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

55     return result;
56 }
```

```

}
58
/* Nasim gore navedenim funkcijama pored n, mora da se salje i 1
60 za vrednost drugog argumenta u kome ce se akumulirati
rezultat. Funkcija faktorijel(n) je ovde radi udobnosti
62 korisnika, jer je sasvim prirodno da za faktorijel zahteva
samo 1 parametar. Funkcija faktorijel izracunava n!, tako Sto
64 odgovarajucoj gore navedenoj funkciji koja zaista racuna
faktorijel, salje ispravne argumente i vraca rezultat koju
66 joj ta funkcija vrati. Za testiranje, zameniti u telu
funkcije faktorijel poziv faktorijelRepna sa pozivom
68 faktorijelRepna_v1, a zatim sa pozivom funkcije
faktorijelRepna_v2. */
70 int faktorijel(int n)
{
72     return faktorijelRepna(n, 1);
}
74
/* Test program */
76 int main()
{
78     int n;

80     printf("Unesite n (<= 12): ");
scanf("%d", &n);
82     printf("%d! = %d\n", n, faktorijel(n));

84     return 0;
}

```

## Rešenje 1.20

## Rešenje 1.21

## Rešenje 1.22

```

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

4  /*
    Ako je n==0, onda je suma(a,0) = 0 Ako je n>0, onda je
6  suma(a,n) = a[n-1] + suma(a,n-1) Suma celog niza je jednaka
sumi prvih n-1 elementa uvećenoj za poslednji element celog
8  niza. */
int sumaNiza(int *a, int n)
10 {
    /* Ne stavljam strogu jednakost n==0, za slucaj da korisnik
12    prilikom prvog poziva, posalje negativan broj za velicinu
niza. */

```

## 1 Uvodni zadaci

---

```
14     if (n <= 0)
15         return 0;
16
17     return a[n - 1] + sumaNiza(a, n - 1);
18 }
19
20 /*
21     n==0, suma(a,0) = 0 n >0, suma(a,n) = a[0]+suma(a+1,n-1) Suma
22     celog niza je jednaka zbiru prvog elementa niza i sume
23     preostalih n-1 elementa. */
24 int sumaNiza2(int *a, int n)
25 {
26     if (n <= 0)
27         return 0;
28
29     return a[0] + sumaNiza2(a + 1, n - 1);
30 }
31
32 int main()
33 {
34     int a[MAX_DIM];
35     int n, i = 0;
36
37     /* Ucitavamo broj elemenata niza */
38     scanf("%d", &n);
39
40     /* Ucitavamo n elemenata niza. */
41     for (i = 0; i < n; i++)
42         scanf("%d", &a[i]);
43
44     printf("Suma elemenata je %d\n", sumaNiza(a, n));
45     // printf("Suma elemenata je %d\n", sumaNiza2(a, n));
46
47     return 0;
48 }
```

### Rešenje 1.23

```
1 #include <stdio.h>
2 #define MAX_DIM 256
3
4 /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza
5    niz dimenzije n */
6 int maksimum_niza(int niz[], int n)
7 {
8     /* Izlazak iz rekurziije: ako je niz dimenzije jedan, najveći
9        je ujedno i jedini element niza */
10    if (n == 1)
11        return niz[0];
12
13    /* Rešavamo problem manje dimenzije */
```

```

14     int max = maksimum_niza(niz, n - 1);

16     /* Ako nam je poznato resenje problema dimenzije n-1, resavamo
       problem dimenzije n */
18     return niz[n - 1] > max ? niz[n - 1] : max;
19 }

20 int main()
21 {
22     int brojevi[MAX_DIM];
23     int n;

24     /* Sve dok ne dodjemo do kraja ulaza, učitavamo brojeve u niz;
       i predstavlja indeks tekućeg broja. */
26     int i = 0;
27     while (scanf("%d", &brojevi[i]) != EOF) {
28         i++;
29     }
30     n = i;

31     /* Stampamo maksimum unetog niza brojeva */
32     printf("%d\n", maksimum_niza(brojevi, n));
33     return 0;
34 }

```

### Rešenje 1.24

```

#include <stdio.h>
#define MAX_DIM 256

4 int skalarno(int a[], int b[], int n)
5 {
6     /* Izlazak iz rekursije */
7     if (n == 0)
8         return 0;

10     /* Na osnovu rešenja problema dimenzije n-1, resavamo problem
       dimenzije n */
12     else
13         return a[n - 1] * b[n - 1] + skalarno(a, b, n - 1);
14 }

16 int main()
17 {
18     int i, a[MAX_DIM], b[MAX_DIM], n;

20     /* Unosimo dimenziju nizova, */
21     scanf("%d", &n);

22     /* a zatim i same nizove. */
23     for (i = 0; i < n; i++)

```

## 1 Uvodni zadaci

---

```
        scanf("%d", &a[i]);
26
    for (i = 0; i < n; i++)
28        scanf("%d", &b[i]);

30    /* Ispisujemo rezultat skalarnog proizvoda dva učitana niza. */
    printf("%d\n", skalarno(a, b, n));

32    return 0;
34 }
```

### Rešenje 1.25

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

4 int br_pojave(int x, int a[], int n)
{
6     /* Izlazak iz rekurzije */
    if (n == 1)
8         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
20     /* UCitavamo broj koji se traži */
    scanf("%d", &x);

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

28     n = i;

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

### 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 rekurzije */
    if (n < 3)
8         return 0;

10     else
        return ((a[n - 3] == x) && (a[n - 2] == y)
12             && (a[n - 1] == z))
            || tri_uzastopna_clana(x, y, z, a, n - 1);
14 }

16 int main()
{
18     int x, y, z, a[MAX_DIM];
    int n;

20     /* Ucitavaju se tri cela broja za koje se ispituje da li su
22         uzastopni clanovi niza */
    scanf("%d%d%d", &x, &y, &z);

24     /* Sve dok ne dodjemo do kraja ulaza, ucitavamo brojeve u niz */
    int i = 0;
    while (scanf("%d", &a[i]) != EOF) {
26         i++;
28     }
    n = i;

30     if (tri_uzastopna_clana(x, y, z, a, i))
        printf("da\n");
32     else
        printf("ne\n");
34     return 0;
36 }
38

```

### Rešenje 1.27

```

#include <stdio.h>
2
3 /* funkcija koja broji bitove svog argumenta */
4 /*
5     ako je x ==0, onda je count(x) = 0 inace count(x) =
6     najvisi_bit +count(x<<1)
7
8     Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
9     siftovanjem sa desne strane uvek dopisuju 0, argument x ce u

```

## 1 Uvodni zadaci

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

### Rešenje 1.29

```
#include<stdio.h>
2
3 /* Rekurzivna funkcija za odredjivanje najvece heksadekadne
4  cifre u broju */
5 int max_oktalna_cifra(unsigned x)
6 {
7     /* izlazak iz rekurzije */
8     if (x == 0)
9         return 0;
10    /* Odredjivanje poslednje heksadekadne cifre u broju */
```

```

12     int poslednja_cifra = x & 7;
    /* Odredjivanje maksimalne oktalne cifre u broju kada se iz
       njega izbrise poslednja oktalna cifra */
14     int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
    return poslednja_cifra >
16         max_bez_poslednje_cifre ? poslednja_cifra :
           max_bez_poslednje_cifre;
18 }

20 int main()
21 {
22     unsigned x;
23     scanf("%u", &x);
24     printf("%d\n", max_oktalna_cifra(x));
25     return 0;
26 }

```

### Rešenje 1.30

```

#include<stdio.h>

2

4 /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u
   broju */
6 int max_heksadekadna_cifra(unsigned x)
7 {
8     /* Izlazak iz rekurzije */
9     if (x == 0)
10         return 0;
11     /* Odredjivanje poslednje heksadekadne cifre u broju */
12     int poslednja_cifra = x & 15;
13     /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
       njega izbrise poslednja heksadekadna cifra */
14     int max_bez_poslednje_cifre = max_heksadekadna_cifra(x >> 4);
15     return poslednja_cifra >
16         max_bez_poslednje_cifre ? poslednja_cifra :
           max_bez_poslednje_cifre;
18 }

20 int main()
21 {
22     unsigned x;
23     scanf("%u", &x);
24     printf("%d\n", max_heksadekadna_cifra(x));
25     return 0;
26 }

```

### Rešenje 1.31

```
1  #include<stdio.h>
2  #include<string.h>
3  /* niska moze imati najviše 32 karaktera + 1 za terminalnu nulu */
4  #define MAX_DIM 33
5
6  int palindrom(char s[], int n)
7  {
8      if ((n == 1) || (n == 0))
9          return 1;
10     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
11 }
12
13 int main()
14 {
15     char s[MAX_DIM];
16     int n;
17
18     /* Ucitavamo nisku sa ulaza */
19     scanf("%s", s);
20
21     /* Odredjujemo duzinu niske */
22     n = strlen(s);
23
24     /* Ispisujemo na izlazu poruku da li je niska palindrom ili
25        nije */
26     if (palindrom(s, n))
27         printf("da\n");
28     else
29         printf("ne\n");
30
31     return 0;
32 }
```

### Rešenje 1.32

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_DUZINA_NIZA 50
4
5  void ispisiNiz(int a[], int n)
6  {
7      int i;
8
9      for (i = 1; i <= n; i++)
10         printf("%d ", a[i]);
11     printf("\n");
12 }
13
14 /* Funkcija proverava da li se x vec nalazi u permutaciji na
15    prethodnih 1...n mesta */
```

```

17 int koriscen(int a[], int n, int x)
18 {
19     int i;
20     for (i = 1; i <= n; i++)
21         if (a[i] == x)
22             return 1;
23
24     return 0;
25 }
26
27 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[]
28    je niz u koji smesta permutacije m - oznacava da se na m-tu
29    poziciju u permutaciji smesta jedan od preostalih celih
30    brojeva n- je velicina skupa koji se permutuje Funkciju
31    pozivamo sa argumentom m=1 jer krecemo da formiramo
32    permutaciju od 1. pozicije i nikada ne koristimo a[0]. */
33 void permutacija(int a[], int m, int n)
34 {
35     int i;
36
37     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti
38        broj premasila velicinu skupa, onda se svi brojevi vec
39        nalaze u permutaciji i ispisujemo permutaciju. */
40     if (m > n) {
41         ispisiNiz(a, n);
42         return;
43     }
44
45     /* Ideja: pronalazimo prvi broj koji mozemo da postavimo na
46        m-to mesto u nizu (broj koji se do sada nije pojavio u
47        permutaciji). Zatim, rekurzivno pronalazimo one permutacije
48        koje odgovaraju ovako postavljenom pocetku permutacije.
49        Kada to završimo, proveravamo da li postoji jos neki broj
50        koji moze da se stavi na m-to mesto u nizu (to se radi u
51        petlji). Ako ne postoji, funkcija je završila sa radom.
52        Ukoliko takav broj postoji, onda ponovo pozivamo rekurzivno
53        pronalazenje odgovarajucih permutacija, ali sada sa
54        drugacije postavljenim prefiksom. */
55
56     for (i = 1; i <= n; i++) {
57         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do
58            m-1 pozicije, onda ga stavljamo na poziciju m i pozivamo
59            funkciju da napravi permutaciju za jedan vece duzine, tj.
60            m+1. Inace, nastavljamo dalje, trazeci broj koji se nije
61            pojavio do sada u permutaciji. */
62         if (!koriscen(a, m - 1, i)) {
63             a[m] = i;
64             /* Pozivamo ponovo funkciju da dopuni ostatak permutacije
65                posle upisivanja i na poziciju m. */
66             permutacija(a, m + 1, n);
67         }
68     }
69 }

```

```
    }
69 }

71 int main(void)
72 {
73     int n;
74     int a[MAX_DUZINA_NIZA];

75     printf("Unesite duzinu permutacije: ");
76     scanf("%d", &n);
77     if (n < 0 || n >= MAX_DUZINA_NIZA) {
78         fprintf(stderr,
79             "Duzina permutacije mora biti broj veci od 0 i manji od %
80             d!\n",
81             MAX_DUZINA_NIZA);
82         exit(EXIT_FAILURE);
83     }

84     permutacija(a, 1, n);

85     exit(EXIT_SUCCESS);
86 }
87 }
```

### Rešenje 1.33

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Rekurzivna funkcija za racunanje binomnog koeficijenta. */
5  /* ako je k=0 ili k=n, onda je binomni koeficijent 0 ako je k
6     izmedju 0 i n, onda je bk(n,k) = bk(n-1,k-1) + bk(n-1,k) */
7  int binomniKoeficijent(int n, int k)
8  {
9      return (0 < k
10             && k < n) ? binomniKoeficijent(n - 1,
11                                     k - 1) +
12             binomniKoeficijent(n - 1, k) : 1;
13 }

14 /* Iterativno izracunavanje datog binomnog koeficijenta.

15
16
17     int binomniKoeficijent (int n, int k) { int i, j, b; for
18         (b=i=1, j=n; i<=k; b=b*j--/i++); return b; }
19
20 */
21
22 /* Prostim opažanjem se uocava da se svaki element n-te
23     hipotenuze (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1
24     hipotenuze. Uz pomenute dve nove ivicne jedinice lako se
25     zakljucuje da ce suma elementa n-te hipotenuze biti tacno 2
26     puta veca. */
```

```
27 int sumaElemenataHipotenuze(int n)
28 {
29     return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
30 }
31
32 int main()
33 {
34     int n, k, i, d;
35
36     scanf("%d %d", &d, &n);
37
38     /* Ispisivanje Paskalovog trougla */
39     putchar('\n');
40     for (n = 0; n <= d; n++) {
41         for (i = 0; i < d - n; i++)
42             printf(" ");
43         for (k = 0; k <= n; k++)
44             printf("%4d", binomniKoeficijent(n, k));
45         putchar('\n');
46     }
47
48     if (n < 0) {
49         fprintf(stderr,
50             "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
51         );
52         exit(EXIT_FAILURE);
53     }
54     printf("%d\n", sumaElemenataHipotenuze(n));
55     exit(EXIT_SUCCESS);
56 }
57 }
```





## Glava 2

# Pokazivači

### 2.1 Pokazivačka aritmetika

**Zadatak 2.1** Za dati celobrojni niz dimenzije  $n$ , napisati funkciju koja obrće njegove elemente:

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju niza  $n$  ( $0 < n \leq 100$ ), a zatim elemente niza. Pozvati funkciju koja obrće njegove elemente korišćenjem indeksne sintakse i prikazati sadržaj niza. Nakon toga pozvati funkciju koja obrće njegove elemente korišćenjem pokazivačke sintakse i prikazati sadržaj niza.

#### *Primer 1*

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 3
Unesite elemente niza:
1 -2 3
Nakon obrtanja elemenata, niz je:
3 -2 1
Nakon ponovnog obrtanja elemenata,
niz je:
3 -2 1
```

#### *Primer 2*

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 0
Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.1]

**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 element niza.
- (d) Napisati funkciju `max_element` koja izračunava najveći element 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.

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 3
Unesite elemente niza:
-1.1 2.2 3.3
Zbir elemenata niza je 4.400.
Proizvod elemenata niza je -7.986
Minimalni element niza je -1.100
Maksimalni element niza je 3.300
```

### Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 5
Unesite elemente niza:
1.2 3.4 0.0 -5.4 2.1
Zbir elemenata niza je 1.300.
Proizvod elemenata niza je -0.000.
Minimalni element niza je -5.400.
Maksimalni element niza je 3.400.
```

[Rešenje 2.2]

**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  $n$  ( $0 < n \leq 100$ ) celobrojnog niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom.

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 5
Unesite elemente niza:
1 2 3 4 5
Transformisan niz je:
2 3 3 3 4
```

### Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 4
Unesite elemente niza:
4 -3 2 -1
Transformisan niz je:
5 -2 1 -2
```

### Primer 3

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 0
Greska: neodgovarajuca dimenzija niza.
```

### Primer 4

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 101
Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.3]

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

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

Od korisnika sa ulaza tražiti da izabere da li koje od ova dva rešenja treba koristiti prilikom ispisa.

#### Primer 1

```
Poziv: ./a.out prvi 2. treci -4

INTERAKCIJA PROGRAMA:
Broj prihvacenih argumenata komandne linije je 5.
Kako zelite da ispisete argumente, koriscenjem
indeksne ili pokazivacke sintakse (I ili P)? I
Argumenti komandne linije su:
0 ./a.out
1 prvi
2 2.
3 treci
4 -4
Pocetna slova argumenata komandne linije su:
. p 2 t -
```

#### Primer 2

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Broj prihvacenih argumenata komandne linije je 1.
Kako zelite da ispisete argumente, koriscenjem
indeksne ili pokazivacke sintakse (I ili P)? P
Argumenti komandne linije su:
0 ./a.out
Pocetna slova argumenata komandne linije su:
.
```

[Rešenje 2.4]

**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.

Milena: nedostaje poziv u prvom primeru

### Primer 1

```
INTERAKCIJA PROGRAMA:
  Broj argumenata komandne linije
  koji su palindromi je 4.
```

### Primer 2

```
POZIV: ./a.out a b 11 212
INTERAKCIJA PROGRAMA:
  Broj argumenata komandne linije koji
  koji su palindromi je 4.
```

### Primer 3

```
POZIV: ./a.out
INTERAKCIJA PROGRAMA:
  Broj argumenata komandne linije koji
  koji su palindromi je 0.
```

[Rešenje 2.5]

**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.

### Primer 1

```
POZIV: ./a.out ulaz.txt 1
ULAZNA DATOTEKA (ULAZ.TXT)
  Ovo je sadržaj datoteke i u njoj ima
  reci koje imaju 1 karakter
INTERAKCIJA PROGRAMA:
  Broj reci ciji je broj karaktera 1 je 3.
```

### Primer 2

```
POZIV: ./a.out ulaz.txt
ULAZNA DATOTEKA (ULAZ.TXT)
  Ovo je sadržaj datoteke i u njoj ima
  reci koje imaju 1 karakter
INTERAKCIJA PROGRAMA:
  Greska: Nedovoljan broj argumenata
  komandne linije.
  Program se poziva sa
  ./a.out ime_dat br_karaktera.
```

### Primer 3

```
POZIV: ./a.out ulaz.txt 2
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA PROGRAMA:
  Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

[Rešenje 2.6]

**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.

*Primer 1*

```
Poziv: ./a.out ulaz.txt ke -s
ULAZNA DATOTEKA (ULAZ.TXT)
Ovo je sadrzaj datoteke i u njoj ima reci
koje se završavaju na ke
INTERAKCIJA PROGRAMA:
Broj reci koje se završavaju na ke je 2.
```

*Primer 2*

```
Poziv: ./a.out ulaz.txt sa -p
ULAZNA DATOTEKA (ULAZ.TXT)
Ovo je sadrzaj datoteke i u njoj ima reci
koje počinju sa sa
INTERAKCIJA PROGRAMA:
Broj reci koje počinju na sa je 3.
```

*Primer 3*

```
Poziv: ./a.out ulaz.txt sa -p
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA PROGRAMA:
Greska: Neuspesno otvaranje
datoteke ulaz.txt.
```

*Primer 4*

```
Poziv: ./a.out ulaz.txt
ULAZNA DATOTEKA (ULAZ.TXT)
Ne postoji
INTERAKCIJA PROGRAMA:
Greska: Nedovoljan broj argumenata
komandne linije.
Program se poziva sa
./a.out ime_dat suf/pref -s/-p.
```

[Rešenje 2.7]

## 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 dimanziju kvadratne matrice  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrice.

## 2 Pokazivači

---

Na standardni izlaz ispisati učitanu matricu a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

```
Test 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 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 dimanziju 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 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.

- Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.
- Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).
- Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).
- 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 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.

Milena: Izbegavala bih komentare koji ulaze u kod i na taj način narušavaju citljivost koda, kao što je to npr u funkciji `indeks_min` i `indeks_max`. Resenje 2.15 - izbacila bih napomenu iz komentara. Slično mi se čini i za zadatak 2.17. Zadatak 2.17 - čini mi se da je resenje bez koriscenja bibliotekskih funkcija visak? Zadatak 2.19 — izvuci komentare za učitaj i ispiši ispred funkcija, umesto što su unutar funkcija. Zadatak 2.21 - čini mi se da komentari unutar funkcije izmeni narušavaju citljivost koda. Resenje 2.26 — izbaciti nasa slova iz komentara, izbaciti možda napomenu sa početka jer je suvisna

*Test Test 1*

```
Poziv: ./a.out 3
Ulaz:  1 2 3
        -4 -5 -6
        7 8 9
Izlaz:  7 2 2 3
```

*Test Test 2*

```
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
```

*Test Test 3*

```
Poziv: ./a.out
Izlaz: Greska: Nedovoljan broj argumenata komandne linije.
        Program se poziva sa ./a.out dim_matrice.
```

**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 Test 1</i>	<i>Test 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>

*Test Test 3*

```

|| Ulaz:  33
|| Izlaz: Greska: neodgovarajuca dimenzija matrice.
  
```

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

<i>Test Test 1</i>	<i>Test 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>

*Test Test 3*

```

|| Ulaz:  11 4
|| Izlaz: Greska: neodgovarajuca dimenzija matrice.
  
```

**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 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.

```
Test Test 1                                Test Test 2
||                                           ||
|| Ulaz: 3                                || Ulaz: -1
||       1 -2 3                            || Izlaz: malloc(): neuspela alokacija memorije.
|| Izlaz: 3 -2 1
```

**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.

```
Test Test 1                                Test Test 2
||                                           ||
|| Ulaz: 1 -2 3 -4 0                        || Ulaz: 0
|| Izlaz: -4 3 -2 1                        || Izlaz:
```

**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.

*Test Test 1*

```
|| Ulaz:  Jedan Dva
|| Izlaz: JedanDva
```

**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.

*Test Test 1*

```
|| Ulaz:  2 3
||         1.2 2.3 3.4
||         4.5 5.6 6.7
|| Izlaz: 6.80
```

**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 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 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 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 elementa 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 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 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):

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

## Test Test 1

```

Ulaz:  tan 1.570795 10000
Izlaz: -10134.5

```

## Test 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>
   #include <stdlib.h>
3
   #define MAX 100
5
   /* Funkcija obrce elemente niza koriscenjem indekse sintakse */
7  void obrni_niz_v1(int a[], int n)
   {
9     int i, j;

11    for (i = 0, j = n - 1; i < j; i++, j--) {
        int t = a[i];
13        a[i] = a[j];
        a[j] = t;
15    }
   }
17
   /* Funkcija obrce elemente niza koriscenjem pokazivacke sintakse */
19  void obrni_niz_v2(int *a, int n)
   {
21     /* Pokazivaci na elemente niza */
        int *prvi, *poslednji;

23
        /* Vrsi se obrtanje niza */
25    for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
        int t = *prvi;

27
        /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
29         vrednost koja se nalazi na adresi na koju pokazuje
        pokazivac "poslednji". Nakon toga se pokazivac "prvi"
31         uvecava za jedan sto za posledicu ima da "prvi" pokazuje
        na sledeci element u nizu */
33        *prvi++ = *poslednji;

35
        /* Vrednost promenljive "t" se postavlja na adresu na koju
        pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
37         umanjuje za jedan, sto za posledicu ima da pokazivac
        "poslednji" sada pokazuje na element koji mu prethodi u
39         nizu */

```

```

    *poslednji-- = t;
41 }

43 /* Drugi nacin za obrtanje niza */
44 /*
45 for (prvi = a, poslednji = a + n - 1;
46      prvi < poslednji; prvi++, poslednji--) {
47     int t = *prvi;
48     *prvi = *poslednji;
49     *poslednji = t;
50 }
51 */
52 }

53 int main()
54 {
55     /* Deklarise se niz od najvise MAX elemenata */
56     int a[MAX];

57     /* Broj elemenata niza a */
58     int n;

59     /* Pokazivac na elemente niza */
60     int *p;

61     printf("Unesite dimenziju niza: ");
62     scanf("%d", &n);

63     /* Proverava se da li je doslo do prekoračenja ograničenja
64        dimenzije */
65     if (n <= 0 || n > MAX) {
66         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
67         exit(EXIT_FAILURE);
68     }

69     printf("Unesite elemente niza:\n");
70     for (p = a; p - a < n; p++)
71         scanf("%d", p);

72     obrni_niz_v1(a, n);

73     printf("Nakon obrtanja elemenata, niz je:\n");

74     for (p = a; p - a < n; p++)
75         printf("%d ", *p);
76     printf("\n");

77     obrni_niz_v2(a, n);

78     printf("Nakon ponovnog obrtanja elemenata, niz je:\n");

79     for (p = a; p - a < n; p++)
```



```
    printf("%d ", *p);  
93 printf("\n");  
  
95 return 0;  
}
```

## Rešenje 2.2

```
#include <stdio.h>  
2 #include <stdlib.h>  
  
4 #define MAX 100  
  
6 /* Funkcija izracunava zbir elemenata niza */  
double zbir(double *a, int n)  
8 {  
    double s = 0;  
10    int i;  
  
12    for (i = 0; i < n; s += a[i++]);  
  
14    return s;  
}  
16  
/* Funkcija izracunava proizvod elemenata niza */  
18 double proizvod(double a[], int n)  
{  
20    double p = 1;  
  
22    for (; n; n--)  
        p *= *a++;  
24  
    return p;  
26 }  
  
28 /* Funkcija izracunava minimalni element niza */  
double min(double *a, int n)  
30 {  
    /* Na pocetku, minimalni element je prvi element */  
32    double min = a[0];  
    int i;  
  
34    /* Ispituje se da li se medju ostalim elementima niza nalazi  
    minimalni */  
36    for (i = 1; i < n; i++)  
        if (a[i] < min)  
            min = a[i];  
40  
    return min;  
42 }
```

```
44 /* Funkcija izracunava maksimalni element niza */
double max(double *a, int n)
46 {
    /* Na pocetku, maksimalni element je prvi element */
48     double max = *a;

50     /* Ispituje se da li se medju ostalim elementima niza nalazi
        maksimalni */
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     printf("Unesite dimenziju niza: ");
66     scanf("%d", &n);

68     /* Proverava se da li je doslo do prekoračenja ograničenja
        dimenzije */
70     if (n <= 0 || n > MAX) {
        fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72         exit(EXIT_FAILURE);
    }

74     printf("Unesite elemente niza:\n");
76     for (i = 0; i < n; i++)
        scanf("%lf", a + i);

78     /* Vrsi se testiranje definisanih funkcija */
80     printf("Zbir elemenata niza je %5.3f.\n", zbir(a, n));
    printf("Proizvod elemenata niza je %5.3f.\n", proizvod(a, n));
82     printf("Minimalni element niza je %5.3f.\n", min(a, n));
    printf("Maksimalni element niza je %5.3f.\n", max(a, n));

84     return 0;
86 }
```

### Rešenje 2.3

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

5 /* Funkcija povecava za jedan sve elemente u prvoj polovini niza
```

```
7      a smanjuje za jedan sve elemente u drugoj polovini niza.  
      Ukoliko niz ima neparan broj elemenata, srednji element ostaje  
      nepromenjen */  
9 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 pokazivac  
17            prvi */  
18         (*prvi)++;  
19  
20         /* Pokazivac prvi se pomera na sledeci element */  
21         prvi++;  
22  
23         /* Smanjuje se vrednost elementa na koji pokazuje pokazivac  
24            poslednji */  
25         (*poslednji)--;  
26  
27         /* Pokazivac poslednji se pomera na prethodni element */  
28         poslednji--;  
29     }  
30  
31     /* Drugi nacin */  
32     while (prvi < poslednji) {  
33         (*prvi++)++;  
34         (*poslednji--)--;  
35     }  
36 }  
37  
38 int main()  
39 {  
40     int a[MAX];  
41     int n;  
42     int *p;  
43  
44     printf("Unesite dimenziju niza: ");  
45     scanf("%d", &n);  
46  
47     /* Proverava se da li je doslo do prekoracenja ogranicenja  
48        dimenzije */  
49     if (n <= 0 || n > MAX) {  
50         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");  
51         exit(EXIT_FAILURE);  
52     }  
53  
54     printf("Unesite elemente niza:\n");  
55     for (p = a; p - a < n; p++)  
56         scanf("%d", p);  
57 }
```

## 2 Pokazivači

```
    povecaj_smanji(a, n);
59
    printf("Transformisan niz je:\n");
61    for (p = a; p - a < n; p++)
        printf("%d ", *p);
63    printf("\n");
65    return 0;
}
```

### Rešenje 2.4

```
#include <stdio.h>
2
int main(int argc, char *argv[])
4 {
    int i;
    char tip_ispisa;
6
    printf("Broj prihvacenih argumenata komandne linije je %d.\n", argc
    );
8
    printf("Kako zelite da ispisete argumente, ");
    printf("koriscenjem indeksne ili pokazivacke sintakse (I ili P)? ")
    ;
10    scanf("%c", &tip_ispisa);
12
    printf("Argumenti komandne linije su:\n");
    if (tip_ispisa == 'I') {
14        /* Ispisuju se argumenti komandne linije koriscenjem indeksne
            sintakse */
16        for (i = 0; i < argc; i++)
            printf("%d %s\n", i, argv[i]);
18    } else if (tip_ispisa == 'P') {
        /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
            sintakse */
20        i = argc;
        for (; argc > 0; argc--)
            printf("%d %s\n", i - argc, *argv++);
22
        /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje
            na polje u memoriji koje se nalazi iza poslednjeg argumenta
            komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
            broja argumenta komandne linije to sada moze ponovo da se
            postavi "argv" da pokazuje na nulti argument komandne linije */
24        argv = argv - i;
        argc = i;
26    }
28
    printf("Pocetna slova argumenata komandne linije su:\n");
    if (tip_ispisa == 'I') {
30
32
34
36
```

```

38     /* koristeći indeksnu sintaksu */
    for (i = 0; i < argc; i++)
39         printf("%c ", argv[i][0]);
    printf("\n");
42 } else if (tip_ispisa == 'P'){
    /* koristeći pokazivačku sintaksu */
44     for (i = 0; i < argc; i++)
        printf("%c ", **argv++);
46     printf("\n");
    }
48
50     return 0;
}

```

### Rešenje 2.5

```

1  #include<stdio.h>
   #include<string.h>
3  #define MAX 100

5  /* Funkcija ispituje da li je niska palindrom */
   int palindrom(char *niska)
7  {
    int i, j;
9    for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
        if (*(niska + i) != *(niska + j))
11         return 0;
    return 1;
13 }

15 int main(int argc, char **argv)
   {
17     int i, n = 0;

19     /* Multi argument komandne linije je ime izvršnog programa */
    for (i = 1; i < argc; i++)
21         if (palindrom(*(argv + i)))
            n++;
23
    printf("Broj argumenata komandne linije koji su palindromi je %d.\n", n);
25     return 0;
   }

```

### Rešenje 2.6

```

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

```

```
4 #define MAX_KARAKTERA 100

6 /* Implementacija funkcija strlen() iz standardne biblioteke */
int duzina(char *s)
8 {
    int i;
10    for (i = 0; *(s + i); i++);
    return i;
12 }

14 int main(int argc, char **argv)
{
16     char rec[MAX_KARAKTERA];
    int br = 0, n;
18     FILE *in;

20     /* Ako korisnik nije uneo trazene argumente, prijavljuje se
       greska */
22     if (argc < 3) {
        printf("Greska: ");
24         printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_dat br_karaktera.\n",
26             argv[0]);
        exit(EXIT_FAILURE);
28     }

30     /* Otvara se datoteka sa imenom koje se zadaje kao prvi
       argument komandne linije. */
32     in = fopen(*(argv + 1), "r");
    if (in == NULL) {
34         fprintf(stderr, "Greska: ");
        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
36             argv[1]);
        exit(EXIT_FAILURE);
38     }

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

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

48     printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);

50     /* Zatvara se datoteka */
    fclose(in);
52     return 0;
}
```

## Rešenje 2.7

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Implementacija funkcije strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
8  {
9      int i;
10     for (i = 0; *(src + i); i++)
11         *(dest + i) = *(src + i);
12 }
13
14 /* Implementacija funkcije strcmp() iz standardne biblioteke */
15 int poredjenje_niski(char *s, char *t)
16 {
17     int i;
18     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 /* Implementacija funkcije strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
26 {
27     int i;
28     for (i = 0; *(s + i); i++);
29     return i;
30 }
31
32 /* Funkcija ispituje da li je niska zadata drugim argumentom
33    funkcije sufiks niske zadate prvi argumentom funkcije */
34 int sufiks_niske(char *niska, char *sufiks)
35 {
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 {
47     int i;
48     if (duzina_niske(prefiks) <= duzina_niske(niska)) {
49         for (i = 0; i < duzina_niske(prefiks); i++)
50             if (*(prefiks + i) != *(niska + i))
```

```
        return 0;
52     return 1;
    } else
54         return 0;
}

56
int main(int argc, char **argv)
58 {
    /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
60     greska */
    if (argc < 4) {
62         printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
64         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
            argv[0]);
66         exit(EXIT_FAILURE);
    }

68
    FILE *in;
70     int br = 0;
    char rec[MAX_KARAKTERA];

72
    in = fopen(*(argv + 1), "r");
74     if (in == NULL) {
        fprintf(stderr, "Greska: ");
76         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
            argv[1]);
78         exit(EXIT_FAILURE);
    }

80
    /* Provera se opcija kojom je pozvan program a zatim se
82     učitavaju reci iz datoteke i broji se koliko njih zadovoljava
        trazeni uslov */
84     if (!(poredjenje_niski(*(argv + 3), "-s"))) {
        while (fscanf(in, "%s", rec) != EOF)
86             br += sufiks_niske(rec, *(argv + 2));
        printf("Broj reci koje se završavaju na %s je %d.\n", *(argv + 2),
            br);
88     } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
        while (fscanf(in, "%s", rec) != EOF)
90             br += prefiks_niske(rec, *(argv + 2));
        printf("Broj reci koje počinju na %s je %d.\n", *(argv + 2), br);
92     }

94     fclose(in);
    return 0;
96 }
```

### Rešenje 2.8

---



```

1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4
5  #define MAX 100
6
7  /* Deklarisemo funkcije koje cemo kasnije da definisemo */
8  double euklidska_norma(int M[][MAX], int n);
9  int trag(int M[][MAX], int n);
10 int gornja_vandijagonalna_norma(int M[][MAX], int n);
11
12 int main()
13 {
14     int A[MAX][MAX];
15     int i, j, n;
16
17     /* Unosimo dimenziju kvadratne matrice */
18     scanf("%d", &n);
19
20     /* Proveravamo da li je prekoraceno ogranicenje */
21     if (n > MAX || n <= 0) {
22         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
23         fprintf(stderr, "matrice.\n");
24         exit(EXIT_FAILURE);
25     }
26
27     /* Popunjavamo vrstu po vrstu matrice */
28     for (i = 0; i < n; i++)
29         for (j = 0; j < n; j++)
30             scanf("%d", &A[i][j]);
31
32     /* Ispis elemenata matrice koriscenjem indeksne sintakse.
33        Ispis vrsimo vrstu po vrstu */
34     for (i = 0; i < n; i++) {
35         /* Ispisujemo elemente i-te vrste */
36         for (j = 0; j < n; j++)
37             printf("%d ", A[i][j]);
38         printf("\n");
39     }
40
41     /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
42        Kod ovako definisane matrice, elementi su uzastopno
43        smesteni u memoriju, kao na traci. To znaci da su svi
44        elementi prve vrste redom smesteni jedan iza drugog. Odmah
45        iza poslednjeg elementa prve vrste smesten je prvi element
46        druge vrste za kojim slede svi elementi te vrste i tako
47        dalje redom */
48     /*
49        for( i = 0; i<n; i++) { for ( j=0 ; j<n; j++) printf("%d ",
50        (*(A+i)+j)); printf("\n"); } */
51
52     int tr = trag(A, n);

```

```
    printf("trag = %d\n", tr);
54
    printf("euklidska norma = %.2f\n", euklidska_norma(A, n));
56
    printf("vandijagonalna norma = %d\n",
        gornja_vandijagonalna_norma(A, n));
58
    return 0;
60 }

62 /* Definisemo funkcije koju smo ranije deklarirali */

64 /* Funkcija izracunava trag matrice */
int trag(int M[][MAX], int n)
66 {
    int trag = 0, i;
68     for (i = 0; i < n; i++)
        trag += M[i][i];
70     return trag;
}

72
74 /* Funkcija izracunava euklidsku normu matrice */
double euklidska_norma(int M[][MAX], int n)
76 {
    double norma = 0.0;
    int i, j;
78
    for (i = 0; i < n; i++)
80         for (j = 0; j < n; j++)
            norma += M[i][j] * M[i][j];
82
    return sqrt(norma);
84 }

86 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
int gornja_vandijagonalna_norma(int M[][MAX], int n)
88 {
    int norma = 0;
    int i, j;
90
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++)
94             norma += abs(M[i][j]);
    }
96
    return norma;
98 }
```

### Rešenje 2.9

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

```
4 #define MAX 100

6 /* Funkcija ucitava elemente kvadratne matrice dimenzije n 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 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
   standardni izlaz */
18 void ispisi_matricu(int m[][MAX], int n)
20 {
     int i, j;

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

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

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

42     /* Prosla je provera jednakosti za sve parove elemenata koji
       su na istim pozicijama sto znaci da su matrice jednake */
44     return 1;
46 }

48 /* Funkcija izracunava zbir dve kvadratne matrice */
void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
50 {
     int i, j;

52     for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
54         
```

```
        c[i][j] = a[i][j] + b[i][j];
56 }

58 /* Funkcija izracunava proizvod dve kvadratne matice */
void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
60 {
    int i, j, k;
62
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
64             /* Mnozimo i-tu vrstu prve sa j-tom kolonom druge matrice */
            c[i][j] = 0;
            for (k = 0; k < n; k++)
66                 c[i][j] += a[i][k] * b[k][j];
68         }
70 }

72 int main()
{
74     /* Matrice ciji se elementi zadaju sa ulaza */
    int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
76
    /* Matrice zbira i proizvoda */
78     int zbir[MAX][MAX], proizvod[MAX][MAX];

80     /* Dimenzija matrica */
    int n;
82     int i, j;

84     /* Ucitavamo dimenziju kvadratnih matrica i proveravamo njenu
        korektnost */
86     scanf("%d", &n);

88     /* Proveravamo da li je prekoraceno ogranicenje */
    if (n > MAX || n <= 0) {
90         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
        fprintf(stderr, "matrica.\n");
92         exit(EXIT_FAILURE);
    }
94

    /* Ucitavamo matrice */
96     ucitaj_matricu(a, n);
    ucitaj_matricu(b, n);
98

    /* Izracunavamo zbir i proizvod matrica */
100     saberi(a, b, zbir, n);
    pomnozi(a, b, proizvod, n);
102

    /* Ispisujemo rezultat */
104     if (jednake_matrice(a, b, n) == 1)
        printf("da\n");
106     else
```

```

    printf("ne\n");
108
    printf("Zbir matrica je:\n");
110    ispisi_matricu(zbir, n);

    printf("Proizvod matrica je:\n");
112    ispisi_matricu(proizvod, n);
114
    return 0;
116 }

```

### Rešenje 2.10

```

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

4  #define MAX 64

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

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

20     return 1;
22 }

24 /* Funkcija odredjuje refleksivno zatvorenje zadate relacije.
   Ono je odredjeno matricom koja sadrzi sve elemente polazne
26 matrice dopunjene jedinicama na glavnoj dijagonali */
void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
28 {
    int i, j;

30     /* Prepisujemo vrednosti elemenata matrice pocetne matrice */
32     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 */
    for (i = 0; i < n; i++)
38         zatvorenje[i][i] = 1;

```

```

    }
40
41 /* Funkcija proverava da li je relacija simetricna. Relacija je
42    simetricna ako za svaki par elemenata vazi: ako je element
43    "i" u relaciji sa elementom "j", onda je i element "j" u
44    relaciji sa elementom "i". Ovakve matrice su simetricne u
45    odnosu na glavnu dijagonalu */
46 int simetricnost(int m[][MAX], int n)
47 {
48     int i, j;
49
50     /* Obilazimo elemente ispod glavne dijagonale matrice i
51        uporedjujemo ih sa njima simetricnim elementima */
52     for (i = 0; i < n; i++)
53         for (j = 0; j < i; j++)
54             if (m[i][j] != m[j][i])
55                 return 0;
56
57     return 1;
58 }
59
60 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono
61    je odredjeno matricom koja sadrzi sve elemente polazne
62    matrice dopunjene tako da matrica postane simetricna u odnosu
63    na glavnu dijagonalu */
64 void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
65 {
66     int i, j;
67
68     /* Prepisujemo vrednosti elemenata matrice m */
69     for (i = 0; i < n; i++)
70         for (j = 0; j < n; j++)
71             zatvorenje[i][j] = m[i][j];
72
73     /* Odredjujemo simetricno zatvorenje matrice */
74     for (i = 0; i < n; i++)
75         for (j = 0; j < n; j++)
76             if (zatvorenje[i][j] == 1)
77                 zatvorenje[j][i] = 1;
78 }
79
80
81 /* Funkcija proverava da li je relacija tranzitivna. Relacija je
82    tranzitivna ako ispunjava sledece svojstvo: ako je element
83    "i" u relaciji sa elementom "j" i element "j" u relaciji sa
84    elementom "k", onda je i element "i" u relaciji sa elementom
85    "k" */
86 int tranzitivnost(int m[][MAX], int n)
87 {
88     int i, j, k;
89
90     for (i = 0; i < n; i++)
```

```

    for (j = 0; j < n; j++)
122     /* Pokusavamo da pronadjemo element koji narusava *
123        tranzitivnost */
124     for (k = 0; k < n; k++)
125         if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
126             return 0;
127
128     return 1;
129 }
130
131 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
132    relacije koriscenjem Varsalovog algoritma */
133 void tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
134 {
135     int i, j, k;
136
137     /* Kopiramo pocetnu matricu u matricu rezultata */
138     for (i = 0; i < n; i++)
139         for (j = 0; j < n; j++)
140             zatvorenje[i][j] = m[i][j];
141
142     /* Primenom Varsalovog algoritma odredjujemo
143        refleksivno-tranzitivno zatvorenje matrice */
144     for (k = 0; k < n; k++)
145         for (i = 0; i < n; i++)
146             for (j = 0; j < n; j++)
147                 if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
148                     && (zatvorenje[i][j] == 0))
149                     zatvorenje[i][j] = 1;
150 }
151
152 /* Funkcija ispisuje elemente matrice */
153 void pisi_matricu(int m[][MAX], int n)
154 {
155     int i, j;
156
157     for (i = 0; i < n; i++) {
158         for (j = 0; j < n; j++)
159             printf("%d ", m[i][j]);
160         printf("\n");
161     }
162 }
163
164 int main(int argc, char *argv[])
165 {
166     FILE *ulaz;
167     int m[MAX][MAX];
168     int pomocna[MAX][MAX];
169     int n, i, j, k;
170
171     /* Ako korisnik nije uneo trazene argumente, prijavljujemo

```

```
    gresku */
144 if (argc < 2) {
    printf("Greska: ");
146 printf("Nedovoljan broj argumenata komandne linije.\n");
    printf("Program se poziva sa %s ime_dat.\n", argv[0]);
148 exit(EXIT_FAILURE);
}

150 /* Otvaramo datoteku za citanje */
152 ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
154 fprintf(stderr, "Greska: ");
    fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
156 argv[1]);
    exit(EXIT_FAILURE);
158 }

160 /* Ucitavamo dimenziju matrice */
    fscanf(ulaz, "%d", &n);
162

    /* Proveravamo da li je prekoraceno ogranicenje */
164 if (n > MAX || n <= 0) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
166 fprintf(stderr, "matrice.\n");
    exit(EXIT_FAILURE);
168 }

170 /* Ucitavamo element po element matrice */
    for (i = 0; i < n; i++)
172     for (j = 0; j < n; j++)
        fscanf(ulaz, "%d", &m[i][j]);
174

    /* Ispisujemo trazene vrednosti */
176 printf("Refleksivnost: %s\n",
        refleksivnost(m, n) == 1 ? "da" : "ne");
178

    printf("Simetricnost: %s\n",
180 simetricnost(m, n) == 1 ? "da" : "ne");

    printf("Tranzitivnost: %s\n",
182 tranzitivnost(m, n) == 1 ? "da" : "ne");
184

    printf("Refleksivno zatvorenje:\n");
186 ref_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
188

    printf("Simetricno zatvorenje:\n");
190 sim_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
192

    printf("Refleksivno-tranzitivno zatvorenje:\n");
194 tran_zatvorenje(m, n, pomocna);
```



```

196     pisi_matricu(pomocna, n);
198     /* Zatvaramo datoteku */
198     fclose(ulaz);
200     return 0;
}

```

### 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 pocetnu 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 pocetnu 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 */

```

```

43 int indeks_max(int m[][MAX], int n)
44 {
45     int i, j;
46     /* Za maksimalni element uzimamo gornji levi ugao */
47     int max = m[0][0], indeks_vrstte = 0;
48
49     for (i = 0; i < n; i++)
50         for (j = 0; j < n; j++)
51             /* Ako je tekuci element manji od minimalnog */
52             if (m[i][j] > max) {
53                 /* cuvamo njegovu vrednost */
54                 max = m[i][j];
55                 /* i cuvamo indeks vrste u kojoj se nalazi */
56                 indeks_vrstte = i;
57             }
58     return indeks_vrstte;
59 }
60
61 /* Funkcija izracunava broj negativnih elemenata matrice */
62 int broj_negativnih(int m[][MAX], int n)
63 {
64     int i, j;
65
66     int broj_negativnih = 0;
67
68     for (i = 0; i < n; i++)
69         for (j = 0; j < n; j++)
70             if (m[i][j] < 0)
71                 broj_negativnih++;
72     return broj_negativnih;
73 }
74
75 int main(int argc, char *argv[])
76 {
77     int m[MAX][MAX];
78     int n;
79     int i, j;
80
81     /* Proveravamo broj argumenata komandne linije */
82     if (argc < 2) {
83         printf("Greska: ");
84         printf("Nedovoljan broj argumenata komandne linije.\n");
85         printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
86         exit(EXIT_FAILURE);
87     }
88
89     /* Ucitavamo vrednost dimenzije i proveravamo njenu korektnost */
90     n = atoi(argv[1]);
91
92     if (n > MAX || n <= 0) {
93         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");

```

```

    fprintf(stderr, "matrice.\n");
95     exit(EXIT_FAILURE);
}

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

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

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

111     /* Prekidamo izvršavanje programa */
112     return 0;
113 }

```

## Rešenje 2.12

```

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

4  #define MAX 32

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

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

```

```
30 /* Funkcija proverava da li je zadata matrica ortonormirana */
31 int ortonormirana(int m[][MAX], int n)
32 {
33     int i, j, k;
34     int proizvod;
35
36     /* Proveravamo uslov normiranosti, odnosno da li je proizvod
37        svake vrste matrice sa samom sobom jednak jedinici */
38     for (i = 0; i < n; i++) {
39
40         /* Izracunavamo skalarni proizvod vrste sa samom sobom */
41         proizvod = 0;
42
43         for (j = 0; j < n; j++)
44             proizvod += m[i][j] * m[i][j];
45
46         /* Ako proizvod bar jedne vrste nije jednak jedinici, odmah
47            zakljucujemo da matrica nije normirana */
48         if (proizvod != 1)
49             return 0;
50     }
51
52     /* Proveravamo uslov ortogonalnosti, odnosno da li je proizvod
53        dve bilo koje razlicite vrste matrice jednak nuli */
54     for (i = 0; i < n - 1; i++) {
55         for (j = i + 1; j < n; j++) {
56
57             /* Izracunavamo skalarni proizvod */
58             proizvod = 0;
59
60             for (k = 0; k < n; k++)
61                 proizvod += m[i][k] * m[j][k];
62
63             /* Ako proizvod dve bilo koje razlicite vrste nije jednak
64                nuli, odmah zakljucujemo da matrica nije ortogonalna */
65             if (proizvod != 0)
66                 return 0;
67         }
68     }
69
70     /* Ako su oba uslova ispunjena, vracamo jedinicu kao rezultat */
71     return 1;
72 }
73
74 int main()
75 {
76     int A[MAX][MAX];
77     int n;
78
79     /* Ucitavamo vrednost dimenzije i proveravamo njenu korektnost
80        */
```

```

scanf("%d", &n);
82
if (n > MAX || n <= 0) {
84     fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
    fprintf(stderr, "matrice.\n");
86     exit(EXIT_FAILURE);
}
88
/* Ucitavamo matricu */
90 ucitaj_matricu(A, n);

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

```

### Rešenje 2.13

```

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

#define MAX_V 10
#define MAX_K 10

4
/* Funkcija proverava da li su ispisani svi elementi iz matrice,
8     odnosno da li se nariusio prirodan poredak medju granicama */
int krajIspisa(int top, int bottom, int left, int right)
10 {
    return !(top <= bottom && left <= right);
12 }

14 /* Funkcija spiralno ispisuje elemente matrice */
void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16 {
    int i, j, top, bottom, left, right;
18
    top = left = 0;
    bottom = n - 1;
20     right = m - 1;
22
    while (!krajIspisa(top, bottom, left, right)) {
24         /* Ispisuje se prvi red */
        for (j = left; j <= right; j++)
26             printf("%d ", a[top][j]);
28
        /* Spustamo prvi red */
        top++;
    }
}

```

```
30     if (krajIspisa(top, bottom, left, right))
31         break;
32
33     for (i = top; i <= bottom; i++)
34         printf("%d ", a[i][right]);
35
36     /* Pomeramo desnu kolonu za naredni krug ispisa blize levom
37        kraju */
38     right--;
39
40     if (krajIspisa(top, bottom, left, right))
41         break;
42
43     /* Ispisujemo donju vrstu */
44     for (j = right; j >= left; j--)
45         printf("%d ", a[bottom][j]);
46
47     /* Podizemo donju vrstu za naredni krug ispisa */
48     bottom--;
49
50     if (krajIspisa(top, bottom, left, right))
51         break;
52
53     /* Ispisujemo prvu kolonu */
54     for (i = bottom; i >= top; i--)
55         printf("%d ", a[i][left]);
56
57     /* Pripremamo levu kolonu za naredni krug ispisa */
58     left++;
59 }
60 putchar('\n');
61 }
62
63 void ucitaj_matricu(int a[][MAX_K], int n, int m)
64 {
65     int i, j;
66
67     for (i = 0; i < n; i++)
68         for (j = 0; j < m; j++)
69             scanf("%d", &a[i][j]);
70 }
71
72 int main()
73 {
74     int a[MAX_V][MAX_K];
75     int m, n;
76
77     /* Ucitaj broj vrsta i broj kolona matrice */
78     scanf("%d", &n);
79     scanf("%d", &m);
80 }
```

```

82  if (n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
      fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
84  fprintf(stderr, "matrice.\n");
      exit(EXIT_FAILURE);
86  }

88  učitaj_matricu(a, n, m);
      ispisi_matricu_spiralno(a, n, m);
90
      return 0;
92 }

```

### Rešenje 2.14

### Rešenje 2.15

```

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

4  /* NAPOMENA: Primer demonstrira dinamičku alokaciju niza od n
   elementa. Dovoljno je alocirati n * sizeof(T) bajtova, gde
6  je T tip elemenata niza. Povratnu adresu malloc()-a treba
   pretvoriti iz void * u T *, kako bismo dobili pokazivac koji
8  pokazuje na prvi element niza tipa T. Na dalje se elementima
   može pristupiti na isti način kao da nam je dato ime niza
10  (koje se tako i ponasa - kao pokazivac na element tipa T koji
   je prvi u nizu) */
12  int main()
   {
14      int *p = NULL;
      int i, n;

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

22      /* Alociramo prostor za n celih brojeva */
24      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 sa
32      p[i] */

34      /* Unosimo elemente niza */

```

## 2 Pokazivači

---

```
36     for (i = 0; i < n; i++)
        scanf("%d", &p[i]);

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

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

44

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;

    /* Velicina alocirane memorije */
    10     int alocirano;

    /* Broj elemenata niza */
    12     int n;

    /* Broj koji se učitava sa ulaza */
    14     int x;
    16     int i;
    18     int *b = NULL;

    /* Inicijalizacija */
    20     alocirano = n = 0;

    /* Unosimo brojeve sa ulaza */
    22     scanf("%d", &x);

    /* Sve dok je procitani broj razlicit od nule... */
    24     while (x != 0) {
        26
        /* Ako broj ucitanih elemenata niza odgovara broju
           alociranih mesta, za smestanje novog elementa treba
           28         obezbediti dodatni prostor. Da se ne bi za svaki sledeci
           element pojedinačno alocirala memorija, prilikom
           30         alokacije se vrši rezervacija za još KORAK dodatnih mesta
           32         za buduće elemente */
        34    }
```



```

36  if (n == alocirano) {
37      /* Povecava se broj alociranih mesta */
38      alocirano = alocirano + KORAK;

40      /* Vrsi se realokacija memorije sa novom velicinom */
41      /*******/
42      /* Resenje sa funkcijom malloc() */
43      /*******/
44      /* Vrsi se alokacija memorije sa novom velicinom, a adresa
45         pocetka novog memorijskog bloka se cuva u promenljivoj
46         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 dinamicki alociranu
56             memoriju da oslobodimo. U ovom slucaju samo memoriju
57             na adresi a */
58          free(a);

59          /* Završavamo program */
60          exit(EXIT_FAILURE);
61      }

62
63      /* Svih n elemenata koji pocinju na adresi a prepisujemo
64         na novu adresu b */
65      for (i = 0; i < n; i++)
66          b[i] = a[i];

67
68      /* Posle prepisivanja oslobadjamo blok memorije sa
69         pocetnom adresom u a */
70      free(a);

71
72      /* Promenljivoj a dodeljujemo adresu pocetka novog, veceg
73         bloka koji je prilikom alokacije zapamcen u
74         promenljivoj b */
75      a = b;

76
77      /*******/
78      /* Resenje sa funkcijom realloc() */
79      /*******/
80      /* Zbog funkcije realloc je neophodno da i u prvoj
81         iteraciji "a" bude inicijalizovano na NULL */
82      /*
83
84      a = (int*) realloc(a,alocirano*sizeof(int));

85
86      if(a == NULL) { fprintf(stderr, "realloc(): ");
87          fprintf(stderr, "greska pri alokaciji memorije.\n");

```

## 2 Pokazivači

```
88         exit(EXIT_FAILURE); } */
89     }
90
91     /* Sместamo element u niz */
92     a[n++] = x;
93
94     /* i učitavamo sledeci element */
95     scanf("%d", &x);
96 }
97
98 /* Ispisujemo brojeve u obrnutom poretку */
99 for (n--; n >= 0; n--)
100     printf("%d ", a[n]);
101     printf("\n");
102
103 /* Oslobadjamo dinamički alociranu memoriju */
104 free(a);
105
106 /* Program se završava */
107 exit(EXIT_SUCCESS);
108 }
```

### Rešenje 2.17

```
#include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX 1000
6
7  /* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
8   Ovako nesto se moze improvizovati tako sto se u funkciji
9   dinamički kreira niz potrebne velicine, popuni se potrebnim
10  informacijama, a zatim se vrati njegova adresa. Imajuci u
11  vidu cinjenicu da dinamički kreiran objekat ne nestaje kada
12  se izadje iz funkcije koja ga je kreirala, vraceni pokazivac
13  se kasnije u pozivajucoj funkciji moze koristiti za pristup
14  "vracenom" nizu. Medjutim, pozivajuca funkcija ima
15  odgovornost i da se brine o dealokaciji istog prostora */
16
17  /* Funkcija dinamički kreira niz karaktera u koji smesta
18  rezultat nadovezivanja niski. Adresa niza se vraci kao
19  povratna vrednost. */
20  char *nadovezi(char *s, char *t)
21  {
22      /* Dinamički kreiramo prostor dovoljne velicine */
23      char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
24                               * sizeof(char));
25
26      /* Proveravamo uspeh alokacije */
27      if (p == NULL) {
```

```

28     fprintf(stderr, "malloc(): ");
    fprintf(stderr, "greska pri alokaciji memorije.\n");
30     exit(EXIT_FAILURE);
}

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

        for(j=0; t[j]!='\0'; i++, j++) p[i]=t[j];

        p[i]='\0'; */

42     /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
        string.h */
44     strcpy(p, s);
46     strcat(p, t);

48     /* Vracamo pokazivac p */
    return p;
50 }

52 int main()
{
54     char *s = NULL;
    char s1[MAX], s2[MAX];

56     /* Ucitavamo dve niske koje cemo da nadovezemo */
58     scanf("%s", s1);
    scanf("%s", s2);

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

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

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

70     return 0;
}

```

### Rešenje 2.18

```

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

```

```
int main()
6 {
    int i, j;

8
    /* Pokazivac na dinamički alociran niz pokazivaca na vrste
       matrice */
10    double **A = NULL;

12
    /* Broj vrsta i broj kolona */
14    int n = 0, m = 0;

16
    /* Trag matrice */
    double trag = 0;

18
    /* Unosimo dimenzije matrice */
20    scanf("%d%d", &n, &m);

22
    /* Dinamički alociramo prostor za n pokazivaca na double */
    A = malloc(sizeof(double *) * n);

24
    /* Proveramo da li je doslo do greske pri alokaciji */
26    if (A == NULL) {
        fprintf(stderr, "malloc(): ");
28        fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
30    }

32
    /* Dinamički alociramo prostor za elemente u vrstama */
    for (i = 0; i < n; i++) {
34        A[i] = malloc(sizeof(double) * m);

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

44            exit(EXIT_FAILURE);
        }
46    }

48
    /* 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++)
59         trag += A[i][i];
60
61     printf("%.2f\n", trag);
62
63     /* Oslobadjamo prostor rezervisan za svaku vrstu */
64     for (j = 0; j < n; j++)
65         free(A[j]);
66
67     /* Oslobadjamo memoriju za niz pokazivaca na vrste */
68     free(A);
69
70     return 0;
71 }
```

### Rešenje 2.19

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  void ucitaj_matricu(int **M, int n, int m)
6  {
7      int i, j;
8
9      /* Popunjavamo matricu vrstu po vrstu */
10     for (i = 0; i < n; i++)
11         /* Popunjavamo i-tu vrstu matrice */
12         for (j = 0; j < m; j++)
13             scanf("%d", &M[i][j]);
14 }
15
16 void ispis_elemente_ispod_dijagonale(int **M, int n, int m)
17 {
18     int i, j;
19
20     for (i = 0; i < n; i++) {
21         /* Ispisujemo elemente ispod glavne dijagonale matrice */
22         for (j = 0; j <= i; j++)
23             printf("%d ", M[i][j]);
24         printf("\n");
25     }
26 }
27
28 int main()
29 {
30     int m, n, i, j;
31     int **matrica = NULL;
32
33     /* Unosimo dimenzije matrice */
```

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

35
/* Alociramo prostor za niz pokazivaca na vrste matrice */
37 matrica = (int **) malloc(n * sizeof(int *));
if (matrica == NULL) {
39     fprintf(stderr, "malloc(): Neuspela alokacija\n");
    exit(EXIT_FAILURE);
41 }

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

    if (matrica[i] == NULL) {
47         fprintf(stderr, "malloc(): Neuspela alokacija\n");
49         for (j = 0; j < i; j++)
            free(matrica[j]);
51         free(matrica);
        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
61   oslobadjamo prostor rezervisan za svaku vrstu */
for (j = 0; j < n; j++)
63     free(matrica[j]);

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

69 return 0;
}
```

### Rešenje 2.20

```
#include<stdio.h>

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

### Rešenje 2.21

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 /* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni(float **a, int n)
7 {
8     int i, j;
9
10    for (i = 0; i < n; i++)
11        for (j = 0; j < n; j++)
12            /* Ako je indeks vrste manji od indeksa kolone */
13            if (i < j)
14                /* element se nalazi iznad glavne dijagonale pa ga
15                 polovimo */
16                a[i][j] /= 2;
17            else
18                /* Ako je indeks vrste veci od indeksa kolone */
19                if (i > j)
20                    /* element se nalazi ispod glavne dijagonale pa ga
21                     dupliramo */
22                    a[i][j] *= 2;
23 }
24
25 /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
26    sporedne dijagonale */
27 float zbir_ispod_sporedne_dijagonale(float **m, int n)
28 {
29     int i, j;
30     float zbir = 0;
31
32     for (i = 0; i < n; i++)
33         for (j = 0; j < n; j++)
34             /* Ukoliko je zbir indeksa vrste i indeksa kolone elementa
35              veci od n-1, to znaci da se element nalazi ispod
36              sporedne dijagonale */
37             if (i + j > n - 1)
38                 zbir += fabs(m[i][j]);
39
40     return zbir;
41 }
42
43 /* Funkcija ucitava elemente kvadratne matrice dimenzije n iz
44    zadate datoteke */
45 void ucitaj_matricu(FILE * ulaz, float **m, int n)
46 {
47     int i, j;
48
49     for (i = 0; i < n; i++)
50         for (j = 0; j < n; j++)
51             fscanf(ulaz, "%f", &m[i][j]);
```

```
52 }

54 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
   standardni izlaz */
56 void ispisi_matricu(float **m, int n)
57 {
58     int i, j;

60     for (i = 0; i < n; i++) {
61         for (j = 0; j < n; j++)
62             printf("%.2f ", m[i][j]);
63         printf("\n");
64     }
65 }

66
67 /* Funkcija alokira memoriju za kvadratnu matricu dimenzije n */
68 float **alociraj_memoriju(int n)
69 {
70     int i, j;
71     float **m;

72
73     m = (float **) malloc(n * sizeof(float *));
74     if (m == NULL) {
75         fprintf(stderr, "malloc(): Neuspela alokacija\n");
76         exit(EXIT_FAILURE);
77     }

78
79     /* Za svaku vrstu matrice */
80     for (i = 0; i < n; i++) {
81         /* Alociramo memoriju */
82         m[i] = (float *) malloc(n * sizeof(float));

83
84         /* Proveravamo da li je doslo do greske pri alokaciji */
85         if (m[i] == NULL) {
86             /* Ako jeste, ispisujemo poruku */
87             printf("malloc(): neuspela alokacija memorije!\n");

88
89             /* Oslobadjamo memoriju zauzetu do ovog koraka */
90             for (j = 0; j < i; j++)
91                 free(m[j]);
92             free(m);
93             exit(EXIT_FAILURE);
94         }
95     }
96     return m;
97 }

98
99 /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom
100 dimenzije n */
101 void oslobodi_memoriju(float **m, int n)
102 {
103     int i;
```



```
104     for (i = 0; i < n; i++)
106         free(m[i]);
107     free(m);
108 }

110 int main(int argc, char *argv[])
111 {
112     FILE *ulaz;
113     float **a;
114     int n;

116     /* Ako korisnik nije uneo trazene argumente, prijavljujemo
117        gresku */
118     if (argc < 2) {
119         printf("Greska: ");
120         printf("Nedovoljan broj argumenata komandne linije.\n");
121         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
122         exit(EXIT_FAILURE);
123     }

124     /* Otvaramo datoteku za citanje */
125     ulaz = fopen(argv[1], "r");
126     if (ulaz == NULL) {
127         fprintf(stderr, "Greska: ");
128         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
129                 argv[1]);
130         exit(EXIT_FAILURE);
131     }

132 }

134     /* citamo dimenziju matrice */
135     fscanf(ulaz, "%d", &n);

136     /* Alociramo memoriju */
137     a = alociraj_memoriju(n);

138     /* Ucitavamo elemente matrice */
139     ucitaj_matricu(ulaz, a, n);

140     float zbir = zbir_ispod_sporodne_dijagonale(a, n);

142     /* Pozivamo funkciju za modifikovanje elemenata */
143     izmeni(a, n);

144     /* Ispisujemo rezultat */
145     printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
146     printf("je %.2f.\n", zbir);

147     printf("Transformisana matrica je:\n");
148     ispisi_matricu(a, n);

149     /* Oslobadjamo memoriju */
```

## 2 Pokazivači

---

```
156   oslobodi_memoriju(a, n);  
158   /* Zatvaramo datoteku */  
      fclose(ulaz);  
160  
      /* i prekidamo sa izvršavanjem programa */  
162   return 0;  
}
```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

Rešenje 2.26

```
2  #include <stdio.h>  
   #include <stdlib.h>  
4  #include <math.h>  
   #include <string.h>  
6  
   /* NAPOMENA: Zaglavlje math.h sadrzi deklaracije raznih  
8   matematičkih funkcija. dIzmeu ostalog, to su čsledee  
   funkcije: double sin(double x); double cos(double x); double  
10  tan(double x); double asin(double x); double acos(double x);  
   double atan(double x); double atan2(double y, double x);  
12  double sinh(double x); double cosh(double x); double  
   tanh(double x); double exp(double x); double log(double x);  
14  double log10(double x); double pow(double x, double y);  
   double sqrt(double x); double ceil(double x); double  
16  floor(double x); double fabs(double x); */  
  
18  /* Funkcija tabela() prihvata granice intervala a i b, broj  
   ekvidistantnih čtaaka n, kao i čpokaziva f koji pokazuje na  
20  funkciju koja prihvata double argument, i čvraa double  
   vrednost. Za tako datu funkciju ispisuje njene vrednosti u  
22  intervalu [a,b] u n ekvidistantnih čtaaka intervala */  
void tabela(double a, double b, int n, double (*fp) (double))  
24 {  
   int i;  
26   double x;  
  
28   printf("-----\n");
```

```

30     for (i = 0; i < n; i++) {
31         x = a + i * (b - a) / (n - 1);
32         printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
33     }
34     printf("-----\n");
35 }
36
37 /* Umesto da koristimo stepenu funkciju iz zaglavlja math.h ->
38    pow(a,2) čpozivaemo čkorisniku sqr(a) */
39 double sqr(double a)
40 {
41     return a * a;
42 }
43
44 int main(int argc, char *argv[])
45 {
46     double a, b;
47     int n;
48     /* Imena funkcija koja ćemo navoditi su čkraa ili čtano
49        duga 5 karaktera */
50     char ime_fje[6];
51     /* Pokazivac na funkciju koja ima jedan argument tipa double i
52        povratnu vrednost istog tipa */
53     double (*fp) (double);
54
55     /* Ako korisnik nije uneo žtraene argumente, prijavljujemo
56        šgreku */
57     if (argc < 2) {
58         printf("Greska: ");
59         printf("Nedovoljan broj argumenata komandne linije.\n");
60         printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
61             argv[0]);
62         exit(EXIT_FAILURE);
63     }
64
65     /* Niska ime_fje žsadri ime žtraene funkcije koja je
66        navedena u komandnoj liniji */
67     strcpy(ime_fje, argv[1]);
68
69     /* Inicijalizujemo čpokaziva na funkciju koja treba da se
70        tabelira */
71     if (strcmp(ime_fje, "sin") == 0)
72         fp = &sin;
73     else if (strcmp(ime_fje, "cos") == 0)
74         fp = &cos;
75     else if (strcmp(ime_fje, "tan") == 0)
76         fp = &tan;
77     else if (strcmp(ime_fje, "atan") == 0)
78         fp = &atan;
79     else if (strcmp(ime_fje, "acos") == 0)
80         fp = &acos;
81     else if (strcmp(ime_fje, "asin") == 0)

```

```
    fp = &asin;
82  else if (strcmp(ime_fje, "exp") == 0)
    fp = &exp;
84  else if (strcmp(ime_fje, "log") == 0)
    fp = &log;
86  else if (strcmp(ime_fje, "log10") == 0)
    fp = &log10;
88  else if (strcmp(ime_fje, "sqrt") == 0)
    fp = &sqrt;
90  else if (strcmp(ime_fje, "floor") == 0)
    fp = &floor;
92  else if (strcmp(ime_fje, "ceil") == 0)
    fp = &ceil;
94  else if (strcmp(ime_fje, "sqr") == 0)
    fp = &sqr;
96  else {
    printf("Program jos uvek ne podrzava trazenu funkciju!\n");
98  exit(EXIT_SUCCESS);
}

100
102 printf("Unesite krajeve intervala:\n");
scanf("%lf %lf", &a, &b);

104 printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
printf("(ukljucujuci krajeve intervala)?\n");
106 scanf("%d", &n);

108 /* Mreza mora da čukljuuje bar krajeve intervala, tako da se
    mora uneti broj veci od 2 */
110 if (n < 2) {
    fprintf(stderr, "Broj čtaaka žmree mora biti bar 2!\n");
112  exit(EXIT_FAILURE);
}

114
116 /* Ispisujemo ime funkcije */
printf("      x %10s(x)\n", ime_fje);

118 /* dProsleujemo funkciji tabela() funkciju zadatu kao
    argument komandne linije */
120 tabela(a, b, n, fp);

122 exit(EXIT_SUCCESS);
}
```

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  $\mathbf{a}$ , dužine  $\mathbf{n}$ , tražeći u njemu broj  $\mathbf{x}$ .
- (b) Napisati funkciju koja vrši binarnu pretragu sortiranog niza  $\mathbf{a}$ , dužine  $\mathbf{n}$ , tražeći u njemu broj  $\mathbf{x}$ .
- (c) Napisati funkciju koja vrši interpoacionu pretragu sortiranog niza  $\mathbf{a}$ , dužine  $\mathbf{n}$ , tražeći u njemu broj  $\mathbf{x}$ .

Napisati i program koji generiše slučajni rastući niz dimenzije  $\mathbf{n}$  (prvi argument komandne linije, ne veći od 1000000), i u njemu već napisanim funkcijama traži element  $\mathbf{x}$  (drugi argument komandne linije). Potrebna vremena za izvršavanje ovih funkcija upisati u fajl `vremena.txt`.

#### Test 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
```

#### Test Test 2

```
Poziv: ./a.out 100000 37842
```

```
Izlaz:
```

```
Linearna pretraga  
Element nije u nizu
```

```
-----  
Binarna pretraga  
Element nije u nizu
```

```
-----  
Interpolaciona pretraga  
Element nije u nizu
```

[Rešenje 3.1]

**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.

#### Primer 1

```
INTERAKCIJA PROGRAMA:
```

```
Unesite traženi broj: 11
```

```
Unesite sortiran niz elemenata:
```

```
2 5 6 8 10 11 23
```

```
Linearna pretraga
```

```
Pozicija elementa je 5.
```

```
Binarna pretraga
```

```
Pozicija elementa je 5.
```

```
Interpolaciona pretraga
```

```
Pozicija elementa je 5.
```

#### Test Upotreba programa 2

```
Interakcija programa:
```

```
Unesite traženi broj: 14
```

```
Unesite sortiran niz elemenata:
```

```
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.
```

[Rešenje 3.2]

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenata 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. U slučaju neuspješnih pretragi, štampati odgovarajuću poruku. 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.

*Primer 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
INTERAKCIJA PROGRAMA:
Unesite indeks studenta cije informacije zelite: 20140076
Indeks: 20140076, Ime i prezime: Sonja Stevanovic
Unesite prezime studenta cije informacije zelite: Popovic
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 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

```

*Test Test 2*

```

Poziv: ./a.out dat.txt
Datoteka:
12 53
2.342 34.1
-0.3 23
-1 2.1
123.5 756.12
Izlaz:
-1 2.1

```

*Test Test 3*

```

Poziv: ./a.out dat.txt -y
Datoteka:
12 53
2.342 34.1
-0.3 0.23
-1 2.1
123.5 756.12
Izlaz:
-0.3 0.23

```

[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: *Korisiti 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 indeks prvog elementa većeg od nule. 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 Upotreba programa 1 Test Upotreba programa 2 Test Upotreba programa 3*

<pre>   Interakcija programa:    Unesi rastuce sortiran    niz celih brojeva:    -151 -44 5 12 13 15    Izlaz:    2</pre>	<pre>   Interakcija programa:    Unesi rastuce sortiran    niz celih brojeva:    -100 -15 -11 -8 -7 -5    Izlaz:    -1</pre>	<pre>   Interakcija programa:    Unesi rastuce sortiran    niz celih brojeva:    -100 -15 0 13 55 124    258 315 516 7000    Izlaz:    3</pre>
---	--	--

[Rešenje 3.7]

**Zadatak 3.8** Napisati funkciju koja u opadajuće sortiranom nizu celih brojeva binarnom pretragom pronalazi indeks prvog elementa manjeg od nule. 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 Upotreba programa 1 Test Upotreba programa 2 Test Upotreba programa 3*

<pre>   Interakcija programa:    Unesi opadajuće    sortiran niz celih    brojeva:    151 44 5 -12 -13 -15    Izlaz:    3</pre>	<pre>   Interakcija programa:    Unesi opadajuće    sortiran niz celih    brojeva:    100 55 15 0 -15 -124    -155 -258 -315 -516    Izlaz:    4</pre>	<pre>   Interakcija programa:    Unesi opadajuće    sortiran niz celih    brojeva:    100 15 11 8 7 5 4 3 2    Izlaz:    -1</pre>
---	--	---

[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.

- Napisati funkciju linearne složenosti koja određuje logaritam pomeranjem broja udesno.
- Napisati funkciju logaritmske složenosti koja određuje logaritam koristeći binarnu pretragu.

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

*Test Upotreba programa 1    Test Upotreba programa 2    Test Upotreba programa 3*

<pre> Interakcija programa:   Unesi pozitivan ceo     broj: 4 Izlaz:   2 2 </pre>	<pre> Interakcija programa:   Unesi pozitivan ceo     broj: 17 Izlaz:   4 4 </pre>	<pre> Interakcija programa:   Unesi pozitivan ceo     broj: 1031 Izlaz:   10 10 </pre>
---	--	--

[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 Test 1*

```

Ulaz:
  2
  2 0 2 1
  1 2 2 2
Izlaz:
  26.57

```

*Test Test 2*

```

Ulaz:
  2
  1 0 1 1
  0 1 1 1
Izlaz:
  45

```

*Test Test 3*

```

Ulaz:
  3
  1 0 1 1
  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 standardnom izlazu.

### 3 Algoritmi pretrage i sortiranja

---

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

<i>Test Test 1</i>	<i>Test Test 2</i>	<i>Test Test 3</i>
Ulaz:    R    Izlaz:    Dusko Radovic	Ulaz:    E    Izlaz:    Dobrica Eric	Ulaz:    A    Izlaz:    Mika Antic

## 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.*

<i>Test Upotreba programa 1</i>	<i>Test Upotreba programa 2</i>	<i>Test Upotreba programa 3</i>
Interakcija programa:    Unesite elemente niza:    23 64 123 76 22 7    Izlaz:    1	Interakcija programa:    Unesite elemente niza:    21 654 65 123 65 12 61    Izlaz:    0	Interakcija programa:    Unesite elemente niza:    34 30    Izlaz:    4

[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 Upotreba programa 1    Test Upotreba programa 2    Test Upotreba programa 3*

<pre>Interakcija programa: Unesite prvu nisku:     anagram Unesite drugu nisku:     ramgana Izlaz: jesu</pre>	<pre>Interakcija programa: Interakcija programa: Unesite prvu nisku:     anagram Unesite drugu nisku:     anagrm Izlaz: nisu</pre>	<pre>Interakcija programa: Interakcija programa: Unesite prvu nisku: test Unesite drugu nisku: tset Izlaz: jesu</pre>
---	--	---

[Rešenje 3.13]

**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.*

*Test Upotreba programa 1    Test Upotreba programa 2    Test Upotreba programa 3*

<pre>Interakcija programa: Unesite elemente niza: 4 23 5 2 4 6 7 34 6 4 5 Izlaz: 4</pre>	<pre>Interakcija programa: Unesite elemente niza: 2 4 6 2 6 7 99 1 Izlaz: 2</pre>	<pre>Interakcija programa: Unesite elemente niza: 123 Izlaz: 123</pre>
--	---	--

[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.*

*Test Upotreba programa 1    Test Upotreba programa 2    Test Upotreba programa 3*

<pre>Interakcija programa: Unesite trazeni zbir: 34 Unesite elemente niza: 134 4 1 6 30 23 Izlaz: da</pre>	<pre>Interakcija programa: Unesite trazeni zbir: 12 Unesite elemente niza: 53 1 43 3 56 13 Izlaz: ne</pre>	<pre>Interakcija programa: Unesite trazeni zbir: 52 Unesite elemente niza: 52 Izlaz: ne</pre>
--	--	---

[Rešenje 3.15]

### 3 Algoritmi pretrage i sortiranja

**Zadatak 3.16** 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 rastuće sortiranim nizovima i na opadajuće 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`.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Poziv:</code> <code>time a.out 100000 -i -o</code>	<code>Poziv:</code> <code>time a.out 100000 -b -r</code>	<code>Poziv:</code> <code>time a.out 100000 -s</code>
<code>Izlaz:</code> <code>real 0m17.631s</code> <code>user 0m17.604s</code> <code>sys 0m0.000s</code>	<code>Izlaz:</code> <code>real 0m0.005s</code> <code>user 0m0.004s</code> <code>sys 0m0.000s</code>	<code>Izlaz:</code> <code>real 0m0.071s</code> <code>user 0m0.068s</code> <code>sys 0m0.000s</code>

[Rešenje 3.16]

**Zadatak 3.17** 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 zatim program koji testira ovu funkciju. Nizovi se unose sa standardnog ulaza sve dok se ne unese 0 i može se pretpostaviti da će njihove dimenzije biti manje od 256.

<i>Test Upotreba programa 1</i>	<i>Test Upotreba programa 2</i>
<code>Interakcija programa:</code> <code>Unesite elemente prvog niza:</code> <code>3 6 7 11 14 35 0</code> <code>Unesite elemente drugog niza:</code> <code>3 5 8 0</code> <code>Izlaz:</code> <code>3 3 5 6 7 8 11 14 35</code>	<code>Interakcija programa:</code> <code>Unesite elemente prvog niza:</code> <code>1 4 7 0</code> <code>Unesite elemente drugog niza:</code> <code>9 11 23 54 75 0</code> <code>Izlaz:</code> <code>1 4 7 9 11 23 54 75</code>

[Rešenje 3.17]

**Zadatak 3.18** Napisati program koji čita sadržaj dveju datoteka 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 jedinstveni

spisak upisuje u datoteku `ceo-tok.txt`. Pretpostaviti da je ime studenta nije duže od 10, a prezime od 15 karaktera.

### Test 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 3.18]

**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 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 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 Test 1

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

#### Test Test 2

```
Ulazna datoteka (biracki-spisak.txt):
Milan Milicevic
Izlaz:
1
```

[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 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

```

*Test Test 2*

```

Poziv: ./a.out in.txt out.txt
Ulazna datoteka (in.out):
  Milijana Maric 2009
Izlazna datoteka (out.txt):
  Milijana Maric 2009

```

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

#### Test 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 aktivnostima 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 broju urađenih zadataka, a ukoliko se i on poklapa sortirati po



prezimeni 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 Test 1

<p>Ulazna datoteka (aktivnosti.txt):</p> <pre> 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 </pre> <p>Izlazna datoteka (dat2.txt):</p> <pre> Studenti sortirani po broju zadataka opadajuće, pa po dužini imena rastuće: 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 </pre>	<p>Izlazna datoteka (dat1.txt):</p> <pre> Studenti sortirani po imenu leksikografski rastuće: 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 </pre> <p>Izlazna datoteka (dat3.txt):</p> <pre> Studenti sortirani po prisustvu opadajuće, pa po broju zadataka, pa po prezimenima leksikografski opadajuće: 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 </pre>
--	--

[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:

### 3 Algoritmi pretrage i sortiranja

---

- 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 standardnom izlazu ispisati informacije o pesmama sortiranim na opisani način. Uraditi zadatak bez pravljenja pretpostavki o maksimalnoj dužini imena izvođača i naslova pesme.

Test Test 1	Test Test 2	Test Test 3
<code>Poziv: ./a.out</code> <code>Ulazna datoteka:</code> 5 Ana - Nebo, 2342 Laza - Oblaci, 29 Pera - Ptice, 327 Jelena - Sunce, 92321 Mika - Kisa, 5341 <code>Izlaz:</code> Jelena - Sunce, 92321 Mika - Kisa, 5341 Ana - Nebo, 2342 Pera - Ptice, 327 Laza - Oblaci, 29	<code>Poziv: ./a.out -i</code> <code>Ulazna datoteka:</code> 5 Ana - Nebo, 2342 Laza - Oblaci, 29 Pera - Ptice, 327 Jelena - Sunce, 92321 Mika - Kisa, 5341 <code>Izlaz:</code> Ana - Nebo, 2342 Jelena - Sunce, 92321 Laza - Oblaci, 29 Mika - Kisa, 5341 Pera - Ptice, 327	<code>Poziv: ./a.out -n</code> <code>Ulazna datoteka:</code> 5 Ana - Nebo, 2342 Laza - Oblaci, 29 Pera - Ptice, 327 Jelena - Sunce, 92321 Mika - Kisa, 5341 <code>Izlaz:</code> 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 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. Na standardnom izlazu ispisati odgovarajuću poruku.

#### Test 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
```

#### Test 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 standardnom izlazu.

#### Test 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 biblioteka 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 standardnom izlazu.

#### Test Upotreba programa 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 biblioteka funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće.

### 3.3 Bibliotečke funkcije pretrage i sortiranja

Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova ili poruka ukoliko nema takvog. Potom, sa standardnog 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.

#### Primer 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 karakter). Sortirati niz  $S$  (bibliotečkom funkcijom `qsort`) i proveriti da li u njemu ima identičnih stringova.

*Test Upotreba programa 1    Test Upotreba programa 2    Test Upotreba programa 3*

<pre>Interakcija programa:   Unesite broj niski: 4   Unesite niske:     prog search sort search Izlaz:   ima</pre>	<pre>Interakcija programa:   Unesite broj niski: 3   Unesite niske:     test kol ispit Izlaz:   nema</pre>	<pre>Interakcija programa:   Unesite broj niski: 5   Unesite niske:     a ab abc abcd abcde Izlaz:   nema</pre>
--	--	---

[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. I ime i prezime neće biti duže od 20 karaktera. Napisati program koji sortira (korišćenjem funkcije `qsort`) studente po broju poena opadajuće (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 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
```

*Test Test 2*

```
Poziv: ./a.out -p
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):
  mr14123 Marko Antic 20
  ml13011 Ivana Mitrovic 19
  mv14003 Jovan Jovanovic 17
  ml13066 Pera Simic 15
  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),

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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 Test 1

<b>Poziv:</b> <code>./a.out datoteka.txt</code>		
<b>Datoteka:</b>	<b>Ulaz:</b>	<b>Izlaz:</b>
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. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu.

#### Test Upotreba programa 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
```

#### Test Upotreba programa 2

```
Interakcija programa:
Unesite dimenzije matrice: 4 4
Unesite elemente matrice po vrstama:
34 12 54 642
1 2 3 4
53 2 1 5
54 23 5 671
Sortirana matrica je:
1 2 3 4
53 2 1 5
34 12 54 642
54 23 5 671
```

[Rešenje 3.37]

**Zadatak 3.38** Za zadatu 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 standardnom izlazu.

#### Test Upotreba programa 1

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

#### Test Upotreba programa 2

```
Interakcija programa:
  Unesite dimenziju matrice: 4
  Unesite elemente matrice po vrstama:
    34 12 54 642
    1 2 3 4
    53 2 1 5
    54 23 5 671
  Sortirana matrica je:
    12 34 54 642
    2 1 3 4
    2 53 1 5
    23 54 5 671
```

## 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 opcijom
7    -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
10    njemu element x. Pretraga se vrši prostom iteracijom kroz niz. Ako
11    se element pronadje funkcija vraca indeks pozicije na kojoj je
12    pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
13    pronadjen u nizu, funkcija vraca -1, kao indikator neuspesne
14    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
24 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
25    pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
26 int binarna_pretraga(int a[], int n, int x)
27 {
28     int levi = 0;
29     int desni = n - 1;
30     int srednji;
```



```

31  /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
33      /* Srednji indeks je njihova aritmeticka sredina */
      srednji = (levi + desni) / 2;
35      /* Ako je element sa sredisnjim indeksom veci od x, tada se x
        mora nalaziti u levoj polovini niza */
37      if (x < a[srednji])
          desni = srednji - 1;
39      /* Ako je element sa sredisnjim indeksom manji od x, tada se x
        mora nalaziti u desnoj polovini niza */
41      else if (x > a[srednji])
          levi = srednji + 1;
43      else
          /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
45         x pronadjen na poziciji srednji */
          return srednji;
47  }
    /* Ako element x nije pronadjen, vraca se -1 */
49  return -1;
}

51
/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
53  pozicije nadjenog elementa ili -1, ako element nije pronadjen */
int interpolaciona_pretraga(int a[], int n, int x)
55 {
    int levi = 0;
57    int desni = n - 1;
    int srednji;
59    /* Dokle god je indeks levi levo od indeksa desni... */
    while (levi <= desni) {
61        /* Ako je trazeni element manji od pocetnog ili veci od
           poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
63        nije u tom delu niza. Ova provera je neophodna, da se ne bi
           dogodilo da se prilikom izracunavanja indeksa srednji izadje
65        izvan opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
67            return -1;
        /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
69        a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
           jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
71        desni). Ova provera je neophodna, jer bi se u suprotnom
           prilikom izracunavanja indeksa srednji pojavilo deljenje
73        nulom. */
        else if (a[levi] == a[desni])
75            return levi;
        /* Racunanje srednjeg indeksa */
77        srednji =
            levi +
79            ((double) (x - a[levi]) / (a[desni] - a[levi])) *
            (desni - levi);
81        /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
           verovatno biti blize trazenoj vrednosti nego da je prosto uvek

```

### 3 Algoritmi pretrage i sortiranja

---

```
83      uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
      porediti sa pretragom recnika: ako neko trazi rec na slovo 'B
      ',
85      sigurno nece da otvori recnik na polovini, vec verovatno negde
      blize pocetku. */
87      /* Ako je element sa indeksom srednji veci od trazenog, tada se
      trazeni element mora nalaziti u levoj polovini niza */
89      if (x < a[srednji])
          desni = srednji - 1;
91      /* Ako je element sa indeksom srednji manji od trazenog, tada se
      trazeni element mora nalaziti u desnoj polovini niza */
93      else if (x > a[srednji])
          levi = srednji + 1;
95      else
          /* Ako je element sa indeksom srednji jednak trazenom, onda se
          pretraga zavrшава na poziciji srednji */
          return srednji;
97      }
99      /* U slucaju neuspesne pretrage vraca se -1 */
101     return -1;
103 }

104 /* Funkcija main */
105 int main(int argc, char **argv)
106 {
107     int a[MAX];
108     int n, i, x;
109     struct timespec time1, time2, time3, time4, time5, time6;
110     FILE *f;
111
112     /* Provera argumenata komandne linije */
113     if (argc != 3) {
114         fprintf(stderr,
115             "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
116         ;
117         exit(EXIT_FAILURE);
118     }
119
120     /* Dimenzija niza */
121     n = atoi(argv[1]);
122     if (n > MAX || n <= 0) {
123         fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
124         exit(EXIT_FAILURE);
125     }
126
127     /* Broj koji se trazi */
128     x = atoi(argv[2]);
129
130     /* Elementi niza se generisu slucajno, tako da je svaki sledeci
131     veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
132     pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
133     izgledao, seed se postavlja na tekuce vreme u sekundama od Nove
```

```

133     godine 1970. random()%100 daje brojeve izmedju 0 i 99 */
134     srandom(time(NULL));
135     for (i = 0; i < n; i++)
136         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
137
138     /* Linearna pretraga */
139     printf("Linearna pretraga\n");
140     /* Vreme proteklo od Nove godine 1970 */
141     clock_gettime(CLOCK_REALTIME, &time1);
142     i = linearna_pretraga(a, n, x);
143     /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
144        linearnu pretragu */
145     clock_gettime(CLOCK_REALTIME, &time2);
146     if (i == -1)
147         printf("Element nije u nizu\n");
148     else
149         printf("Element je u nizu na poziciji %d\n", i);
150     printf("-----\n");
151
152     /* Binarna pretraga */
153     printf("Binarna pretraga\n");
154     clock_gettime(CLOCK_REALTIME, &time3);
155     i = binarna_pretraga(a, n, x);
156     clock_gettime(CLOCK_REALTIME, &time4);
157     if (i == -1)
158         printf("Element nije u nizu\n");
159     else
160         printf("Element je u nizu na poziciji %d\n", i);
161     printf("-----\n");
162
163     /* Interpolaciona pretraga */
164     printf("Interpolaciona pretraga\n");
165     clock_gettime(CLOCK_REALTIME, &time5);
166     i = interpolaciona_pretraga(a, n, x);
167     clock_gettime(CLOCK_REALTIME, &time6);
168     if (i == -1)
169         printf("Element nije u nizu\n");
170     else
171         printf("Element je u nizu na poziciji %d\n", i);
172     printf("-----\n");
173
174     /* Podaci o izvrsavanju programa bivaju upisani u log fajl */
175     if ((f = fopen("vremena.txt", "a")) == NULL) {
176         fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
177         exit(EXIT_FAILURE);
178     }
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 +
            time4.tv_nsec - time3.tv_nsec);
187     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
            (time6.tv_sec - time5.tv_sec) * 1000000000 +
189         time6.tv_nsec - time5.tv_nsec);

191     /* Zatvaranje datoteke */
    fclose(f);
193
    return 0;
195 }
```

#### Rešenje 3.2

```
1  #include <stdio.h>

3  int lin_pretgraga_rek_sufiks(int a[], int n, int x)
{
5     int tmp;
    /* Izlaz iz rekurzije */
7     if (n <= 0)
        return -1;
9     /* Ako je prvi element trazeni */
    if (a[0] == x)
11        return 0;
    /* Pretraga ostatka niza */
13    tmp = lin_pretgraga_rek_sufiks(a + 1, n - 1, x);
    return tmp < 0 ? tmp : tmp + 1;
15 }

17 int lin_pretgraga_rek_prefiks(int a[], int n, int x)
{
19     /* Izlaz iz rekurzije */
    if (n <= 0)
21        return -1;
    /* Ako je poslednji element trazeni */
23    if (a[n - 1] == x)
        return n - 1;
25    /* Pretraga ostatka niza */
    return lin_pretgraga_rek_prefiks(a, n - 1, x);
27 }

29 int bin_pretgraga_rek(int a[], int l, int d, int x)
{
31     int srednji;
    if (l > d)
33        return -1;
    /* Sredisnja pozicija na kojoj se trazi vrednost x */
35    srednji = (l + d) / 2;
    /* Ako je element na sredisnjoj poziciji trazeni */
37    if (a[srednji] == x)
```

```

    return srednji;
39 /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
    pretrazuje se desna polovina niza */
41 if (a[srednji] < x)
    return bin_pretgraga_rek(a, srednji + 1, d, x);
43 /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
    pretrazuje se leva polovina niza */
45 else
    return bin_pretgraga_rek(a, l, srednji - 1, x);
47 }

49 int interp_pretgraga_rek(int a[], int l, int d, int x)
51 {
    int p;
53 if (x < a[l] || x > a[d])
    return -1;
55 if (a[d] == a[l])
    return l;
57 /* Pozicija na kojoj se trazi vrednost x */
    p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
59 if (a[p] == x)
    return p;
61 if (a[p] < x)
    return interp_pretgraga_rek(a, p + 1, d, x);
63 else
    return interp_pretgraga_rek(a, l, p - 1, x);
65 }

67 #define MAX 1024

69 int main()
71 {
    int a[MAX];
    int x;
    int i, indeks;

75 /* Ucitavanje trazenog broja */
    printf("Unesite trazeni broj: ");
77 scanf("%d", &x);

79 /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
    korisnik pritisne CTRL+D za naznaku kraja */
81 i = 0;
    printf("Unesite sortiran niz elemenata: ");
83 while (scanf("%d", &a[i]) == 1) {
    i++;
85 }

87 /* Linearna pretraga */
    printf("Linearna pretraga\n");
89 indeks = lin_pretgraga_rek_sufiks(a, i, x);

```

### 3 Algoritmi pretrage i sortiranja

```
91     if (indeks == -1)
92         printf("Element se ne nalazi u nizu.\n");
93     else
94         printf("Pozicija elementa je %d.\n", indeks);
95
96     /* Binarna pretraga */
97     printf("Binarna pretraga\n");
98     indeks = bin_pretgraga_rek(a, 0, i - 1, x);
99     if (indeks == -1)
100         printf("Element se ne nalazi u nizu.\n");
101     else
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 postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
9
10 typedef struct {
11     /* Indeks mora biti tipa long jer su podaci u datoteci preveliki za
12      int, npr. 20140123 */
13     long indeks;
14     char ime[MAX_DUZINA];
15     char prezime[MAX_DUZINA];
16 } Student;
17
18 /* Učitani niz studenata će biti sortiran rastuće prema indeksu, jer
   su studenti u datoteci već sortirani. Iz tog razloga pretraga po
   indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
   jer nema garancije da postoji uredjenje po prezimenu. */
19
20
21
22 /* Funkcija traži u sortiranom nizu studenata a[] dužine n studenata
   sa indeksom x i vraća indeks pozicije nadjenog člana niza ili -1,
   ako element nije pronađen. */
23
```

```
26 int binarna_pretraga(Student a[], int n, long x)
27 {
28     int levi = 0;
29     int desni = n - 1;
30     int srednji;
31     /* Dokle god je indeks levi levo od indeksa desni */
32     while (levi <= desni) {
33         /* Racuna se srednja pozicija */
34         srednji = (levi + desni) / 2;
35         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
36            se trazeni indeks mora nalaziti u levoj polovini niza */
37         if (x < a[srednji].indeks)
38             desni = srednji - 1;
39         /* Ako je pak manji od trazenog, tada se on mora nalaziti u
40            desnoj polovini niza */
41         else if (x > a[srednji].indeks)
42             levi = srednji + 1;
43         else
44             /* Ako je jednak trazenom indeksu x, tada je pronadjen student
45                sa trazenom indeksom na poziciji srednji */
46             return srednji;
47     }
48     /* Ako nije pronadjen, vraca se -1 */
49     return -1;
50 }

52 /* Linearnom pretragom niza studenata trazi se prezime x */
53 int linearna_pretraga(Student a[], int n, char x[])
54 {
55     int i;
56     for (i = 0; i < n; i++)
57         /* Poredjenje prezimena i-tog studenta i poslatog x */
58         if (strcmp(a[i].prezime, x) == 0)
59             return i;
60     return -1;
61 }

62 /* Main funkcija mora imati argumente jer se ime datoteke prosledjuje
63    kao argument komandne linije */
64 int main(int argc, char *argv[])
65 {
66     Student dosije[MAX_STUDENATA];
67     FILE *fin = NULL;
68     int i;
69     int br_studenata = 0;
70     long trazeni_indeks = 0;
71     char trazeno_prezime[MAX_DUZINA];

72     /* Provera da li je korisnik prilikom poziva programa prosledio ime
73        datoteke sa informacijama o studentima */
74     if (argc != 2) {
75         fprintf(stderr,
```

### 3 Algoritmi pretrage i sortiranja

---

```
78         "Greska: Program se poziva sa %s ime_datoteke\n",
          argv[0]);
80     exit(EXIT_FAILURE);
81 }
82
83 /* Otvaranje datoteke */
84 fin = fopen(argv[1], "r");
85 if (fin == NULL) {
86     fprintf(stderr,
87         "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
88     exit(EXIT_FAILURE);
89 }
90
91 /* Citanje se vrši sve dok postoji red sa informacijama o studentu
92  */
93 i = 0;
94 while (1) {
95     if (i == MAX_STUDENATA)
96         break;
97     if (fscanf
98         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
99          dosije[i].prezime) != 3)
100         break;
101     i++;
102 }
103 br_studenata = i;
104
105 /* Nakon citanja, datoteka vise nije neophodna i odmah se zatvara
106  */
107 fclose(fin);
108
109 /* Unos indeksa koji se binarno trazi u nizu */
110 printf("Unesite indeks studenta cije informacije želite: ");
111 scanf("%ld", &trazen_indeks);
112 i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
113 /* Rezultat binarne pretrage */
114 if (i == -1)
115     printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
116 else
117     printf("Indeks: %ld, Ime i prezime: %s %s\n",
118           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
119
120 /* Unos prezimena koje se linearno trazi u nizu */
121 printf("Unesite prezime studenta cije informacije želite: ");
122 scanf("%s", trazeno_prezime);
123 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
124 /* Rezultat linearne pretrage */
125 if (i == -1)
126     printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
127 else
128     printf("Indeks: %ld, Ime i prezime: %s %s\n",
129           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
```



```

128     return 0;
130 }

```

### Rešenje 3.4

```

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

typedef struct {
    long indeks;
    char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
} Student;

int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
{
    /* Ako je pozicija elementa na levom kraju veca od pozicije
       elementa na desnom kraju dela niza koji se pretrazuje, onda se
       zapravo pretrazuje prazan deo niza. U praznom delu niza nema
       trazenog elementa pa se vraca -1 */
    if (levi > desni)
        return -1;
    /* Racunanje pozicije srednjeg elementa */
    int srednji = (levi + desni) / 2;
    /* Da li je srednji bas onaj trazeni */
    if (a[srednji].indeks == x) {
        return srednji;
    }
    /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
       poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
       je poznato da je niz sortirani po indeksu u rastucem poretku. */
    if (x < a[srednji].indeks)
        return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
    /* Inace ga treba traziti u desnoj polovini */
    else
        return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
}

int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
{
    /* Ako je niz prazan, vraca se -1 */
    if (n == 0)
        return -1;
    /* Kako se trazi prvi student sa trazanim prezimenom, pocinje se sa
       prvim studentom u nizu. */

```

### 3 Algoritmi pretrage i sortiranja

---

```
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 }

52 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
53 {
54     /* Ako je niz prazan, vraca se -1 */
55     if (n == 0)
56         return -1;
57     /* Ako se trazi poslednji student sa trazanim prezimenom, pocinje
58        se 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 }

64 /* Main funkcija mora imate argumente jer se ime datoteke prosledjuje
65    kao argument komandne linije */
66 int main(int argc, char *argv[])
67 {
68     Student dosije[MAX_STUDENATA];
69     FILE *fin = NULL;
70     int i;
71     int br_studenata = 0;
72     long trazeni_indeks = 0;
73     char trazeno_prezime[MAX_DUZINA];
74
75     /* Provera da li je korisnik prilikom poziva prosledio ime datoteke
76        sa informacijama o studentima */
77     if (argc != 2) {
78         fprintf(stderr,
79             "Greska: Program se poziva sa %s ime_datoteke\n",
80             argv[0]);
81         exit(EXIT_FAILURE);
82     }
83
84     /* Otvaranje datoteke */
85     fin = fopen(argv[1], "r");
86     if (fin == NULL) {
87         fprintf(stderr,
88             "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
89         exit(EXIT_FAILURE);
90     }
91
92     /* Citanje se vrshi sve dok postoji sledeci red sa informacijama o
93        studentu */
94     i = 0;
95     while (1) {
96         if (i == MAX_STUDENATA)
97             break;
```

```

98     if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
100         dosije[i].prezime) != 3)
        break;
102     i++;
    }
104     br_studenata = i;

106     /* Nakon citanja datoteka nam vise nije neophodna i odmah se
        zatvara */
108     fclose(fin);

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

    if (i == -1)
116         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
118         printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
120

    /* Unos prezimena koje se linearno trazi u nizu */
122     printf("Unesite prezime studenta cijje informacije zelite: ");
    scanf("%s", trazeno_prezime);
124     i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
        trazeno_prezime);

126     if (i == -1)
        printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
128     else
        printf
130            ("Prvi takav student:\nIndeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132

    return 0;
134 }

```

### Rešenje 3.5

```

1  #include <stdio.h>
   #include <string.h>
3  #include <math.h>
   #include <stdlib.h>
5
   /* Struktura koja opisuje tacku u ravni */
7  typedef struct Tacka {
    float x;
9    float y;
    } Tacka;
11

```

### 3 Algoritmi pretrage i sortiranja

---

```
13  /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
    pocetka (0,0) */
15  float rastojanje(Tacka A)
17  {
    return sqrt(A.x * A.x + A.y * A.y);
19  }

19  /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u nizu
    zadatih tacaka t dimenzije n */
21  Tacka najbliza_koordinatnom(Tacka t[], int n)
23  {
    Tacka najbliza;
    int i;
    najbliza = t[0];
    for (i = 1; i < n; i++) {
25        if (rastojanje(t[i]) < rastojanje(najbliza)) {
27            najbliza = t[i];
29        }
    }
    return najbliza;
31 }

33 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih tacaka
    t dimenzije n */
35 Tacka najbliza_x_osi(Tacka t[], int n)
37 {
    Tacka najbliza;
    int i;
    najbliza = t[0];
    for (i = 1; i < n; i++) {
41        if (fabs(t[i].x) < fabs(najbliza.x)) {
43            najbliza = t[i];
45        }
    }
    return najbliza;
47 }

49 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
    t dimenzije n */
51 Tacka najbliza_y_osi(Tacka t[], int n)
53 {
    Tacka najbliza;
    int i;
    najbliza = t[0];
    for (i = 1; i < n; i++) {
55        if (fabs(t[i].y) < fabs(najbliza.y)) {
57            najbliza = t[i];
59        }
    }
    return najbliza;
61 }
63 }
```

```
65 #define MAX 1024
67 int main(int argc, char *argv[])
68 {
69     FILE *ulaz;
70     Tacka tacke[MAX];
71     Tacka najbliza;
72     int i, n;
73
74     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75        ime datoteke sa tackama */
76     if (argc < 2) {
77         fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
79         return EXIT_FAILURE;
80     }
81
82     /* Otvaranje datoteke za citanje */
83     ulaz = fopen(argv[1], "r");
84     if (ulaz == NULL) {
85         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
87         return EXIT_FAILURE;
88     }
89
90     /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
91        tackama; i predstavlja indeks tekuce tacke */
92     i = 0;
93     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94         i++;
95     }
96     n = i;
97
98     /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
99        dodatnih argumenata */
100     if (argc == 2)
101         /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
102         najbliza = najbliza_koordinatnom(tacke, n);
103     /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
104        pitanju opcija -x */
105     else if (strcmp(argv[2], "-x") == 0)
106         /* Racuna se rastojanje u odnosu na x osu */
107         najbliza = najbliza_x_osi(tacke, n);
108     /* Ako je u pitanju opcija -y */
109     else if (strcmp(argv[2], "-y") == 0)
110         /* Racuna se rastojanje u odnosu na y osu */
111         najbliza = najbliza_y_osi(tacke, n);
112     else {
113         /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
114            korisnika i prekida se izvršavanje programa */
115         fprintf(stderr, "Pogresna opcija\n");
```

```
117     return EXIT_FAILURE;
118 }
119 /* Stapanje koordinata trazene tacke */
120 printf("%g %g\n", najbliza.x, najbliza.y);
121
122 /* Zatvaranje datoteke */
123 fclose(ulaz);
124
125 return 0;
126 }
```

#### Rešenje 3.6

```
1 #include <stdio.h>
2 #include <math.h>
3
4 /* Tacnost */
5 #define EPS 0.001
6
7 int main()
8 {
9     double l, d, s;
10
11     /* Kako je u pitanju interval [0, 2] leva granica je 0, a desna 2
12      */
13     l = 0;
14     d = 2;
15
16     /* Sve dok se ne pronadje trazena vrednost argumenta */
17     while (1) {
18         /* Polovi se interval */
19         s = (l + d) / 2;
20         /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
21            tacnosti, prekida se pretraga */
22         if (fabs(cos(s)) < EPS) {
23             break;
24         }
25         /* Ako je nula u levom delu intervala, nastavlja se pretraga na
26            [l, s] */
27         if (cos(l) * cos(s) < 0)
28             d = s;
29         else
30             /* Inace, na intervalu [s, d] */
31             l = s;
32     }
33
34     /* Stapanje vrednost trazene tacke */
35     printf("%g\n", s);
36
37     return 0;
38 }
```

```
}

```

### Rešenje 3.7

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   int prvi_veci_od_nule(int niz[], int n)
5  {
   /* Granice pretrage */
7   int l = 0, d = n - 1;
   int s;
9   /* Sve dok je leva manja od desne granice */
   while (l <= d) {
11    /* Racuna se sredisnja pozicija */
    s = (l + d) / 2;
13    /* Ako je broj na toj poziciji veci od nule, a eventualni njegov
    prethodnik manji ili jednak nuli, pretraga je zavrшена */
15    if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
        return s;
17    /* U slucaju broja manjeg ili jednakog nuli, pretrazuje se desna
    polovina niza */
19    if (niz[s] <= 0)
        l = s + 1;
21    /* A inace, leva polovina */
    else
23        d = s - 1;
   }
25   return -1;
   }
27
   #define MAX 256
29
   int main()
31 {
   int niz[MAX];
33   int n = 0;

35   /* Unos niza */
   printf("Unesi rastuce sortiran niz celih brojeva: ");
37   while (scanf("%d", &niz[n]) == 1)
       n++;

39   /* Stampanje rezultata */
41   printf("%d\n", prvi_veci_od_nule(niz, n));

43   return 0;
   }

```

#### Rešenje 3.8

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int prvi_manji_od_nule(int niz[], int n)
5 {
6     /* Granice pretrage */
7     int l = 0, d = n - 1;
8     int s;
9     /* Sve dok je leva manja od desne granice */
10    while (l <= d) {
11        /* Racuna se sredisnja pozicija */
12        s = (l + d) / 2;
13        /* Ako je broj na toj poziciji manji od nule, a eventualni njegov
14         prethodnik veci ili jednak nuli, pretraga se zavrшава */
15        if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
16            return s;
17        /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
18         niza */
19        if (niz[s] >= 0)
20            l = s + 1;
21        /* A inace leva */
22        else
23            d = s - 1;
24    }
25    return -1;
26 }
27
28 #define MAX 256
29
30 int main()
31 {
32     int niz[MAX];
33     int n = 0;
34
35     /* Unos niza */
36     printf("Unesi opadajuće sortiran niz celih brojeva: ");
37     while (scanf("%d", &niz[n]) == 1)
38         n++;
39
40     /* Stampanje rezultata */
41     printf("%d\n", prvi_manji_od_nule(niz, n));
42
43     return 0;
44 }
```

#### Rešenje 3.9

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



```
3  #include <stdlib.h>
5  unsigned int logaritam_a(unsigned int x)
6  {
7      /* Izlaz iz rekurzije */
8      if (x == 1)
9          return 0;
10     /* Rekurzivni korak */
11     return 1 + logaritam_a(x >> 1);
12 }
13 unsigned int logaritam_b(unsigned int x)
14 {
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
16     najvece vaznosti, tj. najlevlja. Pretragu se vrši od pozicije 0
17     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         /* Racuna se sredisnja pozicija */
23         s = (l + d) / 2;
24         /* Proverava se da li je na toj poziciji trazena jedinica */
25         if ((1 << s) <= x && (1 << (s + 1)) > x)
26             return s;
27         /* Pretraga desne polovine binarnog zapisa */
28         if ((1 << s) > x)
29             l = s - 1;
30         /* Pretraga leve polovine 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 }
```

```
    return 0;  
55 }
```

#### Rešenje 3.12

```
1  #include<stdio.h>  
2  #define MAX 256  
3  
4  /* Iterativna verzija funkcije koja sortira niz celih brojeva,  
5   primenom algoritma Selection Sort */  
6  void selectionSort(int a[], int n)  
7  {  
8      int i, j;  
9      int min;  
10     int pom;  
11  
12     /* U svakoj iteraciji ove petlje se pronalazi najmanji element  
13      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  
15      min, na kojoj se nalazio najmanji od gore navedenih elemenata.  
16      */  
17     for (i = 0; i < n - 1; i++) {  
18         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi  
19          najmanji od elemenata a[i],...,a[n-1]. */  
20         min = i;  
21         for (j = i + 1; j < n; j++)  
22             if (a[j] < a[min])  
23                 min = j;  
24         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako  
25          su (i) i min razliciti, inace je nepotrebno. */  
26         if (min != i) {  
27             pom = a[i];  
28             a[i] = a[min];  
29             a[min] = pom;  
30         }  
31     }  
32 }  
33  
34 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u  
35    sortiranom nizu celih brojeva */  
36 int najmanje_rastojanje(int a[], int n)  
37 {  
38     int i, min;  
39     min = a[1] - a[0];  
40     for (i = 2; i < n; i++)  
41         if (a[i] - a[i - 1] < min)  
42             min = a[i] - a[i - 1];  
43     return min;  
44 }  
45
```

```

47 int main()
{
    int i, a[MAX];

    /* Ucitavaju se elementi niza sve do kraja ulaza */
    i = 0;
    printf("Unesite elemente niza: ");
    while (scanf("%d", &a[i]) != EOF)
        i++;

    /* Sortiranje */
    selectionSort(a, i);

    /* Ispis rezultata */
    printf("%d\n", najmanje_rastojanje(a, i));

    return 0;
}
63

```

### Rešenje 3.13

```

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

4 #define MAX_DIM 128

6 /* Funkcija za sortiranje niza karaktera */
void selectionSort(char s[])
8 {
    int i, j, min;
    char pom;
    for (i = 0; s[i] != '\0'; i++) {
        min = i;
        for (j = i + 1; s[j] != '\0'; j++)
            if (s[j] < s[min])
                min = j;
        if (min != i) {
            pom = s[i];
            s[i] = s[min];
            s[min] = pom;
        }
    }
22 }

24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
26 {
    int i;

    /* Ako dve niske imaju razlicit broj karaktera onda one nisu
30     anagrami */

```

### 3 Algoritmi pretrage i sortiranja

---

```

32     if (strlen(s) != strlen(t))
        return 0;

34     /* Sortiramo niske */
    selectionSort(s);
    selectionSort(t);

36

38     /* Dve sortirane niske su anagrami ako i samo ako su jednake */
    for (i = 0; s[i] != '\0'; i++)
        if (s[i] != t[i])
            return 0;
    return 1;
42 }

44
46 int main()
47 {
    char s[MAX_DIM], t[MAX_DIM];

48

    /* Ucitavanje niski sa ulaza */
    printf("Unesite prvu nisku: ");
    scanf("%s", s);
    printf("Unesite drugu nisku: ");
    scanf("%s", t);

54

    /* Poziv funkcije */
    if (anagrami(s, t))
        printf("jesu\n");
    else
        printf("nisu\n");

60

    return 0;
62 }
```

#### Rešenje 3.14

```

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

3

   /* Funkcija za sortiranje niza */
5  void selectionSort(int s[], int n)
   {
7      int i, j, min;
      char pom;
9      for (i = 0; i < n; i++) {
          min = i;
11         for (j = i + 1; j < n; j++)
             if (s[j] < s[min])
13                 min = j;
          if (min != i) {
15              pom = s[i];
              s[i] = s[min];
          }
      }
   }
```

```

17     s[min] = pom;
18     }
19 }
20 }
21
22 /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
23    najvise puta pojavio u tom nizu */
24 int najvise_puta(int a[], int n)
25 {
26     int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
27     /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
28     for (i = 0; i < n; i = j) {
29         br_pojava = 1;
30         for (j = i + 1; j < n && a[i] == a[j]; j++)
31             br_pojava++;
32         /* Ispitivanje da li se do tog trenutka i-ti element pojavio
33            najvise puta u nizu */
34         if (br_pojava > max_br_pojava) {
35             max_br_pojava = br_pojava;
36             i_max_pojava = i;
37         }
38     }
39     /* Vraca se element koji se najvise puta pojavio u nizu */
40     return a[i_max_pojava];
41 }
42
43 int main()
44 {
45     int a[MAX_DIM], i;
46
47     /* Ucitavanje elemenata niza sve do kraja ulaza */
48     i = 0;
49     printf("Unesite elemente niza: ");
50     while (scanf("%d", &a[i]) != EOF)
51         i++;
52
53     /* Niz se sortira */
54     selectionSort(a, i);
55
56     /* Odredjuje se broj koji se najvise puta pojavio u nizu */
57     printf("%d\n", najvise_puta(a, i));
58
59     return 0;
60 }

```

### Rešenje 3.15

```

1 #include<stdio.h>
2 #define MAX_DIM 256
3
4 /* Funkcija za sortiranje niza */

```

### 3 Algoritmi pretrage i sortiranja

---

```
5 void selectionSort(int a[], int n)
6 {
7     int i, j, min, pom;
8     for (i = 0; i < n - 1; i++) {
9         min = i;
10        for (j = i + 1; j < n; j++)
11            if (a[j] < a[min])
12                min = j;
13        if (min != i) {
14            pom = a[i];
15            a[i] = a[min];
16            a[min] = pom;
17        }
18    }
19 }

21 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
22    u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
23    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 int main()
40 {
41     int a[MAX_DIM], n = 0, zbir, i;
42
43     /* Ucitava se trazeni zbir */
44     printf("Unesite trazeni zbir: ");
45     scanf("%d", &zbir);
46
47     /* Ucitavaju se elementi niza sve do kraja ulaza */
48     i = 0;
49     printf("Unesite elemente niza: ");
50     while (scanf("%d", &a[i]) != EOF)
51         i++;
52     n = i;
53
54     /* Sortira se niz */
55     selectionSort(a, n);
```

```

57     for (i = 0; i < n; i++)
58         /* Za i-ti element niza binarno se pretražuje da li se u ostatku
59            niza nalazi element koji sabran sa njim ima ucitanu vrednost
60            zbira */
61         if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
62             printf("da\n");
63             return 0;
64         }
65     printf("ne\n");
66
67     return 0;
68 }

```

### Rešenje 3.16

```

/* 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

```

```

/* Datoteka sort.c */
2
3 #include "sort.h"
4
5 /* Funkcija sortira niz celih brojeva metodom sortiranja izborom.
6    Ideja algoritma je sledeca: U svakoj iteraciji pronalazi se
7    najmanji element i premesta se na pocetak niza. Dakle, u prvoj
8    iteraciji, pronalazi se najmanji element, i dovodi na nultu mesto
9    u nizu. U i-toj iteraciji najmanjih i elemenata su vec na svojim
10   pozicijama, pa se od i+1 do n-1 elementa trazi najmanji, koji se
11   dovodi na i+1 poziciju. */
12 void selectionsort(int a[], int n)
13 {
14     int i, j;
15     int min;

```

### 3 Algoritmi pretrage i sortiranja

---

```
16  int pom;

18  /* U svakoj iteraciji ove petlje pronalazi se najmanji element
20     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
    poziciju i, dok se element na poziciji i premesta na poziciju
    min, na kojoj se nalazio najmanji od gore navedenih elemenata.
    */
22  for (i = 0; i < n - 1; i++) {
    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
24     najmanji od elemenata a[i],...,a[n-1]. */
    min = i;
26     for (j = i + 1; j < n; j++)
        if (a[j] < a[min])
28         min = j;

30     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
    su (i) i min razliciti, inace je nepotrebno. */
32     if (min != i) {
        pom = a[i];
34         a[i] = a[min];
        a[min] = pom;
36     }
    }
38 }

40 /* Funkcija sortira niz celih brojeva metodom sortiranja umetanjem.
    Ideja algoritma je sledeca: neka je na pocetku i-te iteracije niz
42     prvih i elemenata (a[0],a[1],...,a[i-1]) sortirano. U i-toj
    iteraciji treba element a[i] umetnuti na pravu poziciju medju
44     prvih i elemenata tako da se dobije niz duzine i+1 koji je
    sortiran. Ovo se radi tako sto se i-ti element najpre uporedi sa
46     njegovim prvim levim susedom (a[i-1]). Ako je a[i] vece, tada je
    on vec na pravom mestu, i niz a[0],a[1],...,a[i] je sortiran, pa
48     se moze preci na sledecu iteraciju. Ako je a[i-1] vece, tada se
    zamenjuju a[i] i a[i-1], a zatim se proverava da li je potrebno
50     dalje potiskivanje elementa u levo, poredeci ga sa njegovim novim
    levim susedom. Ovim uzastopnim premestanjem se a[i] umece na pravo
52     mesto u nizu. */
void insertionsort(int a[], int n)
54 {
    int i, j;

56     /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
    sortiran */
58     for (i = 1; i < n; i++) {

60         /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
        potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...a[i]
62         bude sortiran. Indeks j je trenutna pozicija na kojoj se
        element koji se umece nalazi. Petlja se zavrшава ili kada
64         element dodje do levog kraja (j==0) ili kada se naidje na
        element a[j-1] koji je manji od a[j]. */
66         for (j = i; j > 0; j--)
            if (a[j] < a[j-1])
                swap(a[j], a[j-1]);
            else
                break;
    }
```



```

68     for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
        int temp = a[j];
        a[j] = a[j - 1];
70         a[j - 1] = temp;
        }
72     }
    }
74
    /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
76     algoritma je sledeca: prolazi se kroz niz redom poredeci susedne
    elemente, i pri tom ih zamenjujuci ako su u pogresnom poretku.
78     Ovim se najveći element poput mehurica istiskuje na "povrsinu",
    tj. na krajnju desnu poziciju. Nakon toga je potrebno ovaj
80     postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih n-1
    elemenata niza bez poslednjeg koji je postavljen na pravu
82     poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
    kracim prefiksima niza, cime se jedan po jedan istiskuju
84     elementi na svoje prave pozicije. */
    void bubblesort(int a[], int n)
    {
86         int i, j;
88         int ind;

90         for (i = n, ind = 1; i > 1 && ind; i--)

92             /* Poput "mehurica" potiskuje se najveći element medju elementima
            od a[0] do a[i-1] na poziciju i-1 upoređujući susedne
94             elemente niza i potiskujući veci u desno */
            for (j = 0, ind = 0; j < i - 1; j++)
96                 if (a[j] > a[j + 1]) {
                    int temp = a[j];
98                     a[j] = a[j + 1];
                    a[j + 1] = temp;

100
                    /* Promenljiva ind registruje da je bilo premestanja. Samo u
102                     tom slucaju ima smisla ici na sledecu iteraciju, jer ako
                    nije bilo premestanja, znaci da su svi elementi vec u
104                     dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
                    niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
106                     ispravno, ali manje efikasano, jer bi se cesto nepotrebno
                    vrsila mnoga uporedjivanja, kada je vec jasno da je
108                     sortiranje zavrшено. */
                    ind = 1;
110                 }
            }
112
    /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
114     dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
    sastoji u tome da se kroz algoritam umetanja prolazi vise puta; u
116     prvom prolazu, umesto koraka 1 uzima se neki korak h koji je manji
    od n (sto omogucuje razmenu udaljenih elemenata) i tako se dobija
118     h-sortiran niz, tj. niz u kome su elementi na rastojanju h

```

### 3 Algoritmi pretrage i sortiranja

---

```
120     sortirani, mada susedni elementi to ne moraju biti. U drugom
122     prolazu kroz isti algoritam sprovodi se isti postupak ali za manji
124     korak h. Sa prolazima se nastavlja sve do koraka h = 1, u kome se
126     dobija potpuno sortirani niz. Izbor pocetne vrednosti za h, i
128     nacina njegovog smanjivanja menja u nekim slucajevima brzinu
130     algoritma, ali bilo koja vrednost ce rezultovati ispravnim
132     sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
134     vrednost 1. */
136 void shellsort(int a[], int n)
138 {
140     int h = n / 2, i, j;
142     while (h > 0) {
144         /* Insertion sort sa korakom h */
146         for (i = h; i < n; i++) {
148             int temp = a[i];
150             j = i;
152             while (j >= h && a[j - h] > temp) {
154                 a[j] = a[j - h];
156                 j -= h;
158             }
160             a[j] = temp;
162         }
164         h = h / 2;
166     }
168 }

170 #define MAX 1000000

172 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem. Sortiranje
174     se vrshi od elementa na poziciji l do onog na poziciji d. Na
176     pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a d je
178     jednako poslednjem validnom indeksu u nizu. Funkcija niz podeli na
180     dve polovine, levu i desnu, koje zatim rekurzivno sortira. Od ova
182     dva sortirana podniza, sortiran niz se dobija ucesljavanjem, tj.
184     istovremenim prolaskom kroz oba niza i izborom trenutnog manjeg
186     elementa koji se smesta u pomocni niz. Na kraju algoritma,
188     sortirani elementi su u pomocnom nizu, koji se kopira u originalni
190     niz. */
192 void mergesort(int a[], int l, int d)
194 {
196     int s;
198     static int b[MAX];          /* Pomocni niz */
200     int i, j, k;

202     /* Izlaz iz rekurzije */
204     if (l >= d)
206         return;

208     /* Odredjivanje sredisnjeg indeksa */
210     s = (l + d) / 2;

212     /* Rekurzivni pozivi */
```

```

172 mergesort(a, l, s);
    mergesort(a, s + 1, d);

174 /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
    niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
176 prolazi kroz pomocni niz b[] */
    i = 1;
178 j = s + 1;
    k = 0;

180 /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
182 while (i <= s && j <= d) {
    if (a[i] < a[j])
184         b[k++] = a[i++];
    else
186         b[k++] = a[j++];
}

188 /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive j
    iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
190 polovine niza */
192 while (i <= s)
    b[k++] = a[i++];

194 /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive i
    iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
196 polovine niza */
198 while (j <= d)
    b[k++] = a[j++];

200 /* Prepisuje se "ucesljani" niz u originalni niz */
202 for (k = 0, i = 1; i <= d; i++, k++)
    a[i] = b[k];
204 }

206 /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
208 {
    int tmp = a[i];
210 a[i] = a[j];
    a[j] = tmp;
212 }

214 /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r. Njena
    ideja sortiranja je izbor jednog elementa niza, koji se naziva
216 pivot, i koji se dovodi na svoje mesto. Posle ovog koraka, svi
    elementi levo od njega bice manji, a svi desno bice veci od njega.
218 Kako je pivot doveden na svoje mesto, da bi niz bio kompletno
    sortiran, potrebno je sortirati elemente levo (manje) od njega, i
220 elemente desno (vece). Kako su dimenzije ova dva podniza manje od
    dimenzije pocetnog niza koji je trebalo sortirati, ovaj deo moze
222

```

### 3 Algoritmi pretrage i sortiranja

```

    se uraditi rekurzivno. */
224 void quicksort(int a[], int l, int r)
{
226     int i, pivot_position;

228     /* Izlaz iz rekurziije -- prazan niz */
    if (l >= r)
230         return;

232
233     /* Particionisanje niza. Svi elementi na pozicijama levo od
234        pivot_position (izuzev same pozicije l) su strogo manji od
235        pivota. Kada se pronadje neki element manji od pivota, uvecava
236        se promenljiva pivot_position i na tu poziciju se premesta
237        nadjeni element. Na kraju ce pivot_position zaista biti pozicija
238        na koju treba smestiti pivot, jer ce svi elementi levo od te
239        pozicije biti manji a desno biti veci ili jednaki od pivota. */
240     pivot_position = l;
241     for (i = l + 1; i <= r; i++)
242         if (a[i] < a[l])
243             swap(a, ++pivot_position, i);

244
245     /* Postavljanje pivota na svoje mesto */
246     swap(a, l, pivot_position);

247
248     /* Rekurzivno sortiranje elementa manjih od pivota */
249     quicksort(a, l, pivot_position - 1);
250     /* Rekurzivno sortiranje elementa vecih od pivota */
251     quicksort(a, pivot_position + 1, r);
252 }
```

```

#include <stdio.h>
2 #include <stdlib.h>
#include <time.h>
4 #include "sort.h"

6 /* Maksimalna duzina niza */
#define MAX 1000000

8
9
10 int main(int argc, char *argv[])
{
11     /******
12         tip_sortiranja == 0 => selectionsort, (podrazumevano)
13         tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
14         tip_sortiranja == 2 => bubblesort, -b opcija komandne linije
15         tip_sortiranja == 3 => shellsort, -s opcija komandne linije
16         tip_sortiranja == 4 => mergesort, -m opcija komandne linije
17         tip_sortiranja == 5 => quicksort, -q opcija komandne linije
18     *****/
19     int tip_sortiranja = 0;
20     /******
21         tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
22     *****/
23 }
```

```

22     tip_niza == 1 => rastuce sortirani nizovi,   -r opcija
23     tip_niza == 2 => opadajuće sortirani nizovi, -o opcija
24     *****/
25     int tip_niza = 0;
26
27     /* Dimenzija niza koji se sortira */
28     int dimenzija;
29     int i;
30     int niz[MAX];
31
32     /* Provera argumenata komandne linije */
33     if (argc < 2) {
34         fprintf(stderr,
35             "Program zahteva bar 2 argumenta komandne linije!\n");
36         exit(EXIT_FAILURE);
37     }
38
39     /* Ocitanje opcija i argumenata prilikom poziva programa */
40     for (i = 1; i < argc; i++) {
41         /* Ako je u pitanju opcija... */
42         if (argv[i][0] == '-') {
43             switch (argv[i][1]) {
44                 case 'i':
45                     tip_sortiranja = 1;
46                     break;
47                 case 'b':
48                     tip_sortiranja = 2;
49                     break;
50                 case 's':
51                     tip_sortiranja = 3;
52                     break;
53                 case 'm':
54                     tip_sortiranja = 4;
55                     break;
56                 case 'q':
57                     tip_sortiranja = 5;
58                     break;
59                 case 'r':
60                     tip_niza = 1;
61                     break;
62                 case 'o':
63                     tip_niza = 2;
64                     break;
65                 default:
66                     printf("Pogresna opcija -%c\n", argv[i][1]);
67                     return 1;
68                     break;
69             }
70         }
71         /* Ako je u pitanju argument, onda je to duzina niza koji treba
72            da se sortira */
73         else {

```

### 3 Algoritmi pretrage i sortiranja

---

```
74     dimenzija = atoi(argv[i]);
75     if (dimenzija <= 0 || dimenzija > MAX) {
76         fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
77         exit(EXIT_FAILURE);
78     }
79 }
80 }

82 /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
83    niza dobijenom iz komandne linije. srandom() funkcija
84    obezbedjuje novi seed za pozivanje random funkcije, i kako
85    generisani niz ne bi uvek bio isti seed je postavljen na tekuce
86    vreme u sekundama od Nove godine 1970. random()%100 daje brojeve
87    izmedju 0 i 99 */
88 srandom(time(NULL));
89 if (tip_niza == 0)
90     for (i = 0; i < dimenzija; i++)
91         niz[i] = random();
92 else if (tip_niza == 1)
93     for (i = 0; i < dimenzija; i++)
94         niz[i] = i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
95 else
96     for (i = 0; i < dimenzija; i++)
97         niz[i] = i == 0 ? random() % 100 : niz[i - 1] - random() % 100;
98
99 /* Ispisivanje elemenata niza */
100 /******
101    Ovaj deo je iskomentarisan jer sledeci ispis ne treba da se nadje
102    na standardnom izlazu. Njegova svrha je samo bila provera da li
103    je niz generisan u skladu sa opcijama komandne linije.
104
105    printf("Niz koji sortiramo je:\n");
106    for (i = 0; i < dimenzija; i++)
107        printf("%d\n", niz[i]);
108    *****/

109
110 /* Sortiranje niza na odgovarajuci nacin */
111 if (tip_sortiranja == 0)
112     selectionsort(niz, dimenzija);
113 else if (tip_sortiranja == 1)
114     insertion sort(niz, dimenzija);
115 else if (tip_sortiranja == 2)
116     bubblesort(niz, dimenzija);
117 else if (tip_sortiranja == 3)
118     shellsort(niz, dimenzija);
119 else if (tip_sortiranja == 4)
120     mergesort(niz, 0, dimenzija - 1);
121 else
122     quicksort(niz, 0, dimenzija - 1);
123
124 /* Ispis elemenata niza */
```

```

126  /*****
128      Ovaj deo je iskomentarisano jer vreme potrebno za njegovo
130      izvršavanje ne bi trebalo da bude uključeno u vreme izmereno
132      programom time. Takođe, kako je svrha ovog programa da prikaže
134      vremena različitih algoritama sortiranja, dimenzije nizova će
136      biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
138      od toliko elemenata. Ovaj deo je koriscen u razvoju programa
140      zarad testiranja korektnosti.
142
143      printf("Sortiran niz je:\n");
144      for (i = 0; i < dimenzija; i++)
145          printf("%d\n", niz[i]);
146  *****/
147
148  return 0;
149  }

```

### Rešenje 3.17

```

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 slučaju da je dimenzija trećeg niza manja od neophodne,
9       funkcija vraća -1 */
10     if (dim3 < dim1 + dim2)
11         return -1;
12
13     /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog
14      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 {

```

### 3 Algoritmi pretrage i sortiranja

```
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 elemenata
38   */
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 */
55  dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);
56
57  /* Ispis niza */
58  for (k = 0; k < dim3; k++)
59      printf("%d ", niz3[k]);
60  printf("\n");
61
62  return 0;
63 }
```

#### Rešenje 3.18

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

int main(int argc, char *argv[])
{
    FILE *fin1 = NULL, *fin2 = NULL;
    FILE *fout = NULL;
    char ime1[11], ime2[11];
    char prezime1[16], prezime2[16];
    int kraj1 = 0, kraj2 = 0;

    /* Ako nema dovoljno argumenata komandne linije */
    if (argc < 3) {
        fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
        ;
        exit(EXIT_FAILURE);
    }
}
```



```

18     }
19
20     /* Otvaranje datoteke zadate prvim argumentom komandne linije */
21     fin1 = fopen(argv[1], "r");
22     if (fin1 == NULL) {
23         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
24         exit(EXIT_FAILURE);
25     }
26
27     /* Otvaranje datoteke zadate drugim argumentom komandne linije */
28     fin2 = fopen(argv[2], "r");
29     if (fin2 == NULL) {
30         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
31         exit(EXIT_FAILURE);
32     }
33
34     /* Otvaranje datoteke za upis rezultata */
35     fout = fopen("ceo-tok.txt", "w");
36     if (fout == NULL) {
37         fprintf(stderr,
38             "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
39         exit(EXIT_FAILURE);
40     }
41
42     /* Citanje narednog studenta iz prve datoteke */
43     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
44         kraj1 = 1;
45
46     /* Citanje narednog studenta iz druge datoteke */
47     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
48         kraj2 = 1;
49
50     /* Sve dok nije dostignut kraj neke datoteke */
51     while (!kraj1 && !kraj2) {
52         if (strcmp(ime1, ime2) < 0) {
53             /* Ime i prezime iz prve datoteke je leksikografski ranije, i
54                biva upisano u izlaznu datoteku */
55             fprintf(fout, "%s %s\n", ime1, prezime1);
56             /* Citanje narednog studenta iz prve datoteke */
57             if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
58                 kraj1 = 1;
59         } else {
60             /* Ime i prezime iz druge datoteke je leksikografski ranije, i
61                biva upisano u izlaznu datoteku */
62             fprintf(fout, "%s %s\n", ime2, prezime2);
63             /* Citanje narednog studenta iz druge datoteke */
64             if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
65                 kraj2 = 1;
66         }
67     }
68
69     /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj

```

### 3 Algoritmi pretrage i sortiranja

```
70     druge datoteke, onda ima jos studenata u prvoj datoteci, koje
       treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
72     */
       while (!kraj1) {
74         fprintf(fout, "%s %s\n", ime1, prezime1);
       if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
           kraj1 = 1;
76     }

78     /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
       datoteke, onda ima jos studenata u drugoj datoteci, koje treba
80     prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
       while (!kraj2) {
82         fprintf(fout, "%s %s\n", ime2, prezime2);
       if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
           kraj2 = 1;
84     }

86     /* Zatvaranje datoteka */
88     fclose(fin1);
       fclose(fin2);
90     fclose(fout);

92     return 0;
       }
```

#### Rešenje 3.19

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

5   #define MAX_BR_TACAKA 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 pocetka
15    (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;
    Tacka tmp;

47
    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) {
            tmp = t[i];
57            t[i] = t[min];
            t[min] = tmp;
59        }
    }
61 }

63
65 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
void sortiraj_po_y(Tacka t[], int n)
67 {
    int min, i, j;
    Tacka tmp;

69
    for (i = 0; i < n - 1; i++) {
        min = i;
71        for (j = i + 1; j < n; j++) {
            if (abs(t[j].y) < abs(t[min].y)) {
73                min = j;
            }
75        }
    }
}
```

```
    }
77     if (min != i) {
        tmp = t[i];
79         t[i] = t[min];
        t[min] = tmp;
81     }
    }
83 }

85 int main(int argc, char *argv[])
86 {
87     FILE *ulaz;
88     FILE *izlaz;
89     Tacka tacke[MAX_BR_TACAKA];
90     int i, n;
91
92     /* Proveravanje broja argumenata komandne linije: ocekuje se ime
93        izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
94        datoteke, tj. 4 argumenta */
95     if (argc != 4) {
96         fprintf(stderr,
97             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
98         return 0;
99     }
100
101     /* Otvaranje datoteke u kojoj su zadate tacke */
102     ulaz = fopen(argv[2], "r");
103     if (ulaz == NULL) {
104         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
105             argv[2]);
106         return 0;
107     }
108
109     /* Otvaranje datoteke u koju treba upisati rezultat */
110     izlaz = fopen(argv[3], "w");
111     if (izlaz == NULL) {
112         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
113             argv[3]);
114         return 0;
115     }
116
117     /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
118        koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
119        brojem i. */
120     i = 0;
121     while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
122         i++;
123     }
124
125     /* Ukupan broj procitanih tacaka */
126     n = i;
127 }
```

```

129  /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
    "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
    (karakter -), a karakter argv[1][1] odredjuje kriterijum
131  sortiranja */
    switch (argv[1][1]) {
133  case 'x':
        /* Sortiranje po vrednosti x koordinate */
135  sortiraj_po_x(tacke, n);
        break;
137  case 'y':
        /* Sortiranje po vrednosti y koordinate */
139  sortiraj_po_y(tacke, n);
        break;
141  case 'o':
        /* Sortiranje po udaljenosti od koordinatnog pocetka */
143  sortiraj_po_rastojanju(tacke, n);
        break;
145  }

147  /* Upisivanje dobijenog niza u izlaznu datoteku */
    for (i = 0; i < n; i++) {
149  fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
    }

151
    /* Zatvaranje otvorenih datoteka */
153  fclose(ulaz);
    fclose(izlaz);
155
    return 0;
157 }

```

### Rešenje 3.20

```

#include <stdio.h>
2  #include <string.h>
#include <stdlib.h>
4
#define MAX 1000
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;

```

### 3 Algoritmi pretrage i sortiranja

---

```

    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 a[i].prezime,...,a[n-1].prezime. */
48        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 */
int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64 {
    int i;
66    for (i = 0; i < n; i++)
        if (strcmp(a[i].ime, x->ime) == 0
68            && strcmp(a[i].prezime, x->prezime) == 0)
            return i;
70    return -1;

```

```

72 }
73
74 int main()
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;
78     int i, n;
79     FILE *fp = NULL;
80
81     /* Otvaranje datoteke */
82     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
83         fprintf(stderr,
84             "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
85         exit(EXIT_FAILURE);
86     }
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
95     /* Zatvaranje datoteke */
96     fclose(fp);
97
98     sort_ime(spisak1, n);
99
100     /******
101     Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
102     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
103
104     printf("Biracki spisak [uredjen prema imenima]:\n");
105     for(i=0; i<n; i++)
106         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107     *****/
108
109     sort_prezime(spisak2, n);
110
111     /******
112     Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
113     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
114
115     printf("Biracki spisak [uredjen prema prezimenima]:\n");
116     for(i=0; i<n; i++)
117         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
118     *****/
119
120     /* Linearno pretrazivanje nizova */
121     for (i = 0; i < n; i++)
122         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))

```

### 3 Algoritmi pretrage i sortiranja

```
        isti_rbr++;
124
/* Alternativno (efikasnije) resenje */
126 /*****
    for(i=0; i<n ;i++)
128         if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
            strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
130             isti_rbr++;
    *****/
132
/* Ispis rezultata */
134 printf("%d\n", isti_rbr);
136
exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.22

```
#include <stdio.h>
2 #include <string.h>
#include <ctype.h>
4
#define MAX_BR_RECII 128
6 #define MAX_DUZINA_RECII 32
8
/* Funkcija koja izracunava broj suglasnika u reci */
10 int broj_suglasnika(char s[])
{
12     char c;
    int i;
14     int suglasnici = 0;
    /* Prolaz karakter po karakter kroz zadatu nisku */
16     for (i = 0; s[i]; i++) {
        /* Ako je u pitanju slovo */
18         if (isalpha(s[i])) {
            /* Konvertuje se u veliko da bi bio pokriven slucaj i malih i
20             velikih suglasnika */
            c = toupper(s[i]);
22             /* Ukoliko slovo nije samoglasnik */
            if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U') {
24                 /* Uvecava se broj suglasnika */
                suglasnici++;
26             }
        }
28     }
    /* Vraca se izracunata vrednost */
30     return suglasnici;
}
32
/* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
```



```

34     duzini reci se mora proslediti zbog pravilnog upravljanja
        memorijom */
36 void sortiraj_reci(char reci[][MAX_DUZINA_REC], int n)
{
38     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
        duzina_j, duzina_min;
40     char tmp[MAX_DUZINA_REC];
    for (i = 0; i < n - 1; i++) {
42         min = i;
        for (j = i; j < n; j++) {
44             /* Prvo se uporedjuje broj suglasnika */
            broj_suglasnika_j = broj_suglasnika(reci[j]);
46             broj_suglasnika_min = broj_suglasnika(reci[min]);
            if (broj_suglasnika_j < broj_suglasnika_min)
48                 min = j;
            else if (broj_suglasnika_j == broj_suglasnika_min) {
50                 /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
                    se duzine */
52                 duzina_j = strlen(reci[j]);
                    duzina_min = strlen(reci[min]);

54                 if (duzina_j < duzina_min)
56                     min = j;
                    else
58                     /* A ako reci imaju i isti broj suglasnika i iste duzine,
                        uporedjuju se leksikografski */
60                     if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
                        min = j;
62                 }
            }
64             if (min != i) {
                strcpy(tmp, reci[min]);
66                 strcpy(reci[min], reci[i]);
                strcpy(reci[i], tmp);
68             }
        }
70    }

72 int main()
{
74     FILE *ulaz;
76     int i = 0, n;

78     /* Niz u kojem ce biti smestane reci. Prvi broj oznacava broj reci,
        a drugi maksimalnu duzinu pojedinačne reci */
80     char reci[MAX_BR_REC][MAX_DUZINA_REC];

82     /* Otvaranje datoteke niske.txt za citanje */
    ulaz = fopen("niske.txt", "r");
84     if (ulaz == NULL) {
        fprintf(stderr,

```

### 3 Algoritmi pretrage i sortiranja

---

```
86         "Greska prilikom otvaranja datoteke niske.txt!\n");
87     return 0;
88 }
89
90 /* Sve dok se moze procitati sledeca rec */
91 while (fscanf(ulaz, "%s", reci[i]) != EOF) {
92     /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
93        prekida se ucitavanje */
94     if (i == MAX_BR_RECI)
95         break;
96     /* Priprema brojac za narednu iteraciju */
97     i++;
98 }
99
100 /* n je duzina niza reci i predstavlja poslednju vrednost
101    koriscenog brojac */
102 n = i;
103 /* Poziv funkcije za sortiranje reci */
104 sortiraj_reci(reci, n);
105
106 /* Ispis sortiranog niza reci */
107 for (i = 0; i < n; i++) {
108     printf("%s ", reci[i]);
109 }
110 printf("\n");
111
112 /* Zatvaranje datoteke */
113 fclose(ulaz);
114
115 return 0;
116 }
```

#### 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 sa
16    traženim bar kodom */
17 int binarna_pretraga(Artikal a[], int n, long x)
```

```

19 {
20     int levi = 0;
21     int desni = n - 1;
22
23     /* Dokle god je indeks levi levo od indeksa desni */
24     while (levi <= desni) {
25         /* Racuna se sredisnji indeks */
26         int srednji = (levi + desni) / 2;
27         /* Ako je sredisnji element veci od trazenog, tada se trazeni
28            mora nalaziti u levoj polovini niza */
29         if (x < a[srednji].kod)
30             desni = srednji - 1;
31         /* Ako je sredisnji element manji od trazenog, tada se trazeni
32            mora nalaziti u desnoj polovini niza */
33         else if (x > a[srednji].kod)
34             levi = srednji + 1;
35         else
36             /* Ako je sredisnji element jednak trazenom, tada je artikal sa
37                bar kodom x pronadjen na poziciji srednji */
38             return srednji;
39     }
40     /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -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;

```

### 3 Algoritmi pretrage i sortiranja

```
FILE *fp = NULL;

71
/* Otvaranje datoteke */
73 if ((fp = fopen("artikli.txt", "r")) == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
75     exit(EXIT_FAILURE);
}

77
/* Ucitavanje artikala */
79 i = 0;
while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
81             asortiman[i].naziv, asortiman[i].proizvodjac,
             &asortiman[i].cena) == 4)
83     i++;

85 /* Zatvaranje datoteke */
fclose(fp);

87
n = i;

89
/* Sortira se celokupan asortiman prodavnice prema kodovima jer ce
91 pri kucanju racuna prodavac unositi kod artikla. Prilikom
kucanja svakog racuna pretrazuje se asortiman, da bi se utvrdila
93 cena artikla. Kucanje racuna obuhvata vise pretraga asortimana i
cilj je da ta operacija bude sto efikasnija. Zato se koristi
95 algoritam binarne pretrage prilikom pretrazivanja po kodu
artikla. Iz tog razloga, potrebno je da asortiman bude sortiran
97 po kodovima i to ce biti uradjeno primenom selection sort
algoritma. Sortiranje se vrši samo jednom na pocetku, ali se
99 zato posle artikli mogu brzo pretrazivati prilikom kucanja
proizvoljno puno racuna. Vreme koje se utrosi na sortiranje na
101 pocetku izvršavanja programa, kasnije se isplati jer se za
brojna trazanja artikla umesto linearne moze koristiti
103 efikasnija binarna pretraga. */
selection_sort(asortiman, n);

105
/* Ispis stanja u prodavnici */
107 printf
    ("Asortiman:\nKOD          Naziv artikla      Ime
    proizvodjaca      Cena\n");
109 for (i = 0; i < n; i++)
    printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
        asortiman[i].naziv, asortiman[i].proizvodjac,
        asortiman[i].cena);

113
kod = 0;
115 while (1) {
    printf("-----\n");
117    printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
    printf("- Za nov racun unesite kod artikla!\n\n");
119    /* Unos bar koda provog artikla sledeceg kupca */
    if (scanf("%ld", &kod) == EOF)
```

```

121     break;
122     /* Trenutni racun novog kupca */
123     racun = 0;
124     /* Za sve artikle trenutnog kupca */
125     while (1) {
126         /* Vrsi se njihov pronalazak u nizu */
127         if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
128             printf("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
129         } else {
130             printf("\tTrazili ste:\t%s %s %12.2f\n",
131                 asortiman[i].naziv, asortiman[i].proizvodjac,
132                 asortiman[i].cena);
133             /* I dodavanje na ukupan racun */
134             racun += asortiman[i].cena;
135         }
136         /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
137            nema vise artikla */
138         printf("Unesite kod artikla [ili 0 za prekid]: \t");
139         scanf("%ld", &kod);
140         if (kod == 0)
141             break;
142     }
143     /* Stampanje ukupnog racuna trenutnog kupca */
144     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
145 }
146
147 printf("Kraj rada kase!\n");
148
149 exit(EXIT_SUCCESS);
150 }

```

### Rešenje 3.24

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX 500
6
/* Struktura sa svim informacijama o pojedinacnom studentu */
8 typedef struct {
    char ime[20];
10    char prezime[25];
    int prisustvo;
12    int zadaci;
} Student;
14
/* Funkcija za sortiranje niza struktura po prezimenu leksikografski
16    rastuce */
void sort_ime_leksikografski(Student niz[], int n)
18 {

```

### 3 Algoritmi pretrage i sortiranja

---

```

20  int i, j;
    int min;
    Student pom;

22
    for (i = 0; i < n - 1; i++) {
24        min = i;
        for (j = i + 1; j < n; j++)
26            if (strcmp(niz[j].ime, niz[min].ime) < 0)
                min = j;

28        if (min != i) {
30            pom = niz[min];
            niz[min] = niz[i];
32            niz[i] = pom;
        }
34    }
}

36
/* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
38   zadataka opadajuće, a ukoliko neki studenti imaju isti broj
   uradjenih zadataka sortiraju se po dužini imena rastuće. */
40 void sort_zadatke_pa_imena(Student niz[], int n)
{
42     int i, j;
    int max;
    Student pom;
44     for (i = 0; i < n - 1; i++) {
46         max = i;
        for (j = i + 1; j < n; j++)
48             if (niz[j].zadaci > niz[max].zadaci)
                    max = j;
50             else if (niz[j].zadaci == niz[max].zadaci
                        && strlen(niz[j].ime) < strlen(niz[max].ime))
52                 max = j;
        if (max != i) {
54            pom = niz[max];
            niz[max] = niz[i];
56            niz[i] = pom;
        }
58    }
}

60
/* Funkcija za sortiranje niza struktura po broju casova na kojima
62   su bili opadajuće. Ukoliko neki studenti imaju isti broj casova,
   sortiraju se opadajuće po broju uradjenih zadataka, a ukoliko se
64   i po broju zadataka poklapaju, njihovo sortiranje ce biti po
   prezimenu opadajuće. */
66 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
{
68     int i, j;
    int max;
70     Student pom;
```

```

72     for (i = 0; i < n - 1; i++) {
        max = i;
        for (j = i + 1; j < n; j++)
74             if (niz[j].prisustvo > niz[max].prisustvo)
                max = j;
76             else if (niz[j].prisustvo == niz[max].prisustvo
                        && niz[j].zadaci > niz[max].zadaci)
                max = j;
78             else if (niz[j].prisustvo == niz[max].prisustvo
                        && niz[j].zadaci == niz[max].zadaci
                        && strcmp(niz[j].prezime, niz[max].prezime) > 0)
                max = j;
80             if (max != i) {
                pom = niz[max];
                niz[max] = niz[i];
                niz[i] = pom;
82             }
        }
88     }
}

90

92
93
94 int main(int argc, char *argv[])
95 {
    Student praktikum[MAX];
96     int i, br_studenata = 0;

97
98     FILE *fp = NULL;

99
100     /* Otvaranje datoteke za citanje */
101     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
102         fprintf(stderr, "Neuspješno otvaranje datoteke aktivnost.txt.\n");
103         exit(EXIT_FAILURE);
104     }

105
106     /* Ucitavanje sadržaja */
107     for (i = 0;
108          fscanf(fp, "%s%d", praktikum[i].ime,
                  praktikum[i].prezime, &praktikum[i].prisustvo,
109                  &praktikum[i].zadaci) != EOF; i++);
110
111     /* Zatvaranje datoteke */
112     fclose(fp);
113     br_studenata = i;

114
115     /* Kreiranje prvog spiska studenata po prvom kriterijumu */
116     sort_ime_leksikografski(praktikum, br_studenata);
117     /* Otvaranje datoteke za pisanje */
118     if ((fp = fopen("dat1.txt", "w")) == NULL) {
119         fprintf(stderr, "Neuspješno otvaranje datoteke dat1.txt.\n");
120         exit(EXIT_FAILURE);
121     }
122     /* Upis niza u datoteku */

```

### 3 Algoritmi pretrage i sortiranja

```
124     fprintf
125     (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
126     for (i = 0; i < br_studenata; i++)
127         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
128                 praktikum[i].prezime, praktikum[i].prisustvo,
129                 praktikum[i].zadaci);
130     /* Zatvaranje datoteke */
131     fclose(fp);
132
133     /* Kreiranje drugog spiska studenata po drugom kriterijumu */
134     sort_zadatke_pa_imena(praktikum, br_studenata);
135     /* Otvaranje datoteke za pisanje */
136     if ((fp = fopen("dat2.txt", "w")) == NULL) {
137         fprintf(stderr, "Neuspesno otvaranje datoteke dat2.txt.\n");
138         exit(EXIT_FAILURE);
139     }
140     /* Upis niza u datoteku */
141     fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
142     fprintf(fp, "pa po duzini imena rastuce:\n");
143     for (i = 0; i < br_studenata; i++)
144         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
145                 praktikum[i].prezime, praktikum[i].prisustvo,
146                 praktikum[i].zadaci);
147     /* Zatvaranje datoteke */
148     fclose(fp);
149
150     /* Kreiranje treceg spiska studenata po trecem kriterijumu */
151     sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum, br_studenata);
152     /* Otvaranje datoteke za pisanje */
153     if ((fp = fopen("dat3.txt", "w")) == NULL) {
154         fprintf(stderr, "Neuspesno otvaranje datoteke dat3.txt.\n");
155         exit(EXIT_FAILURE);
156     }
157     /* Upis niza u datoteku */
158     fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
159     fprintf(fp, "pa po broju zadataka,\n");
160     fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
161     for (i = 0; i < br_studenata; i++)
162         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
163                 praktikum[i].prezime, praktikum[i].prisustvo,
164                 praktikum[i].zadaci);
165     /* Zatvaranje datoteke */
166     fclose(fp);
167
168     return 0;
169 }
```

#### Rešenje 3.25

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



```
#include <string.h>
#define KORAK 10

/* Struktura koja opisuje jednu pesmu */
typedef struct {
    char *izvodjac;
    char *naslov;
    int broj_gledanja;
} Pesma;

/* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
   rad qsort funkcije) */
int uporedi_gledanost(const void *pp1, const void *pp2)
{
    Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

    return p2->broj_gledanja - p1->broj_gledanja;
}

/* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
   funkcije) */
int uporedi_naslove(const void *pp1, const void *pp2)
{
    Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

    return strcmp(p1->naslov, p2->naslov);
}

/* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
   qsort funkcije) */
int uporedi_izvodjace(const void *pp1, const void *pp2)
{
    Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

    return strcmp(p1->izvodjac, p2->izvodjac);
}

int main(int argc, char *argv[])
{
    FILE *ulaz;
    Pesma *pesme;                                /* Pokazivac na deo memorije za
                                                    cuvanje pesama */

    int alocirano_za_pesme;                       /* Broj mesta alociranih za pesme */
    int i;                                         /* Redni broj pesme cije se
                                                    informacije citaju */

    int n;                                         /* Ukupan broj pesama */
    int j, k;
    char c;
```

### 3 Algoritmi pretrage i sortiranja

---

```
56     int alocirano;                                /* Broj mesta alociranih za propratne
                                                    informacije o pesmama */
58     int broj_gledanja;

59     /* Priprema datoteke za citanje */
60     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
61     if (ulaz == NULL) {
62         printf("Greska pri otvaranju ulazne datoteke!\n");
63         return 0;
64     }

65     /* Citanje informacija o pesmama */
66     pesme = NULL;
67     alocirano_za_pesme = 0;
68     i = 0;

69     while (1) {

70         /* Proverava da li je dostignut kraj datoteke */
71         c = fgetc(ulaz);
72         if (c == EOF) {
73             /* Nema vise sadrzaja za citanje */
74             break;
75         } else {
76             /* Inace, vracamo procitani karakter nazad */
77             ungetc(c, ulaz);
78         }

79         /* Provera da li postoji dovoljno memorije za citanje nove pesme
80         */
81         if (alocirano_za_pesme == i) {

82             /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
83             novih KORAK mesta */
84             alocirano_za_pesme += KORAK;
85             pesme =
86                 (Pesma *) realloc(pesme,
87                                   alocirano_za_pesme * sizeof(Pesma));

88             /* Proverava da li je nova memorija uspesno realocirana */
89             if (pesme == NULL) {
90                 /* Ako nije ispisuje se obavestenje */
91                 printf("Problem sa alokacijom memorije!\n");
92                 /* I oslobadja sva memorija zauzeta do ovog koraka */
93                 for (k = 0; k < i; k++) {
94                     free(pesme[k].izvodjac);
95                     free(pesme[k].naslov);
96                 }
97                 free(pesme);
98                 return 0;
99             }
100         }
101     }
```

```
106     }
108
109     /* Ako jeste, nastavlja se sa citanjem pesama ... */
110     /* Cita se ime izvodjaca */
111
112     j = 0;                                /* Pozicija na koju treba smestiti
113                                           procitani karakter */
114     alocirano = 0;                        /* Broj alociranih mesta */
115     pesme[i].izvodjac = NULL;            /* Memoriya za smestanje procitanih
116                                           karaktera */
117
118     /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
119       izvodjaca) citaju se karakteri iz datoteke */
120     while ((c = fgetc(ulaz)) != ' ') {
121         /* Proverav da li postoji dovoljno memorije za smestanje
122            procitanog karaktera */
123         if (j == alocirano) {
124
125             /* Ako ne, ako je potrosena sva alocirana memorija, alocira
126                se novih KORAK mesta */
127             alocirano += KORAK;
128             pesme[i].izvodjac =
129                 (char *) realloc(pesme[i].izvodjac,
130                                 alocirano * sizeof(char));
131
132             /* Provera da li je nova alokacija uspesna */
133             if (pesme[i].izvodjac == NULL) {
134                 /* Ako nije oslobadja se sva memorija zauzeta do ovog
135                    koraka */
136                 for (k = 0; k < i; k++) {
137                     free(pesme[k].izvodjac);
138                     free(pesme[k].naslov);
139                 }
140                 free(pesme);
141                 /* I prekida sa izvorsavanjem programa */
142                 return 0;
143             }
144         }
145
146         /* Ako postoji dovoljno memorije, smestamo procitani karakter
147            */
148         pesme[i].izvodjac[j] = c;
149         j++;
150         /* I nastavlja se sa citanjem */
151     }
152
153     /* Upis terminirajuće nule na kraj reci */
154     pesme[i].izvodjac[j] = '\\0';
155
156     /* Preskace se karakter - */
```

```
158     fgetc(ulaz);
160     /* Preskace se razmak */
162     fgetc(ulaz);
164     /* Cita se naslov pesme */
166     j = 0;                                /* Pozicija na koju treba smestiti
                                           procitani karakter */
168     alocirano = 0;                        /* Broj alociranih mesta */
170     pesme[i].naslov = NULL;              /* Memoriya za smestanje procitanih
                                           karaktera */
172     /* Sve do zarez (koji se nalazi nakon naslova pesme) citaju se
       karakteri iz datoteke */
174     while ((c = fgetc(ulaz)) != ',') {
176         /* Provera da li postoji dovoljno memorije za smestanje
           procitanog karaktera */
178         if (j == alocirano) {
180             /* Ako ne, ako je potrosena sva alocirana memoriya, alocira
               se novih KORAK mesta */
182             alocirano += KORAK;
184             pesme[i].naslov =
186                 (char *) realloc(pesme[i].naslov,
188                                 alocirano * sizeof(char));
190             /* Provera da li je nova alokacija uspesna */
192             if (pesme[i].naslov == NULL) {
194                 /* Ako nije, oslobadja se sva memoriya zauzeta do ovog
                   koraka */
196                 for (k = 0; k < i; k++) {
198                     free(pesme[k].izvodjac);
200                     free(pesme[k].naslov);
202                 }
204                 free(pesme[i].izvodjac);
206                 free(pesme);
208                 /* I prekida izvršavanje programa */
210                 return 0;
212             }
214         }
216         /* Ako postoji dovoljno memorije, smesta se procitani karakter
           */
218         pesme[i].naslov[j] = c;
220         j++;
222         /* I nastavlja dalje sa citanjem */
224     }
226     /* Upisuje se terminirajuca nula na kraj reci */
228     pesme[i].naslov[j] = '\\0';
230     /* Preskace se razmak */
```

```
208     fgetc(ulaz);

210     /* Cita se broj gledanja */
    broj_gledanja = 0;

212

214     /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
    datoteke */
    while ((c = fgetc(ulaz)) != '\n') {
216         broj_gledanja = broj_gledanja * 10 + (c - '0');
    }
218     pesme[i].broj_gledanja = broj_gledanja;

220     /* Prelazi se na citanje sledece pesme */
    i++;
222 }

224 /* Informacija o broju procitanih pesama */
n = i;
226 /* Zatvaranje nepotrebne datoteke */
fclose(ulaz);

228

230 /* Analiza argumenta komandne linije */
if (argc == 1) {
232     /* Nema dodatnih opcija => sortiranje po broju gledanja */
    qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
} else {
234     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
        /* Sortiranje po naslovu */
236         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
    } else {
238         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
            /* Sortiranje po izvodjacu */
240             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
        } else {
242             printf("Nedozvoljeni argumenti!\n");
            free(pesme);
244             return 0;
        }
    }
246 }
}

248

250 /* Ispis rezultata */
for (i = 0; i < n; i++) {
    printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
252         pesme[i].broj_gledanja);
}

254

256 /* Oslobadjanje memorije */
for (i = 0; i < n; i++) {
    free(pesme[i].izvodjac);
258     free(pesme[i].naslov);
}
}
```

### 3 Algoritmi pretrage i sortiranja

---

```
260     free(pesme);
262     return 0;
}
```

#### Rešenje 3.28

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <search.h>
5
6  #define MAX 100
7
8  /* Funkcija poredjenja dva cela broja */
9  int compare_int(const void *a, const void *b)
10 {
11     /* Konvertovanje void pokazivaca u int pokazivace koji se zatim
12        dereferenciraju, a dobijeni int-ovi se oduzimaju i razlika
13        vraca. */
14
15     /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
16        oduzimanje treba izbegavati 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         return -1;
25     else
26         return 0;
27 }
28
29 int compare_int_desc(const void *a, const void *b)
30 {
31     /* Za obrnuti poredak treba samo oduzimati a od b */
32     /* return *((int *)b) - *((int *)a); */
33
34     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
35        funkcija */
36     return -compare_int(a, b);
37 }
38
39 int main()
40 {
41     size_t n;
42     int i, x;
43     int a[MAX], *p = NULL;
```

```

45  /* Unos dimenzije */
    printf("Uneti dimenziju niza: ");
47  scanf("%ld", &n);
    if (n > MAX)
49      n = MAX;

51  /* Unos elementa niza */
    printf("Uneti elemente niza:\n");
53  for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

55
    /* Sortiranje niza celih brojeva */
57  qsort(a, n, sizeof(int), &compare_int);

59  /* Prikaz sortiranog niz */
    printf("Sortirani niz u rastucem poretku:\n");
61  for (i = 0; i < n; i++)
        printf("%d ", a[i]);
63  putchar('\n');

65  /* Pretrazivanje niza */
    /* Vrednost koja ce biti trazena u nizu */
67  printf("Uneti element koji se trazi u nizu: ");
    scanf("%d", &x);

69
    /* Binarna pretraga */
71  printf("Binarna pretraga: \n");
    p = bsearch(&x, a, n, sizeof(int), &compare_int);
73  if (p == NULL)
        printf("Elementa nema u nizu!\n");
75  else
        printf("Element je nadjen na poziciji %ld\n", p - a);
77

    /* Linearna pretraga */
79  printf("Linearna pretraga (lfind): \n");
    p = lfind(&x, a, &n, sizeof(int), &compare_int);
81  if (p == NULL)
        printf("Elementa nema u nizu!\n");
83  else
        printf("Element je nadjen na poziciji %ld\n", p - a);
85
    return 0;
87 }

```

### Rešenje 3.29

```

1  #include <stdio.h>
2  #include <stdlib.h>
    #include <math.h>
4  #include <search.h>

```

### 3 Algoritmi pretrage i sortiranja

---

```
6 #define MAX 100

8 /* Funkcija racuna broj delilaca broja x */
9 int no_of_deviders(int x)
10 {
11     int i;
12     int br;

14     /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
15     if (x < 0)
16         x = -x;
17     if (x == 0)
18         return 0;
19     if (x == 1)
20         return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22     br = 2;
23     for (i = 2; i < sqrt(x); i++)
24         if (x % i == 0)
25             /* Ako i deli x onda su delioci: i, x/i */
26             br += 2;
27     /* Ako je broj x bas kvadrat, onda se iz petlje izašlo kada je
28        promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29        jos jednog delioca */
30     if (i * i == x)
31         br++;

32     return br;
33 }

34

36 /* Funkcija poredjenja dva cela broja po broju delilaca */
37 int compare_no_deviders(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = no_of_deviders(ak);
42     int n_d_b = no_of_deviders(bk);

44     if (n_d_a > n_d_b)
45         return 1;
46     else if (n_d_a < n_d_b)
47         return -1;
48     else
49         return 0;
50 }

52 int main()
53 {
54     size_t n;
55     int i;
56     int a[MAX];
```



```

58  /* Unos dimenzije */
    printf("Uneti dimenziju niza: ");
60  scanf("%ld", &n);
    if (n > MAX)
62      n = MAX;

64  /* Unos elementa niza */
    printf("Uneti elemente niza:\n");
66  for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

68

70  /* Sortiranje niza celih brojeva prema broju delilaca */
    qsort(a, n, sizeof(int), &compare_no_deviders);

72  /* Prikaz sortiranog niza */
    printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
74  for (i = 0; i < n; i++)
        printf("%d ", a[i]);
76  putchar('\n');

78  return 0;
}

```

### 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. Za
16  svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
    nalazi na adresi koja je za 4 veka od prve reci, a za 4 manja od
18  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 elemenata
22  koji trebaju biti uporedjeni, (npr. pri porecenu prve i poslednje
    reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
24  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
    nad njima.
26  *****/

```

```
int poredi_leksikografski(const void *a, const void *b)
28 {
    return strcmp((char *) a, (char *) b);
30 }

/* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
32 int poredi_duzine(const void *a, const void *b)
{
34     return strlen((char *) a) - strlen((char *) b);
36 }

38 int main()
40 {
    int i;
    size_t n;
    FILE *fp = NULL;
    42 char niske[MAX_NISKI][MAX_DUZINA];
    char *p = NULL;
    44 char x[MAX_DUZINA];

    /* Otvaranje datoteke */
    46 if ((fp = fopen("niske.txt", "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
        exit(EXIT_FAILURE);
    48 }

    /* Citanje sadrzaja datoteke */
    50 for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);

    /* Zatvaranje datoteke */
    52 fclose(fp);
    n = i;

    /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
       prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
       niski po duzini */
    54 qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);

    printf("Leksikografski sortirane niske:\n");
    56 for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
    printf("\n");

    /* Unos trazene niski */
    58 printf("Uneti trazenu nisku: ");
    scanf("%s", x);

    /* Binarna pretraga */
    60 /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
       je niz vec sortiran leksikografski. */
    62 p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
```

```

        &poredi_leksikografski);
80
    if (p != NULL)
82        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
                p, (p - (char *) niske) / MAX_DUZINA);
84    else
        printf("Niska nije pronadjena u nizu\n");
86
    /* Linearna pretraga */
88    p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
              &poredi_leksikografski);
90
    if (p != NULL)
92        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
                p, (p - (char *) niske) / MAX_DUZINA);
94    else
        printf("Niska nije pronadjena u nizu\n");
96
    /* Sortiranje po duzini */
98    qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
100
    printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
102        printf("%s ", niske[i]);
    printf("\n");
104
    exit(EXIT_SUCCESS);
106 }

```

### Rešenje 3.31

```

#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 pokazivaca na karaktere ovog potpisa
   char *niske[3];
   posle alokacije u main-u se moze graficki predstaviti ovako:
12
   | X | ----->   | 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
20

```

```
    reci. Njegov tip je char*.

22
    Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
24 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 se tako
26 moraju i kastovati. Da bi se leksikografski uporedili elementi X i
    Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
28 u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
    kastovani na odgovarajuci tip.
30 *****/
    int poredi_leksikografski(const void *a, const void *b)
32 {
    return strcmp(*(char **) a, *(char **) b);
34 }

36 /* 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 kastovati na
    char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
46 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
48 ovde samo treba kastovati i ne dereferencirati. */
    int poredi_leksikografski_b(const void *a, const void *b)
50 {
    return strcmp((char *) a, *(char **) b);
52 }

54 int main()
    {
56     int i;
    size_t n;
58     FILE *fp = NULL;
    char *niske[MAX_NISKI];
60     char **p = NULL;
    char x[MAX_DUZINA];

62     /* Otvaranje datoteke */
64     if ((fp = fopen("niske.txt", "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
66         exit(EXIT_FAILURE);
    }

68     /* Citanje sadrzaja datoteke */
70     i = 0;
    while (fscanf(fp, "%s", x) != EOF) {
72         /* Alociranje dovoljne memorije za i-tu nisku */
```

```

74     if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
75         fprintf(stderr, "Greska pri alociranju niske\n");
76         exit(EXIT_FAILURE);
77     }
78     /* Kopiranje procitane niske na svoje mesto */
79     strcpy(niske[i], x);
80     i++;
81 }
82
83 /* Zatvaranje datoteke */
84 fclose(fp);
85 n = i;
86
87 /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
88    prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
89    niske po duzini */
90 qsort(niske, n, sizeof(char *), &poredi_leksikografski);
91
92 printf("Leksikografski sortirane niske:\n");
93 for (i = 0; i < n; i++)
94     printf("%s ", niske[i]);
95 printf("\n");
96
97 /* Unos trazene niske */
98 printf("Uneti trazenu nisku: ");
99 scanf("%s", x);
100
101 /* Binarna pretraga */
102 p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
103 if (p != NULL)
104     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
105           *p, p - niske);
106 else
107     printf("Niska nije pronadjena u nizu\n");
108
109 /* Linearna pretraga */
110 p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
111 if (p != NULL)
112     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
113           *p, p - niske);
114 else
115     printf("Niska nije pronadjena u nizu\n");
116
117 /* Sortiramo po duzini */
118 qsort(niske, n, sizeof(char *), &poredi_duzine);
119
120 printf("Niske sortirane po duzini:\n");
121 for (i = 0; i < n; i++)
122     printf("%s ", niske[i]);
123 printf("\n");
124
125 /* Oslobadjanje zauzete memorije */

```

### 3 Algoritmi pretrage i sortiranja

---

```
126     for (i = 0; i < n; i++)
        free(niske[i]);
128     exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.32

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <search.h>
5
6 #define MAX 500
7
8 /* Struktura sa svim informacijama o pojedinacnom studentu */
9 typedef struct {
10     char ime[21];
11     char prezime[21];
12     int bodovi;
13 } Student;
14
15 /* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
16    istim brojem bodova se dodatno sortiraju leksikografski po
17    prezimenu */
18 int compare(const void *a, const void *b)
19 {
20     Student *prvi = (Student *) a;
21     Student *drugi = (Student *) b;
22
23     if (prvi->bodovi > drugi->bodovi)
24         return -1;
25     else if (prvi->bodovi < drugi->bodovi)
26         return 1;
27     else
28         /* Ako su jednaki po broju bodova, treba ih uporediti po
29            prezimenu */
30         return strcmp(prvi->prezime, drugi->prezime);
31 }
32
33 /* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
34    Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
35    parametar je element niza ciji se bodovi porede. */
36 int compare_zabsearch(const void *a, const void *b)
37 {
38     int bodovi = *(int *) a;
39     Student *s = (Student *) b;
40     return s->bodovi - bodovi;
41 }
42
43 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
```

```
45     Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
    parametar je element niza cije se prezime poredi. */
46 int compare_zalinearnaprezimena(const void *a, const void *b)
47 {
    char *prezime = (char *) a;
48     Student *s = (Student *) b;
    return strcmp(prezime, s->prezime);
51 }

53
54 int main(int argc, char *argv[])
55 {
    Student kolokvijum[MAX];
56     int i;
    size_t br_studenata = 0;
57     Student *nadjen = NULL;
    FILE *fp = NULL;
61     int bodovi;
    char prezime[21];
63
    /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64     informacija korisniku kako se program koristi i prekida se
    izvorsavanje. */
65     if (argc < 2) {
        fprintf(stderr,
66             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
                argv[0]);
71         exit(EXIT_FAILURE);
    }
73
    /* Otvaranje datoteke */
74     if ((fp = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
77         exit(EXIT_FAILURE);
    }
79
    /* Ucitavanje sadrzaja */
81     for (i = 0;
        fscanf(fp, "%s%d", kolokvijum[i].ime,
83             kolokvijum[i].prezime,
                &kolokvijum[i].bodovi) != EOF; i++);
85
    /* Zatvaranje datoteke */
86     fclose(fp);
    br_studenata = i;
89

    /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
90     studenata sa istim brojem bodova sortiranje vrši po prezimenu */
    qsort(kolokvijum, br_studenata, sizeof(Student), &compare);
93
    printf("Studenti sortirani po broju poena opadajuće, ");
95     printf("pa po prezimenu rastuće:\n");
```

### 3 Algoritmi pretrage i sortiranja

```
for (i = 0; i < br_studenata; i++)
    printf("%s %s %d\n", kolokvijum[i].ime,
           kolokvijum[i].prezime, kolokvijum[i].bodovi);

/* Pretrazivanje studenata po broju bodova se vrši binarnom
   pretragom jer je niz sortirani po broju bodova. */
printf("Unesite broj bodova: ");
scanf("%d", &bodovi);

nadjen =
    bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
           &compare_za_bsearch);

if (nadjen != NULL)
    printf
        ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
else
    printf("Nema studenta sa unetim brojem bodova\n");

/* Pretraga po prezimenu se mora vršiti linearno jer je niz
   sortiran po bodovima. */
printf("Unesite prezime: ");
scanf("%s", prezime);

nadjen =
    lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
          &compare_za_linearna_prezimeni);

if (nadjen != NULL)
    printf
        ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
else
    printf("Nema studenta sa unetim prezimenom\n");

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 }
```



```

11 }
13 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14 int anagrami(char s[], char t[])
15 {
16     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
17     if (strlen(s) != strlen(t))
18         return 0;
19
20     /* Sortiranje niski */
21     qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);
23
24     /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
25        suprotnom, nisu */
26     return !strcmp(s, t);
27 }
29 int main()
30 {
31     char s[MAX], t[MAX];
32
33     /* Unos niski */
34     printf("Unesite prvu nisku: ");
35     scanf("%s", s);
36     printf("Unesite drugu nisku: ");
37     scanf("%s", t);
38
39     /* Ispituje se da li su niske anagrami */
40     if (anagrami(s, t))
41         printf("jesu\n");
42     else
43         printf("nisu\n");
44
45     return 0;
46 }

```

### Rešenje 3.34

```

#include <stdio.h>
2 #include <string.h>
#include <stdlib.h>
4
#define MAX 10
6 #define MAX_DUZINA 32
7
8 /* Funkcija porenjenja */
9 int uporedi_niske(const void *pa, const void *pb)
10 {
11     return strcmp((char *) pa, (char *) pb);
12 }

```

```
14 int main()
15 {
16     int i, n;
17     char S[MAX][MAX_DUZINA];
18
19     /* Unos broja niski */
20     printf("Unesite broj niski:");
21     scanf("%d", &n);
22
23     /* Unos niza niski */
24     printf("Unesite niske:\n");
25     for (i = 0; i < n; i++)
26         scanf("%s", S[i]);
27
28     /* Sortiranje niza niski */
29     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
30
31     /******
32     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
33     sortiranih niski. Koriscen je samo u fazi testiranja programa.
34
35     printf("Sortirane niske su:\n");
36     for(i = 0; i < n; i++)
37         printf("%s ", S[i]);
38     *****/
39
40     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
41     niza biti jedna do druge */
42     for (i = 0; i < n - 1; i++)
43         if (strcmp(S[i], S[i + 1]) == 0) {
44             printf("ima\n");
45             return 0;
46         }
47
48     printf("nema\n");
49     return 0;
50 }
```

#### Rešenje 3.35

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 #define MAX 21
6
7 /* Struktura koja predstavlja jednog studenta */
8 typedef struct student {
9     char nalog[8];
10    char ime[MAX];
11 }
```

```
11  char prezime[MAX];
12  int poeni;
13 } Student;

15
16 /* Funkcija poredi studente prema broju poena, rastuce */
17 int uporedi_poeni(const void *a, const void *b)
18 {
19     Student s = *(Student *) a;
20     Student t = *(Student *) b;
21     return s.poeni - t.poeni;
22 }

23
24 /* Funkcija poredi studente prvo prema godini, zatim prema smeru i
25    na kraju prema indeksu */
26 int uporedi_nalog(const void *a, const void *b)
27 {
28     Student s = *(Student *) a;
29     Student t = *(Student *) b;
30     /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
31        broj indeksa */
32     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
33     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
34     char smer1 = s.nalog[1];
35     char smer2 = t.nalog[1];
36     int indeks1 =
37         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
38         s.nalog[6] - '0';
39     int indeks2 =
40         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
41         t.nalog[6] - '0';
42     if (godina1 != godina2)
43         return godina1 - godina2;
44     else if (smer1 != smer2)
45         return smer1 - smer2;
46     else
47         return indeks1 - indeks2;
48 }

49
50 int uporedi_bsearch(const void *a, const void *b)
51 {
52     /* Nalog studenta koji se trazi */
53     char *nalog = (char *) a;
54     /* Kljuc pretrage */
55     Student s = *(Student *) b;

56
57     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
58     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
59     char smer1 = nalog[1];
60     char smer2 = s.nalog[1];
61     int indeks1 =
62         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
```

```

;
63 int indeks2 =
    (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
65     s.nalog[6] - '0';
if (godina1 != godina2)
67     return godina1 - godina2;
else if (smer1 != smer2)
69     return smer1 - smer2;
else
71     return indeks1 - indeks2;
}

73
75 int main(int argc, char **argv)
{
    Student *nadjen = NULL;
77     char nalog_trazeni[8];
    Student niz_studenata[100];
79     int i = 0, br_studenata = 0;
    FILE *in = NULL, *out = NULL;

81
83     /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
        korisnik nije ispravno pozvao program i prijavljuje se greska.
        */
    if (argc != 2 && argc != 3) {
85         fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
        );
87         exit(EXIT_FAILURE);
    }

89
91     /* Otvaranje datoteke za citanje */
    in = fopen("studenti.txt", "r");
    if (in == NULL) {
93         fprintf(stderr,
            "Greska prilikom otvaranja datoteke studenti.txt!\n");
95         exit(EXIT_FAILURE);
    }

97
99     /* Otvaranje datoteke za pisanje */
    out = fopen("izlaz.txt", "w");
    if (out == NULL) {
101         fprintf(stderr,
            "Greska prilikom otvaranja datoteke izlaz.txt!\n");
103         exit(EXIT_FAILURE);
    }

105
107     /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
    while (fscanf
        (in, "%s %s %s %d", niz_studenata[i].nalog,
109         niz_studenata[i].ime, niz_studenata[i].prezime,
            &niz_studenata[i].poeni) != EOF)
111         i++;

```

```

113     br_studenata = i;

115     /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
116     if (strcmp(argv[1], "-p") == 0)
117         qsort(niz_studenata, br_studenata, sizeof(Student),
118             &uporedi_poeni);
119     /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
120     else if (strcmp(argv[1], "-n") == 0)
121         qsort(niz_studenata, br_studenata, sizeof(Student),
122             &uporedi_nalog);
123
124     /* Sortirani studenti se ispisuju u izlaznu datoteku */
125     for (i = 0; i < br_studenata; i++)
126         fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
127             niz_studenata[i].ime, niz_studenata[i].prezime,
128             niz_studenata[i].poeni);
129
130     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
131        studenta... */
132     if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
133         strcpy(nalog_trazeni, argv[2]);
134
135         /* ... pronalazi se student sa tim nalogom... */
136         nadjen =
137             (Student *) bsearch(nalog_trazeni, niz_studenata,
138                 br_studenata, sizeof(Student),
139                 &uporedi_bsearch);
140
141         if (nadjen == NULL)
142             printf("Nije nadjen!\n");
143         else
144             printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
145                 nadjen->prezime, nadjen->poeni);
146     }
147
148     /* Zatvaranje datoteka */
149     fclose(in);
150     fclose(out);
151
152     return 0;
153 }

```

### 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)

```

```
{
8  printf("Unesite elemente matrice po vrstama:\n");
   int i, j;

10
   for (i = 0; i < n; i++) {
12       for (j = 0; j < m; j++) {
           scanf("%d", &a[i][j]);
14       }
   }
16 }

18 /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
   kolona */
19 int zbir_vrste(int **a, int v, int m)
20 {
21     int i, zbir = 0;

22     for (i = 0; i < m; i++) {
23         zbir += a[v][i];
24     }
25     return zbir;
26 }

28 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
   osnovu zbira koriscenjem selection sort algoritma */
29 void sortiraj_vrste(int **a, int n, int m)
30 {
31     int i, j, min;

32     for (i = 0; i < n - 1; i++) {
33         min = i;
34         for (j = i + 1; j < n; j++) {
35             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
36                 min = j;
37             }
38         }
39         if (min != i) {
40             int *tmp;
41             tmp = a[i];
42             a[i] = a[min];
43             a[min] = tmp;
44         }
45     }
46 }

48 /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
   standardni izlaz */
49 void ispisi_matricu(int **a, int n, int m)
50 {
51     int i, j;

52     for (i = 0; i < n; i++) {
53         for (j = 0; j < m; j++) {
54             printf("%d ", a[i][j]);
55             if (j % 10 == 9) printf("\n");
56         }
57     }
58 }
```

```
60     for (j = 0; j < m; j++) {
61         printf("%d ", a[i][j]);
62     }
63     printf("\n");
64 }
65
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;
71
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++) {
79         /* Alocira se memorija */
80         a[i] = (int *) malloc(m * sizeof(int));
81         /* Proverava se da li je doslo do greske prilikom alokacije */
82         if (a[i] == NULL) {
83             /* Ako jeste, ispisuje se poruka */
84             fprintf(stderr, "Problem sa alokacijom memorije!\n");
85             /* I oslobadja memorija zauzeta do ovog koraka */
86             for (j = 0; j < i; j++) {
87                 free(a[j]);
88             }
89             free(a);
90             exit(EXIT_FAILURE);
91         }
92     }
93
94     return a;
95 }
96
97 /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije nxm */
98 void oslobodi_memoriju(int **a, int n, int m)
99 {
100     int i;
101     for (i = 0; i < n; i++) {
102         free(a[i]);
103     }
104     free(a);
105 }
106
107
108
```

### 3 Algoritmi pretrage i sortiranja

---

```
110 int main(int argc, char *argv[])
111 {
112     int **a;
113     int n, m;
114
115     /* Unos dimenzija matrice */
116     printf("Unesite dimenzije matrice: ");
117     scanf("%d %d", &n, &m);
118
119     /* Alokacija memorije */
120     a = alociraj_memoriju(n, m);
121
122     /* Ucitavanje elementa matrice */
123     ucitaj_matricu(a, n, m);
124
125     /* Poziv funkcije koja sortira vrste matrice prema zbiru */
126     sortiraj_vrste(a, n, m);
127
128     /* Ispis rezultujuce matrice */
129     printf("Sortirana matrica je:\n");
130     ispisi_matricu(a, n, m);
131
132     /* Oslobadjanje memorije */
133     oslobodi_memoriju(a, n, m);
134
135     return 0;
136 }
```



## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

**Zadatak 4.1** Napisati biblioteku za rad sa jednostruko povezanom listom, čiji čvorovi sadrže cele brojeve.

- (a) Definirati strukturu `Cvor` kojom se predstavlja čvor liste. Treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.
- (b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje polja novog čvora i vraća njegovu adresu.
- (c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.
- (d) Napisati funkciju `Cvor* pronadji_poslednji(Cvor* glava)` koja pronalazi poslednji čvor u listi.
- (e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste.
- (f) Napisati funkciju `Cvor* pronadji_mesto_umetanja(Cvor* glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.

- (g) Napisati funkciju `void dodaj_iza(Cvor* tekuci, Cvor* novi)` koja uvezuje u postojeću listu čvor `novi` iza čvora `tekuci`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor* glava)` koja ispisuje čvorove liste, uokvirene zagradama `[, ]` i međusobno razdvojene zapetama.
- (j) Napisati funkciju `Cvor* pretrazi_listu(Cvor* glava, int broj)` koja proverava da li se u listi nalazi čvor čija se vrednost zadaje kao argument funkcije.
- (k) Napisati funkciju `Cvor* pretrazi_sortiranu_listu(Cvor* glava, int broj)` koja proverava da li se u listi nalazi čvor čija se vrednost zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenoj listi.
- (l) Napisati funkciju `void obrisi_cvor(Cvor** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost koja se zadaje kao argument funkcije.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost koja se zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenoj listi.
- (n) Napisati funkciju `void oslobodi_listu(Cvor** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

NAPOMENA: *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste 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.

- (1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi, i na ekran se ispisuje rezultat pretrage.

*Primer 1*

```

Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj unesite CTRL+D)
2
Lista: [2]
3
Lista: [3, 2]
14
Lista: [14, 3, 2]
5
Lista: [5, 14, 3, 2]
3
Lista: [3, 5, 14, 3, 2]
3
Lista: [3, 3, 5, 14, 3, 2]
17
Lista: [17, 3, 3, 5, 14, 3, 2]
3
Lista: [3, 17, 3, 3, 5, 14, 3, 2]
1
Lista: [1, 3, 17, 3, 3, 5, 14, 3, 2]
9
Lista: [9, 1, 3, 17, 3, 3, 5, 14, 3, 2]

Unosite broj koji se trazi u listi: 17
Trazeni broj 17 je u listi!

```

*Primer 2*

```

Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj unesite CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [35, 14, 23]

Unosite broj koji se trazi u listi: 8
Broj 8 se ne nalazi u listi!

```

*Primer 3*

```

Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj unesite CTRL+D)

Unosite broj koji se trazi u listi: 1
Broj 1 se ne nalazi u listi!

```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj

## 4 Dinamičke strukture podataka

---

čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

### *Primer 1*

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj unesite CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
5
Lista: [2, 3, 14, 5]
3
Lista: [2, 3, 14, 5, 3]
3
Lista: [2, 3, 14, 5, 3, 3]
17
Lista: [2, 3, 14, 5, 3, 3, 17]
3
Lista: [2, 3, 14, 5, 3, 3, 17, 3]
1
Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1]
3
Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1, 3]

Unosite broj koji se briše iz liste: 3
Lista nakon brisanja: [2, 14, 5, 17, 1]
```

### *Primer 2*

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj unesite CTRL+D)
23
Lista: [23]
14
Lista: [23, 14]
35
Lista: [23, 14, 35]

Unosite broj koji se briše iz liste: 3
Lista nakon brisanja: [23, 14, 35]
```

### *Primer 3*

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj unesite CTRL+D)

Unosite broj koji se briše iz liste: 12
Lista nakon brisanja: []
```

- (3) U glavnom programu se učitani celi brojevi dodaju u listu tako da je vrednosti budu uređene u neopadajućem poretku. Unosi ceo broj koji se traži u unetoj listi, i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja cvora liste koristiti činjenicu da je lista uređena.*

### Primer 1

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unesite brojeve: (za kraj unesite CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
5
Lista: [2, 3, 5, 14]
3
Lista: [2, 3, 3, 5, 14]
3
Lista: [2, 3, 3, 3, 5, 14]
17
Lista: [2, 3, 3, 3, 5, 14, 17]
3
Lista: [2, 3, 3, 3, 3, 5, 14, 17]
1
Lista: [1, 2, 3, 3, 3, 3, 5, 14, 17]
9
Lista: [1, 2, 3, 3, 3, 3, 5, 9, 14, 17]

Unesite broj koji se trazi u listi: 5
Trazeni broj 5 je u listi!

Unesite broj koji se brise iz liste: 3
Lista nakon brisanja: [1, 2, 5, 9, 14, 17]
```

### Primer 3

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unesite brojeve: (za kraj unesite CTRL+D)

Unesite broj koji se trazi u listi: 1
Broj 1 se ne nalazi u listi!

Unesite broj koji se brise iz liste: 12
Lista nakon brisanja: []
```

### Primer 2

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unesite brojeve: (za kraj unesite CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [14, 23, 35]

Unesite broj koji se trazi u listi: 8
Broj 8 se ne nalazi u listi!

Unesite broj koji se brise iz liste: 3
Lista nakon brisanja: [14, 23, 35]
```

[Rešenje 4.1]

**Zadatak 4.2** Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekurzivno. NAPOMENA: *Koristiti iste main programe i upotrebe programa iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva, koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

- (a) Napisati funkciju `void obrisi_tekuci(Cvor** adresa_glave, Cvor* tekuci)` koja iz liste čija se glava nalazi na adresi `adresa_glave` briše čvor na koji pokazuje pokazivač `tekuci`.
- (b) Napisati funkciju `void ispisi_listu_u_nazad(Cvor* glava)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. NAPOMENA: *Koristiti iste main programe i upotrebe programa iz zadatka 4.1. Ove programe dopuniti pozivom funkcije koja ispisuje listu u nazad.*

[Rešenje 4.3]

**Zadatak 4.4** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

*Test 1*

```

Poziv: ./a.out

IZRAZ.TXT
{[23 + 5344] * (24 - 234)} - 23

IZLAZ:
  Zagrade su ispravno uparene.

```

*Test 2*

```

Poziv: ./a.out

IZRAZ.TXT
{[23 + 5] * (9 * 2)} - {23}

IZLAZ:
  Zagrade su ispravno uparene.

```

*Test 3*

```

Poziv: ./a.out

IZRAZ.TXT
{[2 + 54] / (24 * 87)} + (234 + 23)

IZLAZ:
  Zagrade nisu ispravno uparene.

```

*Test 3*

```

Poziv: ./a.out

IZRAZ.TXT
{(2 - 14) / (23 + 11)} * (2 + 13)

IZLAZ:
  Zagrade nisu ispravno uparene.

```

*Test 4*

```

Poziv: ./a.out

IZRAZ.TXT
  Datoteka je prazna.

IZLAZ:
  Zagrade su ispravno uparene.

```

*Test 5*

```

Poziv: ./a.out

IZRAZ.TXT
  Datoteka ne postoji.

IZLAZ:
  Greska prilikom otvaranja
  datoteke izraz.txt!

```

[Rešenje 4.4]

**Zadatak 4.5** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: Za rešavanje problema koristiti stek implementiran preko liste čiji su čvorovi HTML etikete.

*Test 1*

```

Poziv: ./a.out datoteka.html

DATOTEKA.HTML
<html>
<head>
<title>Primer</title>
</head>
<body>
</body>

IZLAZ:
  Etikete nisu pravilno uparene
  (etiketa <html> nije zatvorena)

```

*Test 2*

```

Poziv: ./a.out datoteka.html

DATOTEKA.HTML
  Datoteka ne postoji.

IZLAZ:
  Greska prilikom otvaranja
  datoteke datoteka.html.

```

### Test 3

```
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:
Etikete su pravilno uparene!
```

### Test 4

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
<html>
<head>
<title>Primer</title>
</head>
<body>
</html>

IZLAZ:
Etikete nisu pravilno uparene
(nadjena etiketa </html>, a poslednja
otvorena etiketa je <body>)
```

### Test 5

```
Poziv: ./a.out

IZLAZ:
Greska! Program se poziva
sa: ./a.out datoteka.html!
```

### Test 6

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
Datoteka je prazna.

IZLAZ:
Etikete su pravilno uparene!
```

[Rešenje 4.5]

**Zadatak 4.6** Napisati program kojim se simulira rad jednog šaltera na kojem se prvo kod službenika zakazuju termini, a potom službenik uslužuje korisnike. Službenik evidentira korisničke JMBG brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (EOF). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije, tj. službeniku se postavlja pitanje **Da li korisnika vracate na kraj reda?** i ukoliko on da odgovor **Da**, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije **Da**, tada službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke `izvestaj.txt`. Ova datoteka, za svaki obrađen zahtev, sadrži `jmbg` i zahtev usluženog korisnika. Posle svakog 5 usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nezvezano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*



## Primer 1

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Sluzbenik evidentira korisnicke zahteve unosenjem
njihovog JMBG broja i opisa potrebne usluge:
Novi zahtev [CTRL+D za kraj]
JMBG: 1234567890123
Opis problema: Otvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 2345678901234
Opis problema: Podizanje novca

Novi zahtev [CTRL+D za kraj]
JMBG: 3456789012345
Opis problema: Reklamacija

Novi zahtev [CTRL+D za kraj]
JMBG: 4567890123456
Opis problema: Zatvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG:

Sledeci je korisnik sa JMBG brojem: 1234567890123
sa zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG brojem: 2345678901234
sa zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 3456789012345
sa zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG brojem: 4567890123456
sa zahtevom: Zatvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 1234567890123
sa zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 3456789012345
sa zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

IZVESTAJ.TXT
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 4567890123456 Zahtev: Zatvaranje racuna
JMBG: 1234567890123 Zahtev: Otvaranje racuna
JMBG: 3456789012345 Zahtev: Reklamacija
```

[Rešenje 4.6]

## 4 Dinamičke strukture podataka

**Zadatak 4.7** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smestati 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 3

```
POZIV: ./a.out

IZLAZ:
Greska! Program se poziva
sa: ./a.out datoteka.html!
```

### Test 2

```
POZIV: ./a.out datoteka.html

DATOTEKA.HTML
Datoteka ne postoji.

IZLAZ:
Greska prilikom otvaranja
datoteke datoteka.html.
```

### Test 4

```
POZIV: ./a.out datoteka.html

DATOTEKA.HTML
Datoteka je prazna.
```

[Rešenje 4.7]

**Zadatak 4.8** U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima, koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih studenata, sa standardnog ulaza unose se, jedan po jedan, indeksi

studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku da ili ne, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). UPUTSTVO: *Dodavanje novog studenta izdvojiti u funkciju void dodaj\_na\_pocetak\_liste(Cvor\*\* glava, char\* broj\_indeksa, char\* ime, char\* prezime). Napisati rekursivnu funkciju Cvor\* pretrazi\_listu(Cvor\* glava, char\* broj\_indeksa) koja određuje da li student sa zadatim brojem indeksa pripada listi ili ne. Nakon završenog pretraživanja rekursivnom funkcijom void oslobodi\_listu(Cvor\*\* glava) osloboditi memoriju koju je lista sa studentima zauzimala. Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

#### Primer 1

```
POZIV: ./a.out studenti.txt

STUDENTI.TXT
123/2014 Marko Lukic
3/2014 Ana Sokic
43/2013 Jelena Ilic
41/2009 Marija Zaric
13/2010 Milovan Lazic

INTERAKCIJA PROGRAMA:
3/2014 da: Ana Sokic
235/2008 ne
41/2009 da: Marija Zaric
```

#### Primer 2

```
POZIV: ./a.out studenti.txt

STUDENTI.TXT
Datoteka je prazna.

INTERAKCIJA PROGRAMA:
3/2014 ne
235/2008 ne
41/2009 ne
```

#### Primer 3

```
POZIV: ./a.out

IZLAZ:
Greska! Program se poziva sa:
./a.out studenti.txt!
```

#### Primer 4

```
POZIV: ./a.out studenti.txt

STUDENTI.TXT
Datoteka ne postoji.

IZLAZ:
Greska prilikom otvaranja datoteke
studenti.txt.
```

[Rešenje 4.8]

**Zadatak 4.9** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove čvorove, već da samo postojće čvorove preraspodeli. 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. NAPOMENA: *Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

### 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 dat1.txt dat2.txt

DAT1.TXT
  2 4 6 10 15

DAT2.TXT
  Datoteka ne postoji.

IZLAZ:
  Greska prilikom otvaranja datoteke
  dat2.txt.
```

### Test 3

```
POZIV: ./a.out dat1.txt dat2.txt

DAT1.TXT
  Datoteka ne postoji.

DAT2.TXT
  5 6 11 12 14 16

IZLAZ:
  Greska prilikom otvaranja datoteke
  dat1.txt.
```

### Test 4

```
POZIV: ./a.out dat1.txt dat2.txt

DAT1.TXT
  2 4 6 10 15

DAT2.TXT
  Datoteka je prazna.

IZLAZ:
  2 4 6 10 15
```

### Test 5

```
POZIV: ./a.out dat1.txt dat2.txt

DAT1.TXT
  Datoteka je prazna.

DAT2.TXT
  5 6 11 12 14 16

IZLAZ:
  5 6 11 12 14 16
```

### Test 6

```
POZIV: ./a.out

IZLAZ:
  Greska! Program se poziva sa:
  ./a.out dat1.txt dat2.txt!
```

### Test 7

```
POZIV: ./a.out dat1.txt

IZLAZ:
  Greska! Program se poziva sa:
  ./a.out dat1.txt dat2.txt!
```

[Rešenje 4.9]

**Zadatak 4.10** Date su dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od tih lista formira novu listu L koja sadrži alternirajući raspoređene čvorove lista L1 i L2 (prvi čvor iz L1, prvi čvor iz L2, drugi čvor L1, drugi čvor L2, itd). Ne formirati nove čvorove, već samo postojeće čvorove rasporediti

u jednu listu. Prva lista se učitava iz datoteke čije se ime 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.

NAPOMENA: *Iskoristiti testove 2 - 7 za zadatak 4.9.*

#### 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
```

**Zadatak 4.11** Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

- Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.
- Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

Napisati program koji u datoteku `Rezultat.txt` upisuje nađeni strogo rastući podniz.

#### 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
Datoteka ne postoji.

IZLAZ:
REZULTAT.TXT
Greska prilikom otvaranja
datoteke brojevi.txt.
```

#### Test 3

```
Poziv: ./a.out

BROJEVI.TXT
Datoteka je prazna.

IZLAZ:
REZULTAT.TXT
Rezultat.txt ce biti prazna.
```

**Zadatak 4.12** 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. Odbrojava 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

## 4 Dinamičke strukture podataka

---

izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: Pri implementaciji koristiti jednostruko povezanu kružnu listu.

### Test 1

```
Poziv: ./a.out
Ulaz:
  5 3
Izlaz:
  3 1 5 2 4
```

### Test 2

```
Poziv: ./a.out
Ulaz:
  8 4
Izlaz:
  4 8 5 2 1 3 7 6
```

### Test 3

```
Poziv: ./a.out
Ulaz:
  3 8
Izlaz:
  n mora biti uvek vece
  od k, a 3 < 8!
```

### Test 4

```
Poziv: ./a.out
Ulaz:
  0 0
Izlaz:
  Broj plesaca mora biti
  veci od 0!
```

**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 počevoši 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 odbrojavalo u smeru kazaljke na satu sada će se obrojavati 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: Pri implementaciji koristiti dvostruko povezanu kružnu listu. NAPOMENA: Iskoristiti 3. i 4. test za 4.12. zadatak.

### Test 1

```
Poziv: ./a.out
Ulaz:
  5 3
Izlaz:
  3 5 4 2 1
```

### Test 2

```
Poziv: ./a.out
Ulaz:
  8 4
Izlaz:
  4 8 5 7 6 3 2 1
```

## 4.2 Stabla

**Zadatak 4.14** Napisati program za rad sa binarnim pretraživačkim stablima.

- (a) Definirati 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.
- (b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alokira 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`.
- (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`.

## 4 Dinamičke strukture podataka

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 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 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 4.14]

**Zadatak 4.15** 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 pretraživačkog stabla uređenog leksikografski po 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 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 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 Test 3

```
Poziv: ./a.out
Izlaz:
  Nedostaje ime ulazne datoteke!
```



[Rešenje 4.15]

**Zadatak 4.16** 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.

*Test 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
  Broj je: 011/3240-987
  Unesite ime i prezime: Marko Markovic
  Broj nije u imeniku!
  Unesite ime i prezime: KRAJ
```

[Rešenje 4.16]

**Zadatak 4.17** 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 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 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9
```

[Rešenje 4.17]

\* **Zadatak 4.18** 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.`

### Test 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:
  Slavljenik: Jovana Milic 05.05.
Ulaz:
  Unesite datum: CTRL+D
```

[Rešenje 4.18]

**Zadatak 4.19** 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. *NAPOMENA: Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka `stabla.h` predstavljati popis funkcija biblioteke, a datoteka `stabla.c` njihove implementacije. Programme koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

*Test 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 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.
```

[Rešenje 4.19]

\* **Zadatak 4.20** 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. 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 Test 1*

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 1 7 8 9 2 2
  Drugo stablo: 3 9 6 11 1
Izlaz:
  Unija: 1 1 2 2 3 6 7 8 9 9 11
  Presek: 1 9
  Razlika: 2 2 7 8
```

*Test Test 2*

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 2 7 5
  Drugo stablo: 4 3 3 7
Izlaz:
  Unija: 2 3 3 4 5 7 7 11
  Presek: 7
  Razlika: 2 5 11
```

[Rešenje 4.20]

**Zadatak 4.21** 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 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 Test 2*

```
Poziv: ./a.out
Ulaz:
  n: 55
Izlaz:
  Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

**Zadatak 4.22** 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 Test 1*

```

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
najveci 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

```

[Rešenje 4.22]

**Zadatak 4.23** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

*Test 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

```

*Test Test 2*

```

Poziv: ./a.out
Ulaz:
6 11 8 3 -2
Izlaz:
0.nivo: 6
1.nivo: 3 11
2.nivo: -2 8

```

[Rešenje 4.23]

\* **Zadatak 4.24** 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 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 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 4.25** AVL-stablo je binarno pretraživačko stablo kod kojeg 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 Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 2 11 16 1 13
Izlaz:
  7
```

### Test Test 2

```
Poziv: ./a.out
Ulaz:
  16 30 40 24 10 18 45 22
Izlaz:
  6
```

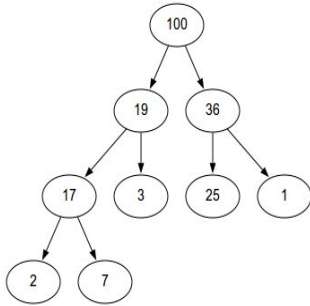
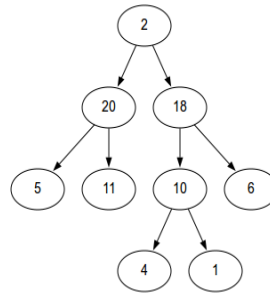
[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo se naziva HEAP ako 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 kreira stablo kao na slici ..., poziva funkciju `heap` i ispisuje rezultat na standardnom izlazu.

[Rešenje 4.26]

**Zadatak 4.27** Dato je binarno 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.

Slika 4.1: *zadatak 4.27*Slika 4.2: *zadatak 4.23*

- (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 gore navedene funkcije nad stablom zadatim slikom

...

## 4.3 Rešenja

### Rešenje 4.1

```

1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
6  typedef struct cvor {
   int vrednost;
8   struct cvor *sledeci;
   } Cvor;

10 Cvor *napravi_cvor(int broj);

12 void oslobodi_listu(Cvor ** adresa_glave);

14 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

16 Cvor *pronadji_poslednji(Cvor * glava);

18 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

20 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
  
```

```
22 void dodaj_iza(Cvor * tekuci, Cvor * novi);
24 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
26 Cvor *pretrazi_listu(Cvor * glava, int broj);
28 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
30 void obrisi_cvor(Cvor ** adresa_glave, int broj);
32 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
34 void ispisi_listu(Cvor * glava);
36 #endif
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
6    cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
7    novom cvoru postavlja na NULL. Funkcija vraca pokazivac na
8    novokreirani cvor ili NULL ako alokacija nije uspesno
9    izvrшена. */
10 Cvor *napravi_cvor(int broj)
11 {
12     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
13     if (novi == NULL)
14         return NULL;
15
16     novi->vrednost = broj;
17     novi->sledeci = NULL;
18     return novi;
19 }
20
21 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove
22    liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
23 void oslobodi_listu(Cvor ** adresa_glave)
24 {
25     Cvor *pomocni = NULL;
26
27     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
28     while (*adresa_glave != NULL) {
29         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
30            osloboditi cvor koji predstavlja glavu liste */
31         pomocni = (*adresa_glave)->sledeci;
32         free(*adresa_glave);
33         /* Sledeci cvor je nova glava liste. */
34         *adresa_glave = pomocni;
35     }
36 }
```



```

}
37
/* Funkcija dodaje broj na pocetak liste. Kreira novi cvor
39   koriscenjem funkcije napravi_cvor i uvezuje ga na pocetak.
   Funkcija treba da vrati 0 ukoliko je sve bilo u redu, odnosno
41   1 ukoliko je šdolo do šgreke prilikom alokacije memorije za
   nov čvor. */
43 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
45   /* Kreira se nov cvor i proverava se da li je bilo greske pri
   alokaciji */
47   Cvor *novi = napravi_cvor(broj);
   if (novi == NULL)
49       return 1;

   /* Novi cvor se uvezuje na pocetak, i tako postaje nova glave
51   liste. */
53   novi->sledeci = *adresa_glave;
   *adresa_glave = novi;

55   return 0;
57 }

59 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste,
   ili NULL ukoliko je lista prazna. */
61 Cvor *pronadji_poslednji(Cvor * glava)
{
63   /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
   if (glava == NULL)
65       return NULL;

67   /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
   pokazivac glava se pomera na sledeci cvor. Nakon izlaska iz
69   petlje, glava ce pokazivati na cvor liste koji nema
   sledeceg, tj, poslednji je cvor liste, vraca se vrednost
71   pokazivaca glava. Pokazivac glava je argument funkcije i
   njegove promene nece se odraziti na vrednost pokazivaca
73   glava u pozivajucoj funkciji. */
   while (glava->sledeci != NULL)
75       glava = glava->sledeci;

77   return glava;
79 }

/* Funkcija dodaje broj na kraj liste. UKoliko dodje do greske
81   pri alokaciji memorije vratice 1, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
83 {
   Cvor *novi = napravi_cvor(broj);
85   if (novi == NULL)
       return 1;

87 }
```

```
89  /* U slucaju prazne liste, glava nove liste je upravo novi
    cvor i ujedno i cela lista. Azurira se vrednost na koju
91  pokazuje adresa_glave i tako se azurira i pokazivacka
    promenljivu u pozivajucoj funkciji. */
93  if (*adresa_glave == NULL) {
    *adresa_glave = novi;
    return 0;
95  }

97  /* Ako lista nije prazna, pronalazi se poslednji cvor i novi
    cvor se dodaje na kraj liste kao sledbenik poslednjeg. */
99  Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
101
    return 0;
103 }

105 /* Pomocna funkcija pronalazi cvor u listi iza koga treba
    umetnuti nov cvor sa vrednoscu broj. */
107 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
    {
109     /* U praznoj listi nema takvog mesta i vraca se NULL. */
    if (glava == NULL)
111         return NULL;

113     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
        pokazivala na cvor ciji je sledeci ili ne postoji ili ima
115         vrednost vecu ili jednaku vrednosti novog cvora.

        Zbog izracunavanja izraza u C-u prvi deo konjukcije mora
        biti provera da li se doslo do poslednjeg cvora liste pre
119         nego sto se proveru vrednost njegovog sledeceg cvora, jer u
        slucaju poslednjeg, sledeci ne postoji, pa ni njegova
121         vrednost. */
    while (glava->sledeci != NULL
           && glava->sledeci->vrednost < broj)
        glava = glava->sledeci;
125

    /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
        poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima
        vrednost vecu od broj. */
127     return glava;
129 }

131 /* Funkcija uvezuje cvor novi iza cvora tekuci. */
133 void dodaj_iza(Cvor * tekuci, Cvor * novi)
    {
135     /* Novi cvor se dodaje iza tekuceg cvora. */
    novi->sledeci = tekuci->sledeci;
137     tekuci->sledeci = novi;
    }
139
```

```

141  /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
      sortirana. Vraca 1 ukoliko je bilo greski pri alokaciji
      memorije, inace vraca 0. */
143  int dodaj_sortirano(Cvor ** adresa_glave, int broj)
      {
145      /* U slucaju prazne liste glava nove liste je novi cvor. */
      if (*adresa_glave == NULL) {
147          Cvor *novi = napravi_cvor(broj);
          if (novi == NULL)
149              return 1;
          *adresa_glave = novi;
151          return 0;
      }

153      /* Lista nije prazna. */
155      /* Ako je broj manji ili jednak vrednosti u glavi liste, onda
          ga treba dodati na pocetak liste. */
157      if ((*adresa_glave)->vrednost >= broj) {
          return dodaj_na_pocetak_liste(adresa_glave, broj);
159      }

161      /* U slucaju da je glava liste cvor sa vrednoscu manjom od
          broj, tada se pronalazi cvor liste iza koga treba da se
163          umetne nov broj. */
      Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
165      Cvor *novi = napravi_cvor(broj);
      if (novi == NULL)
167          return 1;

169      /* Uvezuje se novi cvor iza pomocnog. */
      dodaj_iza(pomocni, novi);
171      return 0;
      }

173      /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
175          broju. Vraca pokazivac na cvor liste u kome je sadrzan
          trazeni broj ili NULL u slucaju da takav cvor ne postoji u
177          listi. */
      Cvor *pretrazi_listu(Cvor * glava, int broj)
      {
179          for (; glava != NULL; glava = glava->sledeci)
181              if (glava->vrednost == broj)
                  return glava;

183          /* Nema trazenog broja u listi i vraca se NULL. */
185          return NULL;
      }

187      /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
189          broju. Vraca pokazivac na cvor liste u kome je sadrzan
          trazeni broj ili NULL u slucaju da takav cvor ne postoji u
191          listi. Funkcija se u pretrazi oslanja na cinjenicu da je

```

```
193     lista koja se pretrazuje neopadajuće sortirana. */
194 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
195 {
196     /* U uslovu ostanka u petlji, bitan je redosled u konjukciji.*/
197     for (; glava != NULL && glava->vrednost <= broj;
198         glava = glava->sledeci)
199         if (glava->vrednost == broj)
200             return glava;
201
202     return NULL;
203 }
204
205 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
206 Funkcija azurira pokazivac na glavu liste, koji moze biti
207 promenjen u slucaju da se obrise stara glava. */
208 void obrisi_cvor(Cvor ** adresa_glave, int broj)
209 {
210     Cvor *tekuci = NULL;
211     Cvor *pomocni = NULL;
212
213     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
214        broju, i azurira se pokazivac na glavu liste. */
215     while (*adresa_glave != NULL
216         && (*adresa_glave)->vrednost == broj) {
217         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora
218            na adresi adresa_glave. */
219         pomocni = (*adresa_glave)->sledeci;
220         free(*adresa_glave);
221         *adresa_glave = pomocni;
222     }
223
224     /* Ako je nakon toga lista ostala prazna, izlazi se iz
225        funkcije. */
226     if (*adresa_glave == NULL)
227         return;
228
229     /* Od ovog trenutka, u svakoj iteraciji petlje tekuci pokazuje
230        na cvor cija vrednost je razlicita od traznog broja. Isto
231        vazi i za sve cvorove levo od tekućeg. Poredi se vrednost
232        sledećeg cvora (ako postoji) sa traznim brojem. Cvor se
233        brise ako je jednak, ili, ako je razlicit, prelazi se na
234        sledeći cvor. Ovaj postupak se ponavlja dok se ne dodje do
235        poslednjeg cvora. */
236     tekuci = *adresa_glave;
237     while (tekuci->sledeci != NULL)
238         if (tekuci->sledeci->vrednost == broj) {
239             /* tekuci->sledeci treba obrisati, zbog toga se njegova
240                adresa prvo cuva u pomocni. */
241             pomocni = tekuci->sledeci;
242             /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
243                njegovog trenutnog sledećeg. Njegov novi sledeci ce
244                biti sledeci od ovog cvor koji se brise. */
245             tekuci->sledeci = pomocni->sledeci;
246             free(pomocni);
247         }
248     }
249 }
```

```

245     tekuci->sledeci = pomocni->sledeci;
/* Sada treba osloboditi cvor sa vrednoscu broj. */
247     free(pomocni);
} else {
249     /* Ne treba brisati sledeceg od tekuceg i pokazivac se
pomera dalje. */
    tekuci = tekuci->sledeci;
251 }
return;
253 }

255 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
oslanjajuci se na cinjenicu da je prosledjena lista sortirana
257 neopadajuce. Funkcija azurira pokazivac na glavu liste, koji
moze biti promenjen ukoliko se obrise stara glava liste. */
259 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
261     Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;
263
    /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
265 broju, i azurira se pokazivac na glavu liste. */
    while (*adresa_glave != NULL
267         && (*adresa_glave)->vrednost == broj) {
        /* Adresu repa liste treba sacuvati pre oslobadjanja cvora
269 na adresi adresa_glave. */
        pomocni = (*adresa_glave)->sledeci;
271        free(*adresa_glave);
        *adresa_glave = pomocni;
273    }

275    /* Ako je nakon toga lista ostala prazna, funkcija se prekida.
Isto se radi i ukoliko glava liste sadrzi vrednost koja je
277 veca od broja, jer kako je lista sortirana rastuce nema
potrebe broj traziti u repu liste. */
279    if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;
281
    /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci
283 pokazuje na cvor cija vrednost je manja od trazenog broja,
kao i svim cvorovima levo od njega. Cvor se brise ako je
285 jednak, ili, ako je razlicit, prelazi se na sledeci cvor.
Ovaj postupak se ponavlja dok se ne dodje do poslednjeg
287 cvora ili prvog cvora cija vrednost je veca od trazenog
broja. */
289    tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL
291         && tekuci->sledeci->vrednost <= broj)
        if (tekuci->sledeci->vrednost == broj) {
293            pomocni = tekuci->sledeci;
            tekuci->sledeci = tekuci->sledeci->sledeci;
295            free(pomocni);

```

```

    } else {
297     /* Ne treba brisati sledeceg od tekuceg, jer je manji od
        trazenog i prelazi se na sledeci. */
299     tekuci = tekuci->sledeci;
    }
301     return;
}
303
/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
305 kraju liste. Ne salje joj se adresa promenljive koja cuva
glavu liste, jer ova funkcija nece menjati listu, pa nema ni
307 potrebe da azuriza pokazivac na glavu liste iz pozivajuce
funkcije. */
309 void ispisi_listu(Cvor * glava)
{
311     putchar('[');
    for (; glava != NULL; glava = glava->sledeci) {
313         printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
315             printf(", ");
    }
317     printf("]\n");
319 }

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  int main()
{
7     /* Lista je prazna na pocetku. */
    Cvor *glava = NULL;
9     Cvor *trazeni = NULL;
    int broj;

11
    /* Testiranje dodavanja novog broja na pocetak liste. */
13     printf("Unosite brojeve: (za kraj unesite CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
15         /* Ako je funkcija vratila 1 onda je bilo greske pri
            alokaciji memorije za nov cvor. Memoriju alociranu za
            cvorove liste treba osloboditi pre napustanja programa. */
17         if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava);
21             exit(EXIT_FAILURE);
        }
23         printf("\tLista: ");
        ispisi_listu(glava);
25     }

27     printf("\nUnosite broj koji se trazi u listi: ");

```

```

scanf("%d", &broj);

29 trazeneni = pretrazi_listu(glava, broj);
31 if (trazeneni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
33 else
    printf("Trazeni broj %d je u listi!\n", trazeneni->vrednost);
35
    oslobodi_listu(&glava);
37
    return 0;
39 }

1 #include <stdio.h>
#include <stdlib.h>
3 #include "lista.h"

5 int main()
{
7     Cvor *glava = NULL;
    int broj;

9
    /* Testiranje dodavanja novog broja na kraj liste. */
11 printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
    printf("\tLista: ");
13 ispisi_listu(glava);

15 while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1 onda je bilo greske pri
17     alokaciji memorije za nov cvor. Memoriju alociranu za
        cvorove liste treba osloboditi pre napustanja programa. */
19     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
        fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21         oslobodi_listu(&glava);
        exit(EXIT_FAILURE);
23     }
        printf("\tLista: ");
25         ispisi_listu(glava);
    }

27
    printf("\nUnesite broj koji se brise iz liste: ");
29 scanf("%d", &broj);

31
    /* Brisu se cvorovi iz liste cije polje vrednost je jednako
        broju procitanom sa ulaza */
33 obrisi_cvor(&glava, broj);

35
    printf("Lista nakon brisanja: ");
    ispisi_listu(glava);
37
    oslobodi_listu(&glava);
39

```

```

    return 0;
41 }

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  int main()
   {
7      Cvor *glava = NULL;
      Cvor *trazeni = NULL;
9      int broj;

11     /* Testira se dodavanje u listu tako da ona bude neopadajuće
       uredjena */
13     printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
     printf("\tLista: ");
15     ispisi_listu(glava);

17     while (scanf("%d", &broj) > 0) {
       /* Ako je funkcija vratila 1 onda je bilo greske pri
19        alokaciji memorije za nov cvor. Memoriju alociranu za
        cvorove liste treba osloboditi pre napustanja programa. */
21         if (dodaj_sortirano(&glava, broj) == 1) {
             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23             oslobodi_listu(&glava);
             exit(EXIT_FAILURE);
25         }
         printf("\tLista: ");
27         ispisi_listu(glava);
     }

29     printf("\nUnosite broj koji se trazi u listi: ");
31     scanf("%d", &broj);

33     trazeni = pretrazi_listu(glava, broj);
     if (trazeni == NULL)
35         printf("Broj %d se ne nalazi u listi!\n", broj);
     else
37         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

39     printf("\nUnosite broj koji se brise iz liste: ");
     scanf("%d", &broj);

41     /* Brisu se cvorovi iz liste cije polje vrednost je jednako
       broju procitanom sa ulaza */
     obrisi_cvor_sortirane_liste(&glava, broj);

45     printf("Lista nakon brisanja: ");
     ispisi_listu(glava);

47     oslobodi_listu(&glava);
49

```



```

51     return 0;
}

```

## Rešenje 4.2

```

2  #ifndef _LISTA_H
3  #define _LISTA_H
4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
5     podatak vrednost i pokazivac na sledeci cvor liste. */
6  typedef struct cvor {
7     int vrednost;
8     struct cvor *sledeci;
9  } Cvor;
10
11 Cvor *napravi_cvor(int broj);
12
13 void oslobodi_listu(Cvor ** adresa_glave);
14
15 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
16
17 Cvor *pronadji_poslednji(Cvor * glava);
18
19 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
20
21 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
22
23 void dodaj_iza(Cvor * tekuci, Cvor * novi);
24
25 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
26
27 Cvor *pretrazi_listu(Cvor * glava, int broj);
28
29 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
30
31 void obrisi_cvor(Cvor ** adresa_glave, int broj);
32
33 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
34
35 void ispisi_vrednosti(Cvor * glava);
36
37 void ispisi_listu(Cvor * glava);
38
39 #endif

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"

```

```
5  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
7     cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
9     novom cvoru postavlja na NULL. Funkcija vraća pokazivac na
        novokreirani cvor ili NULL ako alokacija nije uspesno
        izvršena. */
11  Cvor *napravi_cvor(int broj)
12  {
13      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
14      if (novi == NULL)
15          return NULL;
16
17      novi->vrednost = broj;
18      novi->sledeci = NULL;
19      return novi;
20  }
21
22  /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove
23     liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
24  void oslobodi_listu(Cvor ** adresa_glave)
25  {
26      /* Lista je vec prazna */
27      if (*adresa_glave == NULL)
28          return;
29
30      /* Ako lista nije prazna, treba osloboditi memoriju. Pre
31         oslobadjanja memorije za glavu liste, treba osloboditi rep
32         liste. */
33      oslobodi_listu(&(*adresa_glave)->sledeci);
34      /* Nakon oslobodjenog repa, oslobadja se glava liste, i
35         azurira se glava u pozivajucoj funkciji tako da odgovara
36         praznoj listi */
37      free(*adresa_glave);
38      *adresa_glave = NULL;
39  }
40
41  /* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor
42     koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak.
43     Funkcija vraća 1 ukoliko je doslo do greske pri alokaciji,
44     inace vraća 0. */
45  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
46  {
47      /* Kreira se nov cvor i proverava se da li je bilo greske pri
48         alokaciji */
49      Cvor *novi = napravi_cvor(broj);
50      if (novi == NULL)
51          return 1;
52
53      /* Novi cvor se uvezuje na pocetak, i tako postaje nova glave
54         liste */
55      novi->sledeci = *adresa_glave;
56      *adresa_glave = novi;
57      return 0;
```

```

57 }

59 /* Funkcija dodaje novi cvor na kraj liste. Prilikom dodavanja u
   listu na kraj u velikoj vecini slucajeva nov broj se dodaje u
61 rep liste u rekurzivnom pozivu. U slucaju da je u rekurzivnom
   pozivu doslo do greske pri alokaciji, funkcija vraca 1 visem
63 rekurzivnom pozivu koji tu informaciju vraca u rekurzivni
   poziv iznad, sve dok se ne vrati u main. Ako je funkcija
65 vratila 0, onda nije bilo greske. Tek je iz main funkcije
   moguće pristupiti pravom pocetku liste i osloboditi je celu,
67 ako ima potrebe. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
69 {
   if (*adresa_glave == NULL) {
71     /* Glava liste je upravo novi cvor i ujedno i cela lista. */
       Cvor *novi = napravi_cvor(broj);
73     if (novi == NULL)
         return 1;
75
       /* Azurira se vrednost na koju pokazuje adresa_glave i
77        ujedno se azurira i pokazivacka promenljiva u pozivajucoj
         funkciji. */
79     *adresa_glave = novi;
       return 0;
81   }

83   /* Ako lista nije prazna, broj se dodaje u rep liste. */
   return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
85 }

87 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova
   lista ostane sortirana. Vraca 0, ako je alokacija novog cvora
89 prosla bez greske, inace vraca 1 da bi ta vrednost bila
   propagirala nazad do prvog poziva. */
91 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
   {
93     /* U slucaju prazne liste adresa_glave nove liste je upravo
       novi cvor. */
95     if (*adresa_glave == NULL) {
       Cvor *novi = napravi_cvor(broj);
97     if (novi == NULL)
         return 1;
99
       *adresa_glave = novi;
101     return 0;
   }

103
   /* Lista nije prazna. Ako je broj manji ili jednak vrednosti u
105    glavi liste, onda se dodaje na pocetak liste i vraca se
     informacija o uspesnosti alokacije. */
107 if ((*adresa_glave)->vrednost >= broj)
   return dodaj_na_pocetak_liste(adresa_glave, broj);

```

```
109      /* Inace, broj treba dodati u rep, tako da rep i sa novim
111         cvorom bude sortirana lista. */
      return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
113  }

115  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
      broju. Vraca pokazivac na cvor liste u kome je sadržan
117     traženi broj ili NULL u slučaju da takav cvor ne postoji u
      listi. */
119  Cvor *pretrazi_listu(Cvor * glava, int broj)
  {
121     /* U praznoj listi ga sigurno nema */
      if (glava == NULL)
123         return NULL;

125     /* Ako glava liste sadrži traženi broj */
      if (glava->vrednost == broj)
127         return glava;

129     /* Ako nije nijedna od prethodnih situacija, pretraga se
      nastavlja u repu liste. */
131     return pretrazi_listu(glava->sledeci, broj);
  }

133  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
      broju. Funkcija se u pretrazi oslanja na činjenicu da je
135     lista koja se pretražuje neopadajuće sortirana. Vraca
      pokazivac na cvor liste u kome je sadržan traženi broj ili
137     NULL u slučaju da takav cvor ne postoji u listi. */
139  Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
  {
141     /* Traženog broja nema ako je lista prazna ili ako je broj
      manji od vrednosti u glavi liste, jer je lista neopadajuće
143     sortirana. */
      if (glava == NULL || glava->vrednost > broj)
145         return NULL;

147     /* Ako glava liste sadrži traženi broj, vraca se glava. */
      if (glava->vrednost == broj)
149         return glava;

151     /* Ako nije nijedna od prethodnih situacija, pretraga se
      nastavlja u repu. */
153     return pretrazi_listu(glava->sledeci, broj);
  }

155  /* Funkcija briše iz liste sve cvorove koji sadrže dati broj.
      Funkcija azurira pokazivac na glavu liste, koji može biti
157     promenjen u slučaju da se obriše stara glava liste. */
159  void obrisi_cvor(Cvor ** adresa_glave, int broj)
  {
```

```

161  /* U praznoj listi, nema cvorova za brisanje. */
162  if (*adresa_glave == NULL)
163      return;

165  /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj. */
166  obrisi_cvor(&(*adresa_glave)->sledeci, broj);

167
168  /* Preostaje provera da li glavu liste treba obrisati. */
169  if ((*adresa_glave)->vrednost == broj) {
170      /* pomocni pokazuje na cvor koji treba da se obrise. */
171      Cvor *pomocni = *adresa_glave;
172      /* Azurira se pokazivac na glavu da pokazuje na sledeci u
173         listi i brise se cvor koji je bio glava liste. */
174      *adresa_glave = (*adresa_glave)->sledeci;
175      free(pomocni);
176  }
177  }

179  /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
180     oslanjajuci se na cinjenicu da je prosledjena lista sortirana
181     neopadajuće. Funkcija azurira pokazivac na glavu liste, koji
182     može biti promenjen ukoliko se obrise stara glava liste. */
183  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
184  {
185      /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
186         veca od broja, kako je lista sortirana rastuce nema potrebe
187         broj traziti u repu liste i zato se funkcija prekida. */
188      if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
189          return;

191      /* Brisu se cvorovi iz repa koji imaju vrednost broj. */
192      obrisi_cvor(&(*adresa_glave)->sledeci, broj);

193
194      /* Preostaje provera da li glavu liste treba obrisati. */
195      if ((*adresa_glave)->vrednost == broj) {
196          /* pomocni pokazuje na cvor koji treba da se obrise. */
197          Cvor *pomocni = *adresa_glave;
198          /* Azurira se pokazivac na glavu da pokazuje na sledeci u
199             listi i brise se cvor koji je bio glava liste. */
200          *adresa_glave = (*adresa_glave)->sledeci;
201          free(pomocni);
202      }
203  }

205  /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
206     zarezom. */
207  void ispisi_vrednosti(Cvor * glava)
208  {
209      /* Prazna lista */
210      if (glava == NULL)
211          return;

```

```
213  /* Ispisuje se vrednost u glavi liste. */
    printf("%d", glava->vrednost);
215
    /* Ako rep nije prazan, ispisuje se znak ',' i razmak.
    Rekurzivno se poziva ista funkcija za ispis ostalih. */
217    if (glava->sledeci != NULL) {
219        printf(", ");
        ispisi_vrednosti(glava->sledeci);
221    }
    }
223
    /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
    kraju liste. Ne salje joj se adresa promenljive koja cuva
    glavu liste, jer ova funkcija nece menjati listu, pa nema ni
    potrebe da azuriza pokazivac na glavu liste iz pozivajuće
    funkcije. Ova funkcija ispisuje samo zagrade, a rekurzivno
    ispisivanje vrednosti u listi prepusta rekurzivnoj pomocnoj
    funkciji ispisi_vrednosti, koja ce ispisati elemente
    razdvojene zapetom i razmakom. */
231 void ispisi_listu(Cvor * glava)
233 {
    putchar('[');
235    ispisi_vrednosti(glava);
    printf("]\n");
237 }
```

### Rešenje 4.3

```
1  #ifndef _LISTA_H
    #define _LISTA_H
3
    /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
    podatak vrednost i pokazivace na sledeci i prethodni cvor
    liste. */
5    typedef struct cvor {
        int vrednost;
        struct cvor *sledeci;
        struct cvor *prethodni;
11    } Cvor;

13    Cvor *napravi_cvor(int broj);

15    void oslobodi_listu(Cvor ** adresa_glave);

17    int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

19    Cvor *pronadji_poslednji(Cvor * glava);

21    int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

23    Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
```

```

25 void dodaj_iza(Cvor * tekuci, Cvor * novi);
27 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
29 Cvor *pretrazi_listu(Cvor * glava, int broj);
31 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
33 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci);
35 void obrisi_cvor(Cvor ** adresa_glave, int broj);
37 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
39 void ispisi_listu(Cvor * glava);
41 void ispisi_listu_u_nazad(Cvor * glava);
43 #endif

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"

5  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
6     cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
7     novom cvoru postavlja na NULL. Funkcija vraća pokazivac na
8     novokreirani cvor ili NULL ako alokacija nije uspesno
9     izvršena. */
10 Cvor *napravi_cvor(int broj)
11 {
12     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
13     if (novi == NULL)
14         return NULL;
15
16     novi->vrednost = broj;
17     novi->sledeci = NULL;
18     return novi;
19 }

21 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove
22     liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
23 void oslobodi_listu(Cvor ** adresa_glave)
24 {
25     Cvor *pomocni = NULL;

27     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
28     while (*adresa_glave != NULL) {
29         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
30            osloboditi memoriju cvora koji predstavlja glavu liste. */
31         pomocni = (*adresa_glave)->sledeci;

```

```
        free(*adresa_glave);
33     /* Sledeci cvor je nova glava liste. */
        *adresa_glave = pomocni;
35     }
36 }
37
38 /* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor
39    koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak.
    Vraca 1 ukoliko je bilo greski pri alokaciji memorije, inace
    vraca 0. */
41 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
42 {
43     Cvor *novi = napravi_cvor(broj);
44     if (novi == NULL)
45         return 1;
46
47     /* Sledbenik novog cvora je glava stare liste */
48     novi->sledeci = *adresa_glave;
49     /* Ako stara lista nije bila prazna, onda prethodni od glave
    treba da bude nov cvor. */
51     if (*adresa_glave != NULL)
52         (*adresa_glave)->prethodni = novi;
53     /* Novi cvor je nova glava liste. */
54     *adresa_glave = novi;
55
56     return 0;
57 }
58
59 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste,
    ili NULL ukoliko je lista prazna. */
61 Cvor *pronadji_poslednji(Cvor * glava)
62 {
63     /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
64     if (glava == NULL)
65         return NULL;
66
67     /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
    pokazivac glava se pomera na sledeci cvor. Nakon izlaska iz
    petlje, glava ce pokazivati na cvor liste koji nema
    sledeceg, tj, poslednji je cvor liste, vraca se vrednost
    pokazivaca glava. Pokazivac glava je argument funkcije i
    njegove promene nece se odraziti na vrednost pokazivaca
    glava u pozivajucoj funkciji. */
75     while (glava->sledeci != NULL)
76         glava = glava->sledeci;
77
78     return glava;
79 }
80
81 /* Funkcija dodaje broj na kraj liste. Ukoliko dodje do greske
    pri alokaciji memorije vratice 1, inace vraca 0. */
83 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
```



```

{
85   Cvor *novi = napravi_cvor(broj);
   if (novi == NULL)
87     return 1;

89   /* U slucaju prazne liste, glava nove liste je upravo novi
   cvor i ujedno i cela lista. Azurira se vrednost na koju
91   pokazuje adresa_glave i tako se azurira i pokazivacka
   promenljivu u pozivajucoj funkciji. */
93   if (*adresa_glave == NULL) {
       *adresa_glave = novi;
95   return 0;
   }

97   /* Kako lista nije prazna, pronalazi se poslednji cvor i novi
99   cvor se dodaje na kraj liste kao sledbenik poslednjeg. */
   Cvor *poslednji = pronadji_poslednji(*adresa_glave);
101  poslednji->sledeci = novi;
   novi->prethodni = poslednji;
103
   return 0;
105 }

107 /* Pomocna funkcija pronalazi cvor u listi iza koga treba
   umetnuti nov cvor sa vrednoscu broj. */
109 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
   {
111   /* U praznoj listi nema takvog mesta i vraca se NULL. */
   if (glava == NULL)
113     return NULL;

115   /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
   pokazivala na cvor ciji je sledeci ili ne postoji ili ima
117   vrednost vecu ili jednaku vrednosti novog cvora.

119   Zbog izracunavanja izraza u C-u prvi deo konjukcije mora
   biti proverava da li se doslo do poslednjeg cvora liste pre
121   nego sto se proveru vrednost njegovog sledeceg cvora, jer u
   slucaju poslednjeg, sledeci ne postoji, pa ni njegova
123   vrednost. */
   while (glava->sledeci != NULL
125         && glava->sledeci->vrednost < broj)
       glava = glava->sledeci;
127

   /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
129   poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima
   vrednost vecu od broj. */
131   return glava;
   }
133

   /* Funkcija uvezuje cvor novi iza cvora tekuci. */
135 void dodaj_iza(Cvor * tekuci, Cvor * novi)

```

```
{
137     novi->sledeci = tekuci->sledeci;
    novi->prethodni = tekuci;
139
    /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje
141     prethodnik i tekuci dobija novog sledeceg postavljanjem
        pokazivaca na ispravne adrese. */
143     if (tekuci->sledeci != NULL)
        tekuci->sledeci->prethodni = novi;
145     tekuci->sledeci = novi;
}

147
/* Funkcija dodaje u listu nov cvor na odgovarajuće mesto, tako
149 sto pronalazi cvor u listi iza kod treba uvezati nov cvor.
    Ukoliko dodje do greske pri alokaciji memorije vratice 1,
151 inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, int broj)
153 {
    /* Ako je lista prazna, glava nove liste je novi cvor. */
155     if (*adresa_glave == NULL) {
        Cvor *novi = napravi_cvor(broj);
157         if (novi == NULL)
            return 1;
        *adresa_glave = novi;
159         return 0;
    }
161

163     /* Ukoliko je vrednost glave liste veka ili jednaka od nove
        vrednosti onda nov cvor treba staviti na pocetak liste. */
165     if ((*adresa_glave)->vrednost >= broj) {
        dodaj_na_pocetak_liste(adresa_glave, broj);
167         return 0;
    }
169

    Cvor *novi = napravi_cvor(broj);
171     if (novi == NULL)
        return 1;
173

    /* Pronazi se cvor iza koga treba uveziti nov cvor. */
175     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
    dodaj_iza(pomocni, novi);
177
    return 0;
179 }

181 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
    broju. Vraca pokazivac na cvor liste u kome je sadržan
183 trazeni broj ili NULL u slucaju da takav cvor ne postoji u
    listi. */
185 Cvor *pretrazi_listu(Cvor * glava, int broj)
{
187     for (; glava != NULL; glava = glava->sledeci)
```

```

189     if (glava->vrednost == broj)
        return glava;

191     /* Nema trazenog broja u listi i vraca se NULL. */
    return NULL;
193 }

195 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
    broju. Funkcija se u pretrazi oslanja na cinjenicu da je
197 lista koja se pretrazuje neopadajuće sortirana. Vraca
    pokazivac na cvor liste u kome je sadržan traženi broj ili
199 NULL u slučaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
201 {
    /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
203     for (; glava != NULL && glava->vrednost <= broj;
            glava = glava->sledeci)
205         if (glava->vrednost == broj)
            return glava;

207     /* Nema trazenog broja u listi i bice vraceno NULL. */
209     return NULL;
}

211 /* Funkcija brise u listi na koju pokazuje pokazivac glava onaj
213 cvor na koji pokazuje pokazivac tekuci. Obratiti paznju da je
    kod dvostruke liste ovo mnogo lakse uraditi jer cvor tekuci
215 sadrži pokazivace na svog sledbenika i prethodnika u listi. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
217 {
    /* Ako je tekuci NULL pokazivac, nema sta da se brise. */
219     if (tekuci == NULL)
        return;

221     /* Ako postoji prethodnik od tekućeg, onda se postavlja da
223 njegov sledeci bude sledeci od tekućeg. */
    if (tekuci->prethodni != NULL)
225         tekuci->prethodni->sledeci = tekuci->sledeci;

227     /* Ako postoji sledbenik tekućeg (cvora koji se brise), onda
        njegov prethodnik treba da bude prethodnik tekućeg. */
229     if (tekuci->sledeci != NULL)
        tekuci->sledeci->prethodni = tekuci->prethodni;

231     /* Ako je glava cvor koji se brise, glava nove liste ce biti
233 sledbenik od stare glave. */
    if (tekuci == *adresa_glave)
235         *adresa_glave = tekuci->sledeci;

237     /* Oslobadja se dinamički alociran prostor za cvor tekuci. */
    free(tekuci);
239 }

```

```
241 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
    Funkcija azurira pokazivac na glavu liste, koji moze biti
243 promenjen u slucaju da se obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, int broj)
245 {
    Cvor *tekuci = *adresa_glave;

247     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
249         obrisi_tekuci(adresa_glave, tekuci);
}

251 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
    oslanjajuci se na cinjenicu da je prosledjena lista
253 neopadajuće sortirana. Funkcija azurira pokazivac na glavu
    liste, koji moze biti promenjen ukoliko se obrise stara glava
255 liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
257 {
    Cvor *tekuci = *adresa_glave;

259     while ((tekuci =
        pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
261         obrisi_tekuci(adresa_glave, tekuci);
}

263 /* Funkcija prikazuje cvorove liste pocv od glave ka kraju
    liste. Ne salje joj se adresu promenljive koja cuva glavu
265 liste, jer ova funkcija neće menjati listu, pa nema ni
    potrebe da azurira pokazivac na glavu liste iz pozivajuće
267 funkcije. */
void ispisi_listu(Cvor * glava)
269 {
    putchar('[');
    for (; glava != NULL; glava = glava->sledeci) {
271         printf("%d", glava->vrednost);
273         if (glava->sledeci != NULL)
275             printf(", ");
277     }

    printf("]\n");
279 }

281 /* Funkcija prikazuje cvorove liste pocv od kraja ka glavi
    liste. */
void ispisi_listu_u_nazad(Cvor * glava)
283 {
    putchar('[');
    if (glava == NULL) {
285         printf("]\n");
287         return;
289     }
291 }
```

```

293     glava = pronadji_poslednji(glava);

295     for (; glava != NULL; glava = glava->prethodni) {
        printf("%d", glava->vrednost);
297         if (glava->prethodni != NULL)
            printf(", ");
299     }
    printf("\n");
301 }

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  int main()
   {
7      /* Lista je prazna na pocetku. */
      Cvor *glava = NULL;
9      Cvor *trazeni = NULL;
      int broj;

11     /* Testiranje dodavanja novog broja na pocetak liste. */
13     printf("Unosite brojeve: (za kraj unesite CTRL+D)\n");
     while (scanf("%d", &broj) > 0) {
15         /* Ako je funkcija vratila 1 onda je bilo greske pri
            alokaciji memorije za nov cvor. Memoriju alociranu za
17         cvorove liste treba osloboditi pre napustanja programa. */
         if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
19             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
             oslobodi_listu(&glava);
21             exit(EXIT_FAILURE);
         }
23         printf("\tLista: ");
         ispisi_listu(glava);
25     }

27     printf("\nUnosite broj koji se trazi u listi: ");
     scanf("%d", &broj);

29     trazeni = pretrazi_listu(glava, broj);
     if (trazeni == NULL)
31         printf("Broj %d se ne nalazi u listi!\n", broj);
     else
33         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

35     printf("\nLista ispisana u nazad: ");
37     ispisi_listu_u_nazad(glava);

39     oslobodi_listu(&glava);

41     return 0;

```

```
}
```

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

4
int main()
6 {
    Cvor *glava = NULL;
    8 int broj;

    /* Testiranje dodavanja novog broja na kraj liste. */
    printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
    12 printf("\tLista: ");
    ispisi_listu(glava);

    14 while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1 onda je bilo greske pri
        16 alokaciji memorije za nov cvor. Memoriju alociranu za
        18 cvorove liste treba osloboditi pre napustanja programa. */
        if (dodaj_na_kraj_liste(&glava, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            20 oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
            22 }
            printf("\tLista: ");
            24 ispisi_listu(glava);
        }
        26

        printf("\nUnesite broj koji se brise iz liste: ");
        28 scanf("%d", &broj);

        /* Brisu se cvorovi iz liste cije polje vrednost je jednako
        30 broju procitanom sa ulaza */
        obrisi_cvor(&glava, broj);
        32

        printf("Lista nakon brisanja: ");
        34 ispisi_listu(glava);

        printf("\nLista ispisana u nazad: ");
        38 ispisi_listu_u_nazad(glava);

        40 oslobodi_listu(&glava);

        42 return 0;
    }
    44 }
```

```
#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"
4
```

```
int main()
6 {
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
    int broj;

    /* Testira se dodavanje u listu tako da ona bude neopadajuće
    uredjena */
    printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
    printf("\tLista: ");
    ispisi_listu(glava);

    while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1 onda je bilo greske pri
        alokaciji memorije za nov cvor. Memoriju alociranu za
        20 cvorove liste treba osloboditi pre napustanja programa. */
        if (dodaj_sortirano(&glava, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
        }
        printf("\tLista: ");
        ispisi_listu(glava);
    }

    printf("\nUnosite broj koji se trazi u listi: ");
    scanf("%d", &broj);

    trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
    else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

    printf("\nUnosite broj koji se brise iz liste: ");
    scanf("%d", &broj);

    /* Brisu se cvorovi iz liste cije polje vrednost je jednako
    broju procitanom sa ulaza */
    obrisi_cvor_sortirane_liste(&glava, broj);

    printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

    printf("\nLista ispisana u nazad: ");
    ispisi_listu_u_nazad(glava);

    oslobodi_listu(&glava);

    return 0;
}
```

## Rešenje 4.4

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

4 typedef struct cvor {
    char zagrada;
    struct cvor *sledeci;
6 } Cvor;

8
9
10 int main()
11 {
    Cvor *stek = NULL;
12     FILE *ulaz = NULL;
    char c;
14     Cvor *pomocni = NULL;

16     ulaz = fopen("izraz.txt", "r");
    if (ulaz == NULL) {
18         fprintf(stderr,
            "Greska prilikom otvaranja datoteke izraz.txt!\n");
20         exit(EXIT_FAILURE);
    }

22
23     while ((c = fgetc(ulaz)) != EOF) {
24         /* Ako je učitana otvorena zagrada, stavlja se na stek. */
        if (c == '(' || c == '{' || c == '[') {
26             pomocni = (Cvor *) malloc(sizeof(Cvor));
            if (pomocni == NULL) {
28                 fprintf(stderr, "Greska prilikom alokacije memorije!\n");
                    return 1;
30             }
            pomocni->zagrada = c;
32             pomocni->sledeci = stek;
            stek = pomocni;
34         }
        /* Ako je učitana zatvorena zagrada, proverava se da li je
36         stek je prazan i ako nije, da li se na vrhu steka nalazi
            odgovarajuca otvorena zagrada. */
38         else {
            if (c == ')' || c == '}' || c == ']') {
40                 if (stek != NULL && ((stek->zagrada == '(' && c == ')')
                    || (stek->zagrada == '{' && c
42                        == '}')
                    || (stek->zagrada == '[' && c
44                        == ']')))) {

                        /* Sa vrha steka se uklanja otvorena zagrada */
46                         pomocni = stek->sledeci;
                        free(stek);
                        stek = pomocni;
48                     } else {
50                         /* Zagrade u izrazu nisu ispravno uparene. */
```



```

        break;
52     }
    }
54 }
}

56 /* Ako je na kraju stek prazan i procitana je cela datoteka,
57    zagrade su ispravno uparene, u suprotnom, nisu. */
58 if (stek == NULL && c == EOF)
60     printf("Zagrade su ispravno uparene.\n");
61 else {
62     printf("Zagrade nisu ispravno uparene.\n");

63     while (stek != NULL) {
64         /* U slucaju neispravnog uparivanja treba osloboditi
65            memoriju koja je ostala zauzeta stekom. */
66         pomocni = stek->sledeci;
67         free(stek);
68         stek = pomocni;
69     }
70 }

71 fclose(ulaz);
72 return 0;
73 }

```

### Rešenje 4.5

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX 100
7
8  #define OTVORENA 1
9  #define ZATVORENA 2
10
11 #define VAN_ETIKETE 0
12 #define PROCITANO_MANJE 1
13 #define U_ETIKETI 2
14
15 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete
16    i pokazivac na sledeci cvor. */
17 typedef struct cvor {
18     char etiketa[MAX];
19     struct cvor *sledeci;
20 } Cvor;
21
22 /* Funkcija kreira novi cvor, upisuje u njega etiketu i
23    vraća njegovu adresu ili NULL ako alokacija nije bila
    uspesna. */

```

```
Cvor *napravi_cvor(char *etiketa)
25 {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
        return NULL;
29
    /* Inicijalizacija polja u novom cvoru */
31     if (strlen(etiketa) >= MAX) {
        fprintf(stderr,
33             "Etiketa koju biste stavili na stek je preduga, \
bice skracena .\n");
35         etiketa[MAX - 1] = '\0';
    }
37     strcpy(novi->etiketa, etiketa);
    novi->sledeci = NULL;
39     return novi;
}

41
/* Funkcija prazni stek */
43 void oslobodi_stek(Cvor ** adresa_vrha)
{
45     Cvor *pomocni;
    while (*adresa_vrha != NULL) {
47         pomocni = *adresa_vrha;
        *adresa_vrha = (*adresa_vrha)->sledeci;
49         free(pomocni);
    }
51 }

53 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
   i ukoliko alokacija nije bila uspesna, oslobadja se sva
55 prethodno zauzeta memorija za listu cija pocetni cvor se
   nalazi na adresi adresa_vrha. */
57 void prover_i_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
{
59     if (novi == NULL) {
        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
61         oslobodi_stek(adresa_vrha);
        exit(EXIT_FAILURE);
63     }
}

65
/* Funkcija postavlja na vrh steka novu etiketu. */
67 void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
{
69     Cvor *novi = napravi_cvor(etiketa);
    prover_i_alokaciju(adresa_vrha, novi);
71     novi->sledeci = *adresa_vrha;
    *adresa_vrha = novi;
73 }

75 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
```

```

77     pokazivac razlicit od NULL, tada u niz karaktera na koji on
79     pokazuje upisuje ime etikete koja je upravo skinuta sa steka
    dok u suprotnom ne radi nista. Funkcija vraca 0 ako je stek
    prazan (pa samim tim nije bilo moguće skinuti vrednost sa
    steka) ili 1 u suprotnom. */
81 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
    {
83     Cvor *pomocni;

85     /* Pokusaj skidanja vrednost sa vrha praznog steka rezultuje
    greskom i vraca se 0. */
87     if (*adresa_vrha == NULL)
        return 0;

89     /* Ako adresa na koju se smesta etiketa nije NULL, onda se na
    tu adresu kopira etiketa sa vrha steka. */
91     if (etiketa != NULL)
93         strcpy(etiketa, (*adresa_vrha)->etiketa);

95     /* Element sa vrha steka se uklanja. */
    pomocni = *adresa_vrha;
97     *adresa_vrha = (*adresa_vrha)->sledeci;
    free(pomocni);

99     return 1;
101 }

103 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na
    vrhu steka. Ukoliko je stek prazan, vraca NULL. */
105 char *vrh_steka(Cvor * vrh)
    {
107     if (vrh == NULL)
        return NULL;
109     return vrh->etiketa;
    }

111 /* Funkcija prikazuje stek pocev od vrha prema dnu. */
113 void prikazi_stek(Cvor * vrh)
    {
115     for (; vrh != NULL; vrh = vrh->sledeci)
        printf("<%s>\n", vrh->etiketa);
117 }

119 /* Funkcija iz fajla na koji pokazuje f cita sledecu etiketu, i
    njeno ime upisuje u niz na koji pokazuje pokazivac etiketa.
121 Funkcija vraca EOF u slucaju da se dodje do kraja fajla pre
    nego sto se procita etiketa, vraca OTVORENA ako je procitana
123 otvorena etiketa, odnosno ZATVORENA ako je procitana
    zatvorena etiketa. */
125 int uzmi_etiketu(FILE * f, char *etiketa)
    {
127     int c;

```

```

129     int i = 0;

131     /* Stanje predstavlja informaciju dokle se stalo sa citanjem
132        etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos
133        uvek nije zapoceto citanje. Tip predstavlja informaciju o
134        tipu etikete uzima vrednosti OTVORENA ili ZATVORENA. */
135     int stanje = VAN_ETIKETE;
136     int tip;

137     /* HTML je neosetljiv na razliku izmedju malih i velikih
138        slova. U HTML-u etikete BODY i body imaju isto znacenje,
139        dok to u C-u ne vazi. Zato ce sve etikete biti prevedene u
140        zapis samo malim slovima. */
141     while ((c = fgetc(f)) != EOF) {
142         switch (stanje) {
143             case VAN_ETIKETE:
144                 if (c == '<')
145                     stanje = PROCITANO_MANJE;
146                 break;
147             case PROCITANO_MANJE:
148                 if (c == '/') {
149                     /* Cita se zatvarac */
150                     tip = ZATVORENA;
151                 } else {
152                     if (isalpha(c)) {
153                         /* Cita se otvarac */
154                         tip = OTVORENA;
155                         etiketa[i++] = tolower(c);
156                     }
157                 }
158                 /* Od sada se cita etiketa i zato se menja stanje. */
159                 stanje = U_ETIKETI;
160                 break;
161             case U_ETIKETI:
162                 if (isalpha(c) && i < MAX - 1) {
163                     /* Ako je procitani karakter slovo i nije premasena
164                        dozvoljena duzina etikete, procitani karakter se
165                        smanjuje i smesta u etiketu. */
166                     etiketa[i++] = tolower(c);
167                 } else {
168                     /* U suprotnom, staje se sa citanjem etikete i stanje se
169                        menja. Korektno se szavrava niska koja sadrzi
170                        procitanu etiketu i vraca se njen tip. */
171                     stanje = VAN_ETIKETE;
172                     etiketa[i] = '\\0';
173                     return tip;
174                 }
175                 break;
176         }
177     }

179     /* Doslo se do kraja datoteke pre nego sto je procitana

```

```

    naredna etiketa i vraća se EOF. */
181 return EOF;
}
183
int main(int argc, char **argv)
185 {
    /* Na početku, stek je prazan i etikete su uparene jer nijedna
187     jos nije procitana. */
    Cvor *vrh = NULL;
189     char etiketa[MAX];
    int tip;
191     int uparene = 1;
    FILE *f = NULL;
193
    /* Ime datoteke se preuzima iz komandne linije. */
195     if (argc < 2) {
        fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n",
197             argv[0]);
        exit(0);
199     }

    /* Datoteka se otvara za citanje. */
201     if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
203             argv[1]);
        exit(1);
205     }

    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
207     while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
        /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
209         etikete <br>, <hr> i <meta> koje nemaju sadrzaj, tako da
        ih nije potrebno zatvoriti. NAPOMENA: U HTML-u postoje
211         jos neke etikete koje nemaju sadrzaj (npr link).
        Pretpostavlja se da njih nema u HTML dokumentu, zbog
213         jednostavnosti. */
        if (tip == OTVORENA) {
            if (strcmp(etiketa, "br") != 0
215                 && strcmp(etiketa, "hr") != 0
                && strcmp(etiketa, "meta") != 0)
217                 potisni_na_stek(&vrh, etiketa);
            else if (tip == ZATVORENA) {
                if (vrh_steka(vrh) != NULL
219                     && strcmp(vrh_steka(vrh), etiketa) == 0)
                    skini_sa_steka(&vrh, NULL);
221             }
        }
        /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti
223         da je u pitanju zatvaranje etikete koja je poslednja
        otvorena, a jos uvek nije zatvorena. Ova etiketa se mora
225         nalaziti na vrhu steka. Ako je taj uslov ispunjen, skida
        se sa steka, jer je zatvorena. U suprotnom, pronadjena je
227         nepravilnost i etikete nisu pravilno uparene. */
        else if (tip == ZATVORENA) {
            if (vrh_steka(vrh) != NULL
229                 && strcmp(vrh_steka(vrh), etiketa) == 0)
                skini_sa_steka(&vrh, NULL);
231
        }
    }
}

```

```
233     else {
234         printf("Etikete nisu pravilno uparene\n(nadjena\
etiketa </%s>", etiketa);
235         if (vrh_steka(vrh) != NULL)
236             printf(", a poslednja otvorena etiketa je </%s>)\n",
237                 vrh_steka(vrh));
238         else
239             printf(" koja nije otvorena)\n");
240         uparene = 0;
241         break;
242     }
243 }
244 }
245 /* Završeno je citanje datoteke i zatvara se. */
246 fclose(f);
247
248 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi
249    trebalo da bude prazan. Ukoliko nije, tada postoje etikete
250    koje su ostale otvorene. */
251 if (uparene) {
252     if (vrh_steka(vrh) == NULL)
253         printf("Etikete su pravilno uparene!\n");
254     else {
255         printf("Etikete nisu pravilno uparene\n(etiketa </%s> \
nije zatvorena)\n", vrh_steka(vrh));
256         /* Oslobadja se memorija zauzeta stekom. */
257         oslobodi_stek(&vrh);
258     }
259 }
260 }
261 return 0;
262 }
```

### Rešenje 4.6

```
1  #ifndef _RED_H
2  #define _RED_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define MAX 1000
8  #define JMBG_DUZINA 14
9
10 /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG
11    korisnika i opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
```

```

korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
    Zahtev nalog;
21     struct cvor *sledeci;
} Cvor;
23
Cvor *napravi_cvor(Zahtev * zahtev);
25
void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
27
void prover_i_alokaciju(Cvor ** adresa_pocetka,
29     Cvor ** adresa_kraja, Cvor * novi);
31
void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
    Zahtev * zahtev);
33
int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
35     Zahtev * zahtev);
37
Zahtev *pocetak_reda(Cvor * pocetak);
39
void prikazi_red(Cvor * pocetak);
41
#endif

1 #include "red.h"

3 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na
   zahtev sa poslate adrese i vraca adresu novog cvora ili NULL
   ako je doslo do greske pri alokaciji. Ako je doslo do greske,
   trebalo bi osloboditi ceo red. To je ostavljeno da to uradi
   funkcija koja je pozvala funkciju napravi_cvor, a gresku ce
   biti signalizirana vracanjem NULL. Funkciji se prosledjuje
   pokazivac na zahtev koji treba smestiti u nov cvor zbog
   smestanja manjeg podatka na sistemski stek. Pokazivac na
   strukturu Zahtev je manje velicine u bajtovima(B) u odnosu na
   strukturu Zahtev. */
13 Cvor *napravi_cvor(Zahtev * zahtev)
{
15     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
17         return NULL;

19     novi->nalog = *zahtev;
    novi->sledeci = NULL;
21     return novi;
}
23

/* Funkcija prazni red */
25 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
{
27     Cvor *pomocni = NULL;

```

```
29 while (*pocetak != NULL) {
    pomocni = *pocetak;
31     *pocetak = (*pocetak)->sledeci;
    free(pomocni);
33 }
    *kraj = NULL;
35 }

37 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
   i ukoliko alokacija nije bila uspesna, oslobadja se sva
39 prethodno zauzeta memorija za listu cija pocetni cvor se
   nalazi na adresi adresa_pocetka. */
41 void prover_i_alokaciju(Cvor ** adresa_pocetka,
                        Cvor ** adresa_kraja, Cvor * novi)
43 {
    if (novi == NULL) {
45         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_red(adresa_pocetka, adresa_kraja);
47         exit(EXIT_FAILURE);
    }
49 }

51 /* Funkcija dodaje na kraj reda novi fajl. */
void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
53                Zahtev * zahtev)
{
55     Cvor *novi = napravi_cvor(zahtev);
    prover_i_alokaciju(adresa_pocetka, adresa_kraja, novi);
57
    /* U red se uvek dodaje na kraj, ali zbog postojanja
59 pokazivaca na kraj, dodavanje na kraj je podjednako
    efikasno kao dodavanje na pocetak. */
61     if (*adresa_kraja != NULL) {
        (*adresa_kraja)->sledeci = novi;
63         *adresa_kraja = novi;
    } else {
65         /* Ako je red bio ranije prazan */
        *adresa_pocetka = novi;
67         *adresa_kraja = novi;
    }
69 }

71 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji
   argument pokazivac razlicit od NULL, tada se u strukturu na
73 koju on pokazuje upisuje zahtev koji je upravo skinut sa reda
   dok u suprotnom ne upisuje nista. Funkcija vraca 0 ako je red
75 bio prazan ili 1 u suprotnom. */
int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
77                  Zahtev * zahtev)
{
79     Cvor *pomocni = NULL;
```



```

81     if (*adresa_pocetka == NULL)
82         return 0;
83
84     if (zahtev != NULL)
85         *zahtev = (*adresa_pocetka)->nalog;
86
87     pomocni = *adresa_pocetka;
88     *adresa_pocetka = (*adresa_pocetka)->sledeci;
89     free(pomocni);
90
91     if (*adresa_pocetka == NULL)
92         *adresa_kraja = NULL;
93
94     return 1;
95 }
96
97 /* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev
98    korisnika na pocetku reda. Ukoliko je red prazan, vraca NULL.
99 */
100 Zahtev *pocetak_reda(Cvor * pocetak)
101 {
102     if (pocetak == NULL)
103         return NULL;
104
105     return &(pocetak->nalog);
106 }
107
108 /* Funkcija prikazuje red. */
109 void prikazi_red(Cvor * pocetak)
110 {
111     for (; pocetak != NULL; pocetak = pocetak->sledeci)
112         printf("%s %s\n",
113             (pocetak->nalog).jmbg, (pocetak->nalog).opis);
114
115     printf("\n");
116 }

```

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #include "red.h"

6 #define VREME_ZA_PAUZU 5

8 int main(int argc, char **argv)
9 {
10     /* Red je prazan. */
11     Cvor *pocetak = NULL, *kraj = NULL;
12     Zahtev nov_zahtev;
13     Zahtev *sledeci = NULL;
14     char odgovor[3];

```

```

16     int broj_usluzenih = 0;
    FILE *izlaz = fopen("izvestaj.txt", "a");

18     if (izlaz == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke \
20 izvestaj.txt\n");
        exit(EXIT_FAILURE);
22     }

24     /* Sluzbenik evidentira korisnicke zahteve. */
    printf("Sluzbenik evidentira korisnicke zahteve unosnjem \
26 njihovog JMBG broja i opisa potrebne usluge:\n[CTRL+D za \
kraj]\n");

28     /* Neophodan je poziv funkcije getchar da bi se i nov red
30 nakon JMBG broja procitao i da bi fgets nakon toga
    procitala ispravan red sa opisom zahteva. */
32     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
    while (scanf("%s", nov_zahtev.jmbg) != EOF) {
34         getchar();
        printf("\tOpis problema: ");
36         fgets(nov_zahtev.opis, MAX - 1, stdin);
        /* Ako je poslednji karakter nov red, eliminiše se. */
38         if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
            nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
40         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
        printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
42     }

44     /* Datoteka više nije potrebna i treba je zatvoriti. */
    fclose(izlaz);

46     /* Dokle god ima korisnika u redu, treba ih usluziti. */
48     while (1) {
        sledeci = pocetak_reda(pocetak);
        /* Ako nema nikog više u redu, prekida se petlja. */
50         if (sledeci == NULL)
52             break;

54         printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
            sledeci->jmbg);
56         printf("sa zahtevom: %s\n", sledeci->opis);

58         skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

60         broj_usluzenih++;

62         printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
        scanf("%s", odgovor);

64         if (strcmp(odgovor, "Da") == 0)
66             dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
    
```

```

68     else
        fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
                nov_zahtev.jmbg, nov_zahtev.opis);
70
72     if (broj_usluzenih == VREME_ZA_PAUZU) {
        printf("\nDa li je kraj smene? [Da/Ne] ");
        scanf("%s", odgovor);
74
        if (strcmp(odgovor, "Da") == 0)
76             break;
        else
78             broj_usluzenih = 0;
    }
80 }

82 /* Prethodno usluživanje korisnika može da se izvrši i na
    sledeći način */
84 /* while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
    printf("\nSledeći je korisnik sa JMBG brojem: %s\n",
86         nov_zahtev.jmbg); printf("sa zahtevom: %s\n",
        nov_zahtev.opis);
88
        broj_usluzenih++;
90
        printf("\tDa li ga vraćate na kraj reda? [Da/Ne] ");
92         scanf("%s", odgovor);

94         if (strcmp(odgovor, "Da") == 0) dodaj_u_red(&pocetak,
            &kraj, &nov_zahtev); else fprintf(izlaz, "JMBG: %s\tZahtev:
96             %s\n", nov_zahtev.jmbg, nov_zahtev.opis);

98         if (broj_usluzenih == VREME_ZA_PAUZU) { printf("\nDa li je
            kraj smene? [Da/Ne] "); scanf("%s", odgovor);
100
            if (strcmp(odgovor, "Da") == 0) break; else broj_usluzenih
102             = 0; } } */

104 /* Ukoliko je službenik prekinuo sa radom, možda je bilo još
    neusluženih korisnika, u tom slučaju treba osloboditi
106     memoriju koju zauzima red sa neobrađenim zahtevima
    korisnika. */
108     oslobodi_red(&pocetak, &kraj);

110     return 0;
}

```

### Rešenje 4.7

```

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

```

```

4  #define MAX_DUZINA 20

6  typedef struct _Element {
7      unsigned broj_pojavljivanja;
8      char etiketa[20];
9      struct _Element *sledeci;
10 } Element;

12 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi
    cvor ili NULL ako alokacija nije uspesno izvršena. */
14 Element *napravi_cvor(unsigned br, char *etiketa)
15 {
16     Element *novi = (Element *) malloc(sizeof(Element));
17     if (novi == NULL)
18         return NULL;

20     novi->broj_pojavljivanja = br;
21     strcpy(novi->etiketa, etiketa);
22     novi->sledeci = NULL;
23     return novi;
24 }

26 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
    liste. */
28 void oslobodi_listu(Element ** glava)
29 {
30     Element *pomocni = NULL;

32     while (*glava != NULL) {
33         pomocni = (*glava)->sledeci;
34         free(*glava);
35         *glava = pomocni;
36     }
37 }

38 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
    i ukoliko alokacija nije bila uspesna, oslobadja se sva
    prethodno zauzeta memorija za listu cija pocetni cvor se
    nalazi na adresi glava. */
42 void prover_a_lokacije(Element * novi, Element ** glava)
43 {
44     if (novi == NULL) {
45         fprintf(stderr, "malloc() greska u funkciji \
napravi_cvor()!\n");
46         oslobodi_listu(glava);
47         exit(EXIT_FAILURE);
48     }
49 }

52 /* Funkcija dodaje novi cvor na pocetak liste. */
54 void dodaj_na_pocetak_liste(Element ** glava, unsigned br,
    char *etiketa)

```

```

56 {
    Element *novi = napravi_cvor(br, etiketa);
58   provera_alokacije(novi, glava);
    novi->sledeci = *glava;
60   *glava = novi;
}

62
/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu.
64   (NULL u suprotnom) */
Element *pretrazi_listu(Element * glava, char etiketa[])
66 {
    Element *tekuci;
68   for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
        if (strcmp(tekuci->etiketa, etiketa) == 0)
70       return tekuci;
    return NULL;
72 }

74 /* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Element * glava)
76 {
    for (; glava != NULL; glava = glava->sledeci)
78     printf("%s - %u\n", glava->etiketa,
            glava->broj_pojavljivanja);
80 }

82 int main(int argc, char **argv)
{
84     if (argc != 2) {
        fprintf(stderr, "Greska! Program se poziva sa: ./a.out \
86     datoteka.html!\n");
        exit(EXIT_FAILURE);
88     }

90     FILE *in = NULL;
    in = fopen(argv[1], "r");
92     if (in == NULL) {
        fprintf(stderr,
94         "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
        exit(EXIT_FAILURE);
96     }

98     char c;
    int i = 0;
100    char a[MAX_DUZINA];

102    Element *glava = NULL;
    Element *trazeni = NULL;
104

    while ((c = fgetc(in)) != EOF) {
106        if (c == '<') {

```

```
108      /* Cita se zatvarac. */
109      if ((c = fgetc(in)) == '/') {
110          i = 0;
111          while ((c = fgetc(in)) != '>')
112              a[i++] = c;
113      }
114      /* Cita se otvarac. */
115      else {
116          i = 0;
117          a[i++] = c;
118          while ((c = fgetc(in)) != ' ' && c != '>')
119              a[i++] = c;
120      }
121      a[i] = '\0';
122
123      /* Trazi se učitana etiketa medju postojećim cvorovima
124       liste. Ukoliko ne postoji, dodaje se novi cvor za
125       učitanu etiketu sa brojem pojavljivanja 1, inace
126       uvecava se broj pojavljivanja etikete. */
127      trazeni = pretrazi_listu(glava, a);
128      if (trazeni == NULL)
129          dodaj_na_pocetak_liste(&glava, 1, a);
130      else
131          trazeni->broj_pojavljivanja++;
132  }
133  }
134
135  fclose(in);
136
137  ispisi_listu(glava);
138  oslobodi_listu(&glava);
139
140  return 0;
141  }
```

### Rešenje 4.8

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4
5  #define MAX_INDEKS 11
6  #define MAX_IME_PREZIME 21
7
8  typedef struct _Cvor {
9      char broj_indeksa[MAX_INDEKS];
10     char ime[MAX_IME_PREZIME];
11     char prezime[MAX_IME_PREZIME];
12     struct _Cvor *sledeci;
13 } Cvor;
14
```

```

16  /* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac
    na nov cvor ili NULL ukoliko alokacija nije prosla. */
17  Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
18  {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
    {
    20      return NULL;
    }
    strcpy(novi->broj_indeksa, broj_indeksa);
    strcpy(novi->ime, ime);
    24     strcpy(novi->prezime, prezime);
    novi->sledeci = NULL;
    26     return novi;
    }

28  /* Funkcija oslobadja memoriju zauzetu za elemente liste. */
    void oslobodi_listu(Cvor ** glava)
    {
    32         if (*glava == NULL)
            return;
    34         oslobodi_listu(&(*glava)->sledeci);
        free(*glava);
    36         *glava = NULL;
    }

38  void prover_i_alokaciju(Cvor ** glava, Cvor * novi)
    {
    40         if (novi == NULL) {
            fprintf(stderr, "Neuspela alokacija za nov cvor\n");
            oslobodi_listu(glava);
            exit(EXIT_FAILURE);
        }
    46     }

48  /* Funkcija dodaje novi cvor na pocetak liste. */
    void dodaj_na_pocetak_liste(Cvor ** glava, char *broj_indeksa,
    50                               char *ime, char *prezime)
    {
    52         Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
        prover_i_alokaciju(glava, novi);
    54         novi->sledeci = *glava;
        *glava = novi;
    56     }

58  void ispisi_listu(Cvor * glava)
    {
    60         for (; glava != NULL; glava = glava->sledeci)
            printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
    62                               glava->prezime);
    }

64  /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu,
    u suprotnom vraca NULL. */
66

```

```
Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
68 {
    if (glava == NULL)
70         return NULL;
    if (!strcmp(glava->broj_indeksa, broj_indeksa))
72         return glava;
    return pretrazi_listu(glava->sledeci, broj_indeksa);
74 }

int main(int argc, char **argv)
{
76     if (argc != 2) {
78         fprintf(stderr, "Greska! Program se poziva sa: ./a.out \
80 student_i.txt!\n");
        exit(EXIT_FAILURE);
82     }

84     FILE *in = NULL;
    in = fopen(argv[1], "r");
86     if (in == NULL) {
        fprintf(stderr,
88             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
90     }

92     char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
    char broj_indeksa[MAX_INDEKS];
94     Cvor *glava = NULL;
    Cvor *trazeni = NULL;
96

    /* Ucitavanje vrednosti u listu. */
98     while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) !=
        EOF)
100         dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);

102     fclose(in);

104     while (scanf("%s", broj_indeksa) != EOF) {
        trazeni = pretrazi_listu(glava, broj_indeksa);
106         if (trazeni == NULL)
            printf("ne\n");
108         else
            printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
110     }

112     oslobodi_listu(&glava);

114     return 0;
}
```

### Rešenje 4.9



```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include "601/lista.h"
4
5  Cvor *objedini(Cvor ** glava1, Cvor ** glava2)
6  {
7      Cvor *l3 = NULL;
8      Cvor **tek = &l3;
9
10     if (*glava1 == NULL && *glava2 == NULL)
11         return NULL;
12
13     /* Ako je prva lista prazna, rezultat je druga lista. */
14     if (*glava1 == NULL)
15         return *glava2;
16
17     /* Ako je druga lista prazna, rezultat je prva lista. */
18     if (*glava2 == NULL)
19         return *glava1;
20
21     /* l3 pokazuje na pocetak nove liste, tj. na manji od brojeva
22        sadrzanih u cvorovima na koje pokazuju glava1 i glava2. */
23     l3 = ((*glava1)->vrednost < (*glava2)->vrednost) ? *glava1 :
24         *glava2;
25
26     while (*glava1 != NULL && *glava2 != NULL) {
27         if ((*glava1)->vrednost < (*glava2)->vrednost) {
28             *tek = *glava1;
29             *glava1 = (*glava1)->sledeci;
30         } else {
31             *tek = *glava2;
32             *glava2 = (*glava2)->sledeci;
33         }
34         tek = &((*tek)->sledeci);
35     }
36
37     /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste,
38        na rezultujucu listu treba nadovezati ostatak druge liste. */
39     if (*glava1 == NULL)
40         *tek = *glava2;
41
42     else if (*glava2 == NULL)
43         *tek = *glava1;
44
45     return l3;
46 }
47
48 int main(int argc, char **argv)
49 {
50     if (argc != 3) {
```

```
53     fprintf(stderr,
        "Greska! Program se poziva sa: ./a.out dat1.txt dat2.txt
        !\n");
        exit(EXIT_FAILURE);
55 }

57 FILE *in1 = NULL;
    in1 = fopen(argv[1], "r");
59 if (in1 == NULL) {
        fprintf(stderr,
61         "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
63 }

65 FILE *in2 = NULL;
    in2 = fopen(argv[2], "r");
67 if (in2 == NULL) {
        fprintf(stderr,
69         "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
        exit(EXIT_FAILURE);
71 }

73 int broj;
    Cvor *glava1 = NULL;
75 Cvor *glava2 = NULL;
    Cvor *l3 = NULL;

77 /* Ucitavanje listi */
79 while (fscanf(in1, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&glava1, broj);
81 while (fscanf(in2, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&glava2, broj);
83
    l3 = objedini(&glava1, &glava2);

85 /* Ispis rezultujuće liste. */
87 ispisi_listu(l3);
    oslobodi_listu(&l3);
89
    fclose(in1);
    fclose(in2);
91     return 0;
93 }
```

Rešenje 4.10

Rešenje 4.11

Rešenje 4.12

## Rešenje 4.13

## Rešenje 4.14

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* a) Struktura kojom se predstavlja cvor binarnog
   pretrazivackog stabla */
6 typedef struct cvor {
7     int broj;
8     struct cvor *levo;
9     struct cvor *desno;
10 } Cvor;
11
12 /* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj)
15 {
16     /* Alociramo memoriju */
17     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
18     if (novi == NULL)
19         return NULL;
20
21     /* Inicijalizujemo polja novog cvora. */
22     novi->broj = broj;
23     novi->levo = NULL;
24     novi->desno = NULL;
25
26     /* Vracamo adresu novog cvora. */
27     return novi;
28 }
29
30 /* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
32 void prover_i_alokaciju(Cvor * novi_cvor)
33 {
34     /* Ukoliko je cvor neuspesno kreiran */
35     if (novi_cvor == NULL) {
36
37         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
           programa */
38         fprintf(stderr, "Malloc greska za novi cvor!\n");
39         exit(EXIT_FAILURE);
40     }
41 }
42
43
44
45 /* c) Funkcija koja dodaje zadati broj u stablo */
46 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
```

```

48 {
   /* Ako je stablo prazno */
50 if (*adresa_korena == NULL) {

   /* Kreiramo novi cvor */
   Cvor *novi = napravi_cvor(broj);
54   prover_i_alokaciju(novi);

   /* I proglašavamo ga korenom stabla */
   *adresa_korena = novi;
58   return;
   }

60
   /* U suprotnom tražimo odgovarajuću poziciju za zadati broj */
62
   /* Ako je zadata vrednost manja od vrednosti korena */
64 if (broj < (*adresa_korena)->broj)

   /* Dodajemo broj u levo podstablo */
   dodaj_u_stablo(&(*adresa_korena)->levo, broj);
68
   else
70   /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa
      ga dodajemo u desno podstablo */
72   dodaj_u_stablo(&(*adresa_korena)->desno, broj);
   }

74

76 /* d) Funkcija koja proverava da li se zadati broj nalazi u
   stablu */
78 Cvor *pretrazi_stablo(Cvor * koren, int broj)
   {
80   /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
      */
82   if (koren == NULL)
       return NULL;
84

   /* Ako je trazena vrednost sadržana u korenu */
86   if (koren->broj == broj) {

88     /* Prekidamo pretragu */
       return koren;
90   }

92   /* Inace, ako je broj manji od vrednosti sadržane u korenu */
   if (broj < koren->broj)

94     /* Pretragu nastavljamo u levom podstablu */
       return pretrazi_stablo(koren->levo, broj);
96
   else
98     /* U suprotnom, pretragu nastavljamo u desnom podstablu */

```

```
100     return pretrazi_stablo(koren->desno, broj);
101 }
102
103 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost u
104     stablu */
105 Cvor *pronadji_najmanji(Cvor * koren)
106 {
107     /* Ako je stablo prazno, prekidamo pretragu */
108     if (koren == NULL)
109         return NULL;
110
111     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze
112         se levo od njega */
113
114     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
115         najmanju vrednost */
116     if (koren->levo == NULL)
117         return koren;
118
119     /* Inace, pretragu treba nastaviti u levom podstablu */
120     return pronadji_najmanji(koren->levo);
121 }
122
123 /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u
124     stablu */
125 Cvor *pronadji_najveci(Cvor * koren)
126 {
127     /* Ako je stablo prazno, prekidamo pretragu */
128     if (koren == NULL)
129         return NULL;
130
131     /* Vrednosti koje su veće od vrednosti u korenu stabla nalaze
132         se desno od njega */
133
134     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
135         najveću vrednost */
136     if (koren->desno == NULL)
137         return koren;
138
139     /* Inace, pretragu treba nastaviti u desnom podstablu */
140     return pronadji_najveci(koren->desno);
141 }
142
143 /* g) Funkcija koja briše cvor stabla koji sadrži zadati broj */
144 void obrisi_element(Cvor ** adresa_korena, int broj)
145 {
146     Cvor *pomocni_cvor = NULL;
147
148     /* ako je stablo prazno, brisanje nije primenljivo pa mozemo
```

```

152     prekinuti rad funkcije */
153     if (*adresa_korena == NULL)
154         return;

156     /* Ako je vrednost koju treba obrisati manja od vrednosti u
157        korenu stabla, ona se eventualno nalazi u levom podstablu,
158        pa treba rekurzivno primeniti postupak na levo podstablo.
159        Koren ovako modifikovanog stabla je nepromenjen. */
160     if (broj < (*adresa_korena)->broj) {
161         obrisi_element(&(*adresa_korena)->levo, broj);
162         return;
163     }

164

166     /* Ako je vrednost koju treba obrisati veca od vrednosti u
167        korenu stabla, ona se eventualno nalazi u desnom podstablu
168        pa treba rekurzivno primeniti postupak na desno podstablo.
169        Koren ovako modifikovanog stabla je nepromenjen. */
170     if ((*adresa_korena)->broj < broj) {
171         obrisi_element(&(*adresa_korena)->desno, broj);
172         return;
173     }

174     /* Slede podslucajevi vezani za slucaj kada je vrednost u
175        korenu jednaka broju koji se brise (tj. slucaj kada treba
176        obrisati koren) */

178     /* Ako koren nema sinova, tada se on prosto brise, i rezultat
179        je prazno stablo (vracamo NULL) */
180     if ((*adresa_korena)->levo == NULL
181         && (*adresa_korena)->desno == NULL) {
182         free(*adresa_korena);
183         *adresa_korena = NULL;
184         return;
185     }

186

188     /* Ako koren ima samo levog sina, tada se brisanje vrši tako
189        sto obrisemo koren, a novi koren postaje levi sin */
190     if ((*adresa_korena)->levo != NULL
191         && (*adresa_korena)->desno == NULL) {
192         pomocni_cvor = (*adresa_korena)->levo;
193         free(*adresa_korena);
194         *adresa_korena = pomocni_cvor;
195         return;
196     }

197

198     /* Ako koren ima samo desnog sina, tada se brisanje vrši tako
199        sto obrisemo koren, a novi koren postaje desni sin */
200     if ((*adresa_korena)->desno != NULL
201         && (*adresa_korena)->levo == NULL) {
202         pomocni_cvor = (*adresa_korena)->desno;
203         free(*adresa_korena);
204         *adresa_korena = pomocni_cvor;

```

```

204     return;
205 }
206
207 /* Slučaj kada koren ima oba sina. Tada se brisanje vrši na
208    sledeći način: - najpre se potraži sledbenik korena (u
209    smislu poretka) u stablu. To je upravo po vrednosti
210    najmanji cvor u desnom podstablu. On se može pronaći npr.
211    funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
212    vrednost tog cvora, a u taj cvor se smesti vrednost korena
213    (tj. broj koji se briše). - Onda se prosto rekurzivno
214    pozove funkcija za brisanje na desno podstablo. S obzirom
215    da u njemu treba obrisati najmanji element, a on zasigurno
216    ima najviše jednog potomka, jasno je da će njegovo brisanje
217    biti obavljeno na jedan od jednostavnijih načina koji su
218    gore opisani. */
219 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
220 (*adresa_korena)->broj = pomocni_cvor->broj;
221 pomocni_cvor->broj = broj;
222 obrisi_element(&(*adresa_korena)->desno, broj);
223 }
224
225 /* h) Funkcija ispisuje stablo u infiksnoj notaciji ( Levo
226    postablo - Koren - Desno podstablo ) */
227 void ispisi_stablo_infiksno(Cvor * koren)
228 {
229     /* Ako stablo nije prazno */
230     if (koren != NULL) {
231
232         /* Prvo ispisujemo sve cvorove levo od korena */
233         ispisi_stablo_infiksno(koren->levo);
234
235         /* Ispisujemo vrednost u korenu */
236         printf("%d ", koren->broj);
237
238         /* Na kraju ispisujemo cvorove desno od korena */
239         ispisi_stablo_infiksno(koren->desno);
240     }
241 }
242
243
244 /* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren -
245    Levo podstablo - Desno podstablo ) */
246 void ispisi_stablo_prefiksno(Cvor * koren)
247 {
248     /* Ako stablo nije prazno */
249     if (koren != NULL) {
250
251         /* Prvo ispisujemo vrednost u korenu */
252         printf("%d ", koren->broj);
253
254         /* Ispisujemo sve cvorove levo od korena */

```

```
256     ispisi_stablo_prefiksno(koren->levo);

258     /* Na kraju ispisujemo sve cvorove desno od korena */
    ispisi_stablo_prefiksno(koren->desno);
260 }
262

264 /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo
    podstablo - Desno podstablo - Koren) */
266 void ispisi_stablo_postfiksno(Cvor * koren)
{
268     /* Ako stablo nije prazno */
    if (koren != NULL) {

270         /* Prvo ispisujemo sve cvorove levo od korena */
272         ispisi_stablo_postfiksno(koren->levo);

274         /* Ispisujemo sve cvorove desno od korena */
        ispisi_stablo_postfiksno(koren->desno);

276         /* Na kraju ispisujemo vrednost u korenu */
278         printf("%d ", koren->broj);
    }
280 }

282 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
284 void oslobodi_stablo(Cvor ** adresa_korena)
{
286     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*adresa_korena == NULL)
288         return;

290     /* U suprotnom rekurzivno oslobadjamo memoriju koje zauzima
        najpre levo, a zatim i desno podstablo */
292     oslobodi_stablo(&(*adresa_korena)->levo);
    oslobodi_stablo(&(*adresa_korena)->desno);
294

296     /* Oslobadjamo memoriju koju zauzima koren */
    free(*adresa_korena);

298     /* I proglašavamo stablo praznim */
    *adresa_korena = NULL;
300 }

302 int main()
{
304     Cvor *koren;
    int n;
306     Cvor *trazeni_cvor;
```



```

308  /* Proglasavamo stablo praznim */
      koren = NULL;

310

312  /* Dodajemo vrednosti u stablo */
      printf("Unesite brojeve (CTRL+D za kraj unosa): ");
      while (scanf("%d", &n) != EOF) {
314          dodaj_u_stablo(&koren, n);
      }

316

318  /* Generisemo trazene ispise: */
      printf("\nInfiksni ispis: ");
      ispisi_stablo_infiksno(koren);
320      printf("\nPrefiksni ispis: ");
      ispisi_stablo_prefiksno(koren);
322      printf("\nPostfiksni ispis: ");
      ispisi_stablo_postfiksno(koren);

324

326  /* Demonstriramo rad funkcije za pretragu */
      printf("\nTrazi se broj: ");
      scanf("%d", &n);
      trazeni_cvor = pretrazi_stablo(koren, n);
      if (trazeni_cvor == NULL)
330          printf("Broj se ne nalazi u stablu!\n");
      else
332          printf("Broj se nalazi u stablu!\n");

334  /* Demonstriramo rad funkcije za brisanje */
      printf("Brise se broj: ");
      scanf("%d", &n);
      obrisi_element(&koren, n);
338      printf("Rezultujuce stablo: ");
      ispisi_stablo_infiksno(koren);
340      printf("\n");

342  /* Oslobadjamo memoriju zauzetu stablom */
      oslobodi_stablo(&koren);

344

346  /* Prekidamo sa programom */
      return 0;
}

```

### Rešenje 4.15

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <ctype.h>
5
   #define MAX 50
7
   /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj

```

```
9      pojavljivanja i redom pokazivace na levo i desno podstablo */
typedef struct cvor {
11     char *rec;
    int brojac;
13     struct cvor *levo;
    struct cvor *desno;
15 } Cvor;

17 /* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *rec)
19 {
    /* Alociramo memoriju za novi cvor */
21     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
    if (novi_cvor == NULL)
23         return NULL;

25     /* Alociramo memoriju za zadatu rec: potrebno je rezervisati
        memoriju za svaki karakter reci ukljucujuci i terminirajucu
        nulu */
27     novi_cvor->rec =
29         (char *) malloc((strlen(rec) + 1) * sizeof(char));
    if (novi_cvor->rec == NULL) {
31         free(novi_cvor);
        return NULL;
33     }

35     /* Inicijalizujemo polja u novom cvoru */
    strcpy(novi_cvor->rec, rec);
37     novi_cvor->brojac = 1;
    novi_cvor->levo = NULL;
39     novi_cvor->desno = NULL;

41     /* Vracamo adresu novog cvora */
    return novi_cvor;
43 }

45 /* Funkcija koja proverava uspesnost kreiranja novog cvora
    stabla */
47 void proveri_alokaciju(Cvor * novi_cvor)
{
49     /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
51         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
            programa */
53         fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
55     }
}

57 /* Funkcija koja dodaje novu rec u stablo. */
59 void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
{

```

```

61  /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
63      /* Kreiramo novi cvor */
        Cvor *novi = napravi_cvor(rec);
65      prover_i_alokaciju(novi);

        /* i proglašavamo ga korenom stabla */
        *adresa_korena = novi;
69      return;
    }

71  /* U suprotnom trazimo odgovarajucu poziciju za novu rec */
73
75  /* Ako je rec leksikografski manju od reci u korenu ubacujemo
    je u levo podstablo */
    if (strcmp(rec, (*adresa_korena)->rec) < 0)
77      dodaj_u_stablo(&(*adresa_korena)->levo, rec);

79  else
        /* Ako je rec leksikografski veca od reci u korenu ubacujemo
        je u desno podstablo */
81      if (strcmp(rec, (*adresa_korena)->rec) > 0)
93          dodaj_u_stablo(&(*adresa_korena)->desno, rec);

85  else
        /* Ako je rec jednaka reci u korenu, uvecavamo njen broj
        pojavljivanja */
87      (*adresa_korena)->brojac++;
89  }

91  /* Funkcija koja oslobadja memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** adresa_korena)
93  {
        /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
95      if (*adresa_korena == NULL)
          return;

97      /* Inace ... */
99      /* Oslobadjamo memoriju zauzetu levim podstablom */
        oslobodi_stablo(&(*adresa_korena)->levo);

101     /* Oslobadjamo memoriju zauzetu desnim podstablom */
103     oslobodi_stablo(&(*adresa_korena)->desno);

105     /* Oslobadjamo memoriju zauzetu korenom */
        free((*adresa_korena)->rec);
107     free(*adresa_korena);

109     /* Proglašavamo stablo praznim */
        *adresa_korena = NULL;
111 }

```

```

113  /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec
115  (rec sa najvećim brojem pojavljivanja) */
Cvor *nadjij_najfrekventniju_rec(Cvor * koren)
117  {
    Cvor *max, *max_levo, *max_desno;

119
    /* Ako je stablo prazno, prekidamo sa pretragom */
121    if (koren == NULL)
        return NULL;

123
    /* Pronalazimo najfrekventniju reci u levom podstablu */
125    max_levo = nadjij_najfrekventniju_rec(koren->levo);

127    /* Pronalazimo najfrekventniju reci u desnom podstablu */
    max_desno = nadjij_najfrekventniju_rec(koren->desno);

129
    /* Trazimo maksimum vrednosti pojavljivanja reci iz levog
131    podstabla, korena i desnog podstabla */
    max = koren;
133    if (max_levo != NULL && max_levo->brojac > max->brojac)
        max = max_levo;
135    if (max_desno != NULL && max_desno->brojac > max->brojac)
        max = max_desno;

137
    /* Vracamo adresu cvora sa najvećim brojcem */
139    return max;
}

141

143  /* Funkcija koja ispisuje reci iz stabla u leksikografskom
    poretku pracene brojem pojavljivanja */
145  void prikazi_stablo(Cvor * koren)
    {
147      /* Ako je stablo prazno, završavamo sa ispisom */
      if (koren == NULL)
149          return;

151      /* Zbog leksikografskog poretka, prvo ispisujemo sve reci iz
        levog podstabla */
153      prikazi_stablo(koren->levo);

155      /* Zatim ispisujemo rec iz korena */
      printf("%s: %d\n", koren->rec, koren->brojac);

157
      /* I nastavljamo sa ispisom reci iz desnog podstabla */
159      prikazi_stablo(koren->desno);
    }

161

163  /* Funkcija ucitava sledecu rec iz zadate datoteke i upisuje je
    u niz rec. Maksimalna duzina reci je odredjena argumentom

```

```
165     max. Funkcija vraca EOF ako nema vise reci ili 0 u suprotnom.  
166     Rec je niz malih ili velikih slova. */  
167 int procitaj_rec(FILE * f, char rec[], int max)  
168 {  
169     /* karakter koji citamo */  
170     int c;  
171  
172     /* indeks pozicije na koju se smesta procitani karakter */  
173     int i = 0;  
174  
175     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle god  
176        nismo stigli do kraja datoteke... */  
177     while (i < max - 1 && (c = fgetc(f)) != EOF) {  
178         /* Proveravamo da li je procitani karakter slovo */  
179         if (isalpha(c))  
180  
181             /* Ako jeste, smestamo ga u niz - pritom vrsimo konverziju  
182                u mala slova jer program treba da bude neosetljiv na  
183                razliku izmedju malih i velikih slova */  
184             rec[i++] = tolower(c);  
185  
186         else  
187             /* Ako nije, proveravamo da li smo procitali barem jedno  
188                slovo nove rece */  
189             /* Ako jesmo prekidamo sa citanjem */  
190             if (i > 0)  
191                 break;  
192  
193         /* U suprotnom idemo na sledecu iteraciju */  
194     }  
195  
196     /* Dodajemo na rec terminirajucu nulu */  
197     rec[i] = '\0';  
198  
199     /* Vracamo 0 ako smo procitali rec, EOF u suprotnom */  
200     return i > 0 ? 0 : EOF;  
201 }  
202  
203 int main(int argc, char **argv)  
204 {  
205     Cvor *koren = NULL, *max;  
206     FILE *f;  
207     char rec[MAX];  
208  
209     /* Proveravamo da li je navedeno ime datoteke prilikom  
210        pokretanja programa */  
211     if (argc < 2) {  
212         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");  
213         exit(EXIT_FAILURE);  
214     }  
215  
216     /* Otvaramo datoteku iz koje citamo reci */
```

```

217     if ((f = fopen(argv[1], "r")) == NULL) {
218         fprintf(stderr, "fopen() greska pri otvaranju %s\n",
219             argv[1]);
220         exit(EXIT_FAILURE);
221     }

222     /* Ucitavamo reci iz datoteke i smestamo u binarno stablo
223        pretrage. */
224     while (procitaj_rec(f, rec, MAX) != EOF)
225         dodaj_u_stablo(&koren, rec);

226     /* Posto smo zavrшили sa citanjem reci zatvaramo datoteku */
227     fclose(f);

228     /* Prikazujemo sve reci iz teksta i brojeve njihovih
229        pojavljivanja. */
230     prikazi_stablo(koren);

231     /* Pronalazimo najfrekventniju rec */
232     max = nadji_najfrekventniju_rec(koren);

233     /* Ako takve reci nema... */
234     if (max == NULL)

235         /* Ispisujemo odgovarajuće obvestenje */
236         printf("U tekstu nema reci!\n");

237     else

238         /* Inace, ispisujemo broj pojavljivanja reci */
239         printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
240             max->rec, max->brojac);

241     /* Oslobadjamo dinamički alociran prostor za stablo */
242     oslobodi_stablo(&koren);

243     /* Završavamo sa programom */
244     return 0;
245 }

```

### Rešenje 4.16

```

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

6 #define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
8 #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,

```

```

    broj telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
    char ime_i_prezime[MAX_IME_I_PREZIME];
14     char telefon[MAX_CIFARA];
    struct cvor *levo;
16     struct cvor *desno;
} Cvor;
18
/* Funkcija koja kreira novi cvora stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
22     /* Alociramo memoriju za novi cvor */
    Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
24     if (novi_cvor == NULL)
        return NULL;

    /* Inicijalizujemo polja u novom cvoru */
28     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
    strcpy(novi_cvor->telefon, telefon);
30     novi_cvor->levo = NULL;
    novi_cvor->desno = NULL;

    /* Vracamo adresu novog cvora */
34     return novi_cvor;
}
36

/* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
38 void prover_i_alokaciju(Cvor * novi_cvor)
{
42     /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
44         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
           programa */
46         fprintf(stderr, "Malloc greska za novi cvor!\n");
         exit(EXIT_FAILURE);
48     }
}
50

/* Funkcija koja dodaje novu osobu i njen broj telefona u
   stablo. */
52 void
dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
54             char *telefon)
{
56     /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
58         /* Kreiramo novi cvor */
        Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
60         prover_i_alokaciju(novi);
62

```

```

64     /* i proglašavamo ga korenom stabla */
        *adresa_korena = novi;
66     return;
    }

68     /* U suprotnom trazimo odgovarajucu poziciju za novi unos */
70     /* Kako pretragu treba vrsiti po imenu i prezimenu, stablo
        treba da bude pretrazivacko po ovom polju */
72     /* Ako je zadato ime i prezime leksikografski manje od imena i
        prezimena sadrzanog u korenu, podatke dodajemo u levo
        podstablo */
74     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
76         < 0)
        dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
78                     telefon);

80     else
        /* Ako je zadato ime i prezime leksikografski vece od imena
82         i prezimena sadrzanog u korenu, podatke kodajemo u desno
        podstablo */
84     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
        dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
86                     telefon);
    }

88

90     /* Funkcija koja oslobadja memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** adresa_korena)
92    {
        /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
94        if (*adresa_korena == NULL)
            return;

96        /* Inace ... */
98        /* Oslobadjamo memoriju zauzetu levim podstablom */
        oslobodi_stablo(&(*adresa_korena)->levo);

100        /* Oslobadjamo memoriju zauzetu desnim podstablom */
102        oslobodi_stablo(&(*adresa_korena)->desno);

104        /* Oslobadjamo memoriju zauzetu korenom */
        free(*adresa_korena);

106        /* Proglašavamo stablo praznim */
108        *adresa_korena = NULL;
    }

110

112     /* Funkcija koja ispisuje imenik u leksikografskom poretku */
    /* Napomena: ova funkcija nije trazena u zadatku ali se moze
114     koristiti za proveru da li je stablo lepo kreirano ili ne */

```



```

void prikazi_stablo(Cvor * koren)
116 {
    /* Ako je stablo prazno, završavamo sa ispisom */
118     if (koren == NULL)
        return;

120     /* Zbog leksikografskog poretka, prvo ispisujemo podatke iz
122        levog podstabla */
    prikazi_stablo(koren->levo);

124     /* Zatim ispisujemo podatke iz korena */
126     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

128     /* I nastavljamo sa ispisom podataka iz desnog podstabla */
    prikazi_stablo(koren->desno);
130 }

132 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
134     ime i prezime i broj telefona u odgovarajuće nizove.
    Maksimalna dužina imena i prezimena određena je konstantom
136     MAX_IME_PREZIME, a maksimalna dužina broja telefona
    konstantom MAX_CIFARA. Funkcija vraća EOF ako nema više
138     kontakata ili 0 u suprotnom. */
int procitaj_kontakt(FILE * f, char *ime_i_prezime,
140                     char *telefon)
{
142     int c;
    int i = 0;

144     /* Linije datoteke koje obradjujemo su formata Ime Prezime
146        BrojTelefona */
    /* Preskacemo eventualne praznine sa pocetka linije datoteke */
148     while ((c = fgetc(f)) != EOF && isspace(c));

150     /* Prvo procitano slovo upisujemo u ime i prezime */
    if (!feof(f))
152         ime_i_prezime[i++] = c;

154     /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
    cifre, tako da cemo citanje forsirati sve dok ne naidjemo
156     na cifru. Pri tom cemo voditi racuna da li ima dovoljno
    mesta za smestanje procitanog karaktera i da slucajno ne
158     dodjemo do kraja datoteke */
    while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
160         if (!isdigit(c))
            ime_i_prezime[i++] = c;

162         else if (i > 0)
            break;
164     }
166 }

```

```

168  /* Upisujemo terminirajucu nulu na mesto poslednjeg procitanog
      blanko karaktera */
      ime_i_prezime[--i] = '\0';
170
172  /* I pocinjemo sa citanjem broja telefona */
      i = 0;
174
176  /* Upisujemo cifru koju smo vec procitali */
      telefon[i++] = c;
178
180  /* I citamo peostale cifre - naznaka kraja ce nam biti pojava
      karaktera cije prisustvo nije dozvoljeno u broju telefona */
      while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
182      if (c == '/' || c == '-' || isdigit(c))
184          telefon[i++] = c;
      else
186          break;
188
190  /* Upisujemo terminirajucu nulu */
      telefon[i] = '\0';
192
194  /* Vracamo 0 ako smo procitali kontakt, EOF u suprotnom */
      return !feof(f) ? 0 : EOF;
196  }
198
199  /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
      prezimenom */
      Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
200  {
202      /* Ako je imenik prazan, završavamo sa pretragom */
      if (koren == NULL)
204          return NULL;
206
208      /* Ako je trazeno ime i prezime sadržano u korenu, takodje
      završavamo sa pretragom */
      if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
210          return koren;
212
214      /* Ako je zadato ime i prezime leksikografski manje od
      vrednosti u korenu pretragu nastavljamo levo */
      if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
216          return pretrazi_imenik(koren->levo, ime_i_prezime);
218
219      else
220          /* u suprotnom, pretragu nastavljamo desno */
221          return pretrazi_imenik(koren->desno, ime_i_prezime);
222  }
223
224  int main(int argc, char **argv)
225  {

```

```

220 char ime_datoteke[MAX_IME_DATOTEKE];
    Cvor *koren = NULL;
    Cvor *trazeni;
222 FILE *f;
    char ime_i_prezime[MAX_IME_I_PREZIME];
224 char telefon[MAX_CIFARA];
    char c;
226 int i;

228 /* Ucitavamo ime datoteke i pripremamo je za citanje */
    printf("Unesite ime datoteke: ");
230 scanf("%s", ime_datoteke);
    if ((f = fopen(ime_datoteke, "r")) == NULL) {
232         fprintf(stderr, "fopen() greska prilikom otvaranja
%s\n", ime_datoteke);
234         exit(EXIT_FAILURE);
    }

236 /* Ucitavamo podatke iz datoteke i smestamo kontakte u binarno
    stablo pretrage. */
238 while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
240     dodaj_u_stablo(&koren, ime_i_prezime, telefon);

242 /* Posto smo zavrшили sa citanjem podataka zatvaramo datoteku */
    fclose(f);

244 /* Omogucavamo pretragu imenika */
    while (1) {
246         /* Ucitavamo ime i prezime */
        printf("Unesite ime i prezime: ");
        i = 0;
250         while ((c = getchar()) != '\n')
            ime_i_prezime[i++] = c;
252         ime_i_prezime[i] = '\0';

254         /* Ako je korisnik uneo naznaku za kraj pretrage,
            obustavljamo funkcionalnost */
256         if (strcmp(ime_i_prezime, "KRAJ") == 0)
            break;

258         /* Inace, ispisujemo rezultat pretrage */
        trazeni = pretrazi_imenik(koren, ime_i_prezime);
        if (trazeni == NULL)
262             printf("Broj nije u imeniku!\n");

264         else
            printf("Broj je: %s \n", trazeni->telefon);
266     }

268 /* Oslobadjamo memoriju zauzetu imenikom */
    oslobodi_stablo(&koren);
270

```

```
272  /* Završavamo sa programom */  
    return 0;  
}
```

### Rešenje 4.17

```
#include<stdio.h>  
2 #include<stdlib.h>  
  #include<string.h>  
4  
  #define MAX 51  
6  
  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime  
8     studenta, ukupan uspsih, uspeh iz matematike, uspeh iz  
    maternjeg jezika i redom pokazivace na levo i desno podstablo  
10  */  
  typedef struct cvor_stabla {  
12     char ime[MAX];  
    char prezime[MAX];  
14     double uspeh;  
    double matematika;  
16     double jezik;  
    struct cvor_stabla *levo;  
18     struct cvor_stabla *desno;  
  } Cvor;  
20  
  /* Funkcija kojom se kreira cvor stabla */  
22  Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,  
    double matematika, double jezik)  
24  {  
    /* Alociramo memoriju za novi cvor */  
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));  
    if (novi == NULL)  
28         return NULL;  
  
30     /* Inicijalizujemo polja strukture */  
    strcpy(novi->ime, ime);  
32     strcpy(novi->prezime, prezime);  
    novi->uspeh = uspeh;  
34     novi->matematika = matematika;  
    novi->jezik = jezik;  
36     novi->levo = NULL;  
    novi->desno = NULL;  
38  
    /* Vracamo adresu kreiranog cvora */  
40     return novi;  
  }  
42  
  /* Funkcija kojom se proverava uspesnost alociranja memorije */  
44  void proveri_alokaciju(Cvor * novi_cvor)  
  {
```

```

46  /* Ako alokacije nije uspesna */
    if (novi_cvor == NULL) {
48      /* Ispisujemo poruku i prekidamo sa izvršavanjem */
      fprintf(stderr, "Malloc greska za novi cvor!\n");
50      exit(EXIT_FAILURE);
    }
52  }
54
55  /* Funkcija kojom se oslobadja memorija zauzeta stablom */
56  void oslobodi_stablo(Cvor ** koren)
57  {
58      /* Ako je stablo prazno, nema potrebe za oslobadjanjem
        memorije */
60      if (*koren == NULL)
        return;
62
        /* oslobadjamo memoriju zauzetu levim podstablom */
64      oslobodi_stablo(&(*koren)->levo);
66
        /* oslobadjamo memoriju zauzetu desnim podstablom */
        oslobodi_stablo(&(*koren)->desno);
68
        /* oslobadjamo memoriju zauzetu korenom */
70      free(*koren);
72
        /* proglašavamo stablo praznim */
        *koren = NULL;
74  }
76
77  /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
78  void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
79                      double uspeh, double matematika,
80                      double jezik)
81  {
82      /* Ako je stablo prazno */
83      if (*koren == NULL) {
84          /* Kreiramo novi cvor */
85          Cvor *novi =
            napravi_cvor(ime, prezime, uspeh, matematika, jezik);
86          prover_i_alokaciju(novi);
88
            /* I proglašavamo ga korenom stabla */
            *koren = novi;
90
            return;
92      }
94
95      /* Inace, dodajemo cvor u stablo tako da bude sortiran po
        ukupnom broju poena */
96      if (uspeh + matematika + jezik >
        (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)

```

```

98     dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
100                     matematika, jezik);
102     else
104         dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
106                     matematika, jezik);
108 }
110
112 /* Funkcija ispisuje sadrzaj stabla - ukoliko je vrednost
114    argumenta polozili jednaka 0 ispisuju se informacije o
116    ucenicima koji nisu polozili prijemni, a ako je vrednost
118    argumenta razlicita od nule, ispisuju se informacije o
120    ucenicima koji su polozili prijemni */
122 void stampaj(Cvor * koren, int polozili)
124 {
126     /* Stablo je prazno - prekidamo sa ispisom */
128     if (koren == NULL)
130         return;
132
134     /* Stampamo informacije iz levog podstabla */
136     stampaj(koren->levo, polozili);
138
139     /* Stampamo informacije iz korenog cvora */
140     if (polozili && koren->matematika + koren->jezik >= 10)
142         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
144             koren->prezime, koren->uspeh, koren->matematika,
146             koren->jezik,
148             koren->uspeh + koren->matematika + koren->jezik);
149     else if (!polozili && koren->matematika + koren->jezik < 10)
150         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
152             koren->prezime, koren->uspeh, koren->matematika,
154             koren->jezik,
156             koren->uspeh + koren->matematika + koren->jezik);
158
159     /* Stampamo informacije iz desnog podstabla */
160     stampaj(koren->desno, polozili);
162 }
164
165 /* Funkcija koja odredjuje koliko studenata nije polozilo
166    prijemni ispit */
167 int nisu_polozili(Cvor * koren)
169 {
171     /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
172     if (koren == NULL)
174         return 0;
176
177     /* Pretragu vrsimo i u levom i u desnom podstablu - ako uslov
178        za polaganje nije ispunjen za koreni cvor, broj studenata
179        uvecavamo za 1 */
180     if (koren->matematika + koren->jezik < 10)
182         return 1 + nisu_polozili(koren->levo) +

```

```
150     nisu_polozili(koren->desno);
152     return nisu_polozili(koren->levo) +
        nisu_polozili(koren->desno);
154 }

156 int main(int argc, char **argv)
157 {
158     FILE *in;
159     Cvor *koren;
160     char ime[MAX], prezime[MAX];
161     double uspeh, matematika, jezik;
162
163     /* Otvaramo datoteku sa rezultatima sa prijemnog za citanje */
164     in = fopen("prijemni.txt", "r");
165     if (in == NULL) {
166         fprintf(stderr, "Greska prilikom citanja podataka!\n");
167         exit(EXIT_FAILURE);
168     }
169
170     /* Citamo podatke i dodajemo ih u stablo */
171     koren = NULL;
172     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
        &matematika, &jezik) != EOF) {
173         dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
        jezik);
174     }
175
176     /* Zatvaramo datoteku */
177     fclose(in);
178
179     /* Stampamo prvo podatke o uenicima koji su polozili prijemni
180     */
181     stampaj(koren, 1);
182
183     /* Liniju iscrtavamo samo ako postoje uenici koji nisu
184     polozili prijemni */
185     if (nisu_polozili(koren) != 0)
186         printf("-----\n");
187
188     /* Stampamo podatke o uenicima koji nisu polozili prijemni */
189     stampaj(koren, 0);
190
191     /* Oslobadjamo memoriju zauzetu stablom */
192     oslobodi_stablo(&koren);
193
194     /* Završavamo sa programom */
195     return 0;
196 }
197
```

## Rešenje 4.18

```

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

5  #define MAX_NISKA 51
   #define MAX_DATUM 3

7
   /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i
9   prezime osobe, dan, mesec i godinu rođenja i redom
   pokazivace na levo i desno podstablo */
11 typedef struct cvor_stabla {
   char ime[MAX_NISKA];
13  char prezime[MAX_NISKA];
   int dan;
15  int mesec;
   int godina;
17  struct cvor_stabla *levo;
   struct cvor_stabla *desno;
19 } Cvor;

21 /* Funkcija koja kreira novi cvor */
   Cvor *napravi_cvor(char ime[], char prezime[], int dan,
23                      int mesec, int godina)
   {
25     /* Alociramo memoriju */
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
       return NULL;

29     /* Inicijalizujemo polja strukture */
31     strcpy(novi->ime, ime);
   strcpy(novi->prezime, prezime);
33     novi->dan = dan;
   novi->mesec = mesec;
35     novi->godina = godina;
   novi->levo = NULL;
37     novi->desno = NULL;

39     /* Vracamo adresu novog cvora */
   return novi;
41 }

43 /* Funkcija koja proverava uspesnost alokacije */
   void proveri_alokaciju(Cvor * novi_cvor)
45 {
   /* Ako memorija nije uspesno alocirana */
47     if (novi_cvor == NULL) {
       /* Ispisujemo poruku i prekidamo izvršavanje programa */
49     fprintf(stderr, "Malloc greska za novi cvor!\n");
       exit(EXIT_FAILURE);
51 }

```



```

}
53
/* Funkcija koja oslobadja memoriju zauzetu stablom */
55 void oslobodi_stablo(Cvor ** koren)
{
57     /* Stablo je prazno */
    if (*koren == NULL)
59         return;

61     /* Oslobadjamo memoriju zauzetu levim podstablom (ako postoji)
        */
63     if ((*koren)->levo)
        oslobodi_stablo(&(*koren)->levo);

65     /* Oslobadjamo memoriju zauzetu desnim podstablom (ako
        postoji) */
67     if ((*koren)->desno)
69         oslobodi_stablo(&(*koren)->desno);

71     /* Oslobadjamo memoriju zauzetu korenom */
    free(*koren);

73     /* Proglasavamo stablo praznim */
75     *koren = NULL;
}

77
/* Funkcija koja dodaje novi cvor u stablo - stablo treba da
79     bude uredjeno po datumu */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
81                     int dan, int mesec, int godina)
{
83     /* Ako je stablo prazno */
    if (*koren == NULL) {
85
87         /* Kreiramo novi cvor */
        Cvor *novi_cvor =
            napravi_cvor(ime, prezime, dan, mesec, godina);
89         prover_i_alokaciju(novi_cvor);

91         /* I proglasavamo ga korenom */
        *koren = novi_cvor;

93         return;
95     }

97     /* Kako se ne unosi godina za pretragu, stablo uredjujemo samo
        po mesecu (i danu u okviru istog meseca) */
99     if (mesec < (*koren)->mesec)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
101                        godina);
    else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
103        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,

```

```

                                godina);
105  else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec,
107                                godina);
    }
109
    /* Funkcija vrši pretragu stabla i vraća cvor sa traženim
111       datumom (null ako takav ne postoji). u promenljivu pom ce
        biti smesten prvi datum (dan i mesec) veci od traženog datuma
113       (null ako takav ne postoji)

115  */
    Cvor *pretrazi(Cvor * koren, int dan, int mesec)
117  {
        /* Stablo je prazno, obustavljamo pretragu */
119         if (koren == NULL)
                return NULL;
121
        /* Nasli smo traženi datum u stablu */
123         if (koren->dan == dan && koren->mesec == mesec)
                return koren;
125
        /* Ako je mesec traženog datuma manji od meseca sadržanog u
127         korenu ili ako su meseci isti ali je dan traženog datuma
            manji od aktuelnog datuma, pretražujemo levo podstablo -
129         pre toga svakako proveravamo da li leva grana postoji - ako
            ne postoji treba da vratimo prvi sledeći, a to je bas
131         vrednost uocenog korena */
        if (mesec < koren->mesec
133             || (mesec == koren->mesec && dan < koren->dan)) {
            if (koren->levo == NULL)
135                 return koren;
            else
137                 return pretrazi(koren->levo, dan, mesec);
        }
139
        /* inace, nastavljamo pretragu u desnom delu */
141         return pretrazi(koren->desno, dan, mesec);
    }
143
145  int main(int argc, char **argv)
    {
147         FILE *in;
        Cvor *koren;
149         Cvor *slavljenik;
        char ime[MAX_NISKA], prezime[MAX_NISKA];
151         int dan, mesec, godina;

153         /* Proveravamo da li je zadato ime ulazne datoteke */
        if (argc < 2) {
155             /* Ako nije, ispisujemo poruku i prekidamo sa izvršavanjem
```

```
    programa */
157     printf("Nedostaje ime ulazne datoteke!\n");
        return 0;
159 }

/* Inace, pripremamo datoteku za citanje */
161 in = fopen(argv[1], "r");
163 if (in == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
165     exit(EXIT_FAILURE);
}

/* I popunjavamo podacima stablo */
167 koren = NULL;
169 while (fscanf
171     (in, "%s %s %d.%d.%d.", ime, prezime, &dan, &mesec,
        &godina) != EOF)
173     dodaj_u_stablo(&koren, ime, prezime, dan, mesec, godina);

/* I zatvaramo datoteku */
175 fclose(in);
177

/* Omogucavamo pretragu podataka */
179 while (1) {

    /* Ucitavamo novi datum */
    printf("Unesite datum: ");
183     if (scanf("%d.%d.", &dan, &mesec) == EOF)
        break;
185

    /* Pretrazujemo stablo */
187     slavljenik = pretrazi(koren, dan, mesec);

    /* Ispisujemo pronadjene podatke */
189     if (slavljenik == NULL) {
191         printf("Nema podataka o ovim ni o sledecem rodjendanu.\n");
        continue;
193     }

    /* Slucaj kada smo pronasli prave podatke */
195     if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
197         printf("Slavljenik: %s %s\n", slavljenik->ime,
            slavljenik->prezime);
199         continue;
    }
201

    /* Slucaj kada smo pronasli podatke o prvom sledecem
203         rodjendanu */
    printf("Slavljenik: %s %s %d.%d.\n", slavljenik->ime,
205         slavljenik->prezime, slavljenik->dan,
        slavljenik->mesec);
207 }
```

```
209  /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);
211
    /* Prekidamo sa izvršavanjem programa */
213  return 0;
}
```

### Rešenje 4.19

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1

4  /* Struktura kojom se predstavlja cvor binarnog pretraživackog
   stabla */
6  typedef struct cvor {
    int broj;
8   struct cvor *levo, *desno;
   } Cvor;

10
11  /* Funkcija koja alokira memoriju za novi cvor stabla,
12   inicijalizuje polja strukture i vraća pokazivac na novi cvor */
   Cvor *napravi_cvor(int broj);

14
15  /* Funkcija koja proverava uspesnost kreiranja novog cvora
16   stabla. */
   void proveri_alokaciju(Cvor * novi_cvor);

18
19  /* Funkcija koja dodaje zadati broj u stablo */
20  void dodaj_u_stablo(Cvor ** adresa_korena, int broj);

22
23  /* Funkcija koja proverava da li se zadati broj nalazi u stablu */
   Cvor *pretrazi_stablo(Cvor * koren, int broj);

24
25  /* Funkcija koja pronalazi cvor koji sadrži najmanju vrednost u
26   stablu */
   Cvor *pronadji_najmanji(Cvor * koren);

28
29  /* Funkcija koja pronalazi cvor koji sadrži najveću vrednost u
30   stablu */
   Cvor *pronadji_najveci(Cvor * koren);

32
33  /* Funkcija koja briše cvor stabla koji sadrži zadati broj */
34  void obrisi_element(Cvor ** adresa_korena, int broj);

36
37  /* Funkcija koja ispisuje sadržaj stabla u infiksnoj notaciji
   (levo podstablo - koren - desno podstablo) */
38  void prikazi_stablo(Cvor * koren);

40
41  /* Funkcija koja oslobađa memoriju zauzetu stablom */
   void oslobodi_stablo(Cvor ** adresa_korena);
```

```

42 #endif

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "stabla.h"

5  /* Funkcija kojom se kreira novi cvor stabla koji sadrzi zadatu
   vrednost */
7  Cvor *napravi_cvor(int broj)
   {
9      /* Alociramo memoriju za novi cvor */
      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
11     if (novi == NULL)
         return NULL;
13     /* Inicijalizujemo polja cvora */
      novi->broj = broj;
15     novi->levo = NULL;
      novi->desno = NULL;
17     /* Vracamo adresu novog cvora */
      return novi;
19 }

21 /* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
23 void prover_i_alokaciju(Cvor * novi_cvor)
   {
25     /* Ukoliko je cvor neuspesno kreiran */
      if (novi_cvor == NULL) {
27         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
           programa */
29         fprintf(stderr, "Malloc greska za novi cvor!\n");
           exit(EXIT_FAILURE);
31     }
   }

33
35 /* Funkcija koja dodaje novi broj u stablo. */
   void dodaj_u_stablo(Cvor ** koren, int broj)
   {
37     /* Ako je stablo prazno */
      if (*koren == NULL) {
39         /* Kreiramo novi cvor */
           Cvor *novi = napravi_cvor(broj);
           prover_i_alokaciju(novi);
           /* i proglašavamo ga korenom stabla */
43         *koren = novi;
           return;
45     }
      /* U suprotnom trazimo odgovarajucu poziciju za novi broj */
47     /* Ako je broj manji od vrednosti sadrzane u korenu, ubacujemo
       ga u levo podstablo */
49     if (broj < (*koren)->broj)

```

```
        dodaj_u_stablo(&(*koren)->levo, broj);
51     else
        /* Inace, ubacujemo broj u desno podstablo */
53     dodaj_u_stablo(&(*koren)->desno, broj);
}

55 /* Funkcija koja oslobadja memoriju zauzetu stablom */
57 void oslobodi_stablo(Cvor ** koren)
{
59     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*koren == NULL)
61         return;
    /* Inace ... */
63     /* Oslobadjamo memoriju zauzetu levom podstablom */
    if ((*koren)->levo)
65         oslobodi_stablo(&(*koren)->levo);
    /* Oslobadjamo memoriju zauzetu desnom podstablom */
67     if ((*koren)->desno)
        oslobodi_stablo(&(*koren)->desno);
69     /* Oslobadjamo memoriju zauzetu korenom */
    free(*koren);
71     /* Proglasavamo stablo praznim */
    *koren = NULL;
73 }

75 Cvor *pronadji_najmanji(Cvor * koren)
{
77     /* ako je stablo prazno, prekidamo pretragu */
    if (koren == NULL)
79         return NULL;
    /* vrednosti koje su manje od vrednosti u korenu stabla nalaze
81     se levo od njega */
    /* ako je koren cvor koji nema levo podstablo, onda on sadrzi
83     najmanju vrednost */
    if (koren->levo == NULL)
85         return koren;
    /* inace, pretragu treba nastaviti u levom podstablu */
87     return pronadji_najmanji(koren->levo);
}

89 Cvor *pronadji_najveci(Cvor * koren)
91 {
    /* ako je stablo prazno, prekidamo pretragu */
93     if (koren == NULL)
        return NULL;
95     /* vrednosti koje su vece od vrednosti u korenu stabla nalaze
        se desno od njega */
97     /* ako je koren cvor koji nema desno podstablo, onda on sadrzi
        najveću vrednost */
99     if (koren->desno == NULL)
        return koren;
101    /* inace, pretragu treba nastaviti u desnom podstablu */
}
```

```

    return pronadji_najveci(koren->desno);
103 }

105 /* Funkcija brise element iz stabla ciji je broj upravo jednak
    broju n. Funkcija azurira koren stabla u pozivajucoj
107 funkciji, jer u ovoj funkciji koren moze biti promenjen u
    funkciji. */
109 void obrisi_element(Cvor ** adresa_korena, int n)
{
111     Cvor *pomocni = NULL;
    /* Izlaz iz rekurzije: ako je stablo prazno, nema sta da se
113     brise */
    if (*adresa_korena == NULL)
115         return;
    /* Ako je vrednost broja veka od vrednosti u korenu stablua,
117     tada se broj eventualno nalazi u desnom podstablu, pa treba
    rekurzivno primeniti postupak na desno podstablo. Koren
119     ovako modifikovanog stabla je nepromenjen. */
    if ((*adresa_korena)->broj < n) {
121         obrisi_element(&(*adresa_korena)->desno, n);
        return;
123     }
    /* Ako je vrednost broja manja od vrednosti korena, tada se
125     broj eventualno nalazi u levom podstablu, pa treba
    rekurzivno primeniti postupak na levo podstablo. Koren
127     ovako modifikovanog stabla je nepromenjen. */
    if ((*adresa_korena)->broj > n) {
129         obrisi_element(&(*adresa_korena)->levo, n);
        return;
131     }
    /* Slede podslucajevi vezani za slucaj kada je vrednost u
133     korenu jednaka broju koji se brise (tj. slucaj kada treba
    obrisati koren) */
135     /* Ako koren nema sinova, tada se on prosto brise, i rezultat
    je prazno stablo (vracamo NULL) */
    if ((*adresa_korena)->levo == NULL
137        && (*adresa_korena)->desno == NULL) {
139         free(*adresa_korena);
        *adresa_korena = NULL;
141         return;
    }
143     /* Ako koren ima samo levog sina, tada se brisanje vrshi tako
    sto obrisemo koren, a novi koren postaje levo sin */
145     if ((*adresa_korena)->levo != NULL
        && (*adresa_korena)->desno == NULL) {
147         pomocni = (*adresa_korena)->levo;
        free(*adresa_korena);
149         *adresa_korena = pomocni;
        return;
151     }
    /* Ako koren ima samo desnog sina, tada se brisanje vrshi tako
153     sto obrisemo koren, a novi koren postaje desno sin */

```

```
155     if ((*adresa_korena)->desno != NULL
156         && (*adresa_korena)->levo == NULL) {
157         pomocni = (*adresa_korena)->desno;
158         free(*adresa_korena);
159         *adresa_korena = pomocni;
160         return;
161     }
162     /* Slučaj kada koren ima oba sina. Tada se brisanje vrši na
163        sledeći način: - najpre se potraži sledbenik korena (u
164        smislu poretka) u stablu. To je upravo po vrednosti
165        najmanji cvor u desnom podstablu. On se može pronaći npr.
166        funkcijom pronadji_najmanji(). - Nakon toga se u koren
167        smesti vrednost tog cvora, a u taj cvor se smesti vrednost
168        korena (tj. broj koji se briše). - Onda se prosto
169        rekurzivno pozove funkcija za brisanje na desno podstablo.
170        S obzirom da u njemu treba obrisati najmanji element, a on
171        definitivno ima najviše jednog potomka, jasno je da će
172        njegovo brisanje biti obavljeno na jedan od jednostavnijih
173        načina koji su gore opisani. */
174     pomocni = pronadji_najmanji((*adresa_korena)->desno);
175     (*adresa_korena)->broj = pomocni->broj;
176     pomocni->broj = n;
177     obrisi_element(&(*adresa_korena)->desno, n);
178 }
179
180 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje
181    elemente u rastućem poretku) */
182 void prikazi_stablo(Cvor * koren)
183 {
184     /* izlaz iz rekurzije */
185     if (koren == NULL)
186         return;
187     prikazi_stablo(koren->levo);
188     printf("%d ", koren->broj);
189     prikazi_stablo(koren->desno);
190 }
191
192 Cvor *pretrazi_stablo(Cvor * koren, int broj)
193 {
194     /* ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
195        */
196     if (koren == NULL)
197         return NULL;
198     /* ako je tražena vrednost sadržana u korenu */
199     if (koren->broj == broj) {
200         /* prekidamo pretragu */
201         return koren;
202     }
203     /* inace, ako je broj manji od vrednosti sadržane u korenu */
204     if (broj < koren->broj)
205         /* pretragu nastavljamo u levom podstablu */
206         return pretrazi_stablo(koren->levo, broj);
```



```

207     else
209         /* u suprotnom, pretragu nastavljamo u desnom podstablu */
        return pretrazi_stablo(koren->desno, broj);
    }

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključujemo biblioteku za rad sa stablima - pogledati uvodni
5     zadatak ove glave */
6  #include "stabla.h"
7
8
9  /* Funkcija koja proverava da li su dva stabla koja sadrže cele
10     brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
11     odnosno 0 ako nisu */
12  int identitet(Cvor * koren1, Cvor * koren2)
13  {
14      /* Ako su oba stabla prazna, jednaka su */
15      if (koren1 == NULL && koren2 == NULL)
16          return 1;
17
18      /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
19         jednaka */
20      if (koren1 == NULL || koren2 == NULL)
21          return 0;
22
23      /* Ako su oba stabla neprazna i u korenu se nalaze različite
24         vrednosti, možemo da zaključimo da se razlikuju */
25      if (koren1->broj != koren2->broj)
26          return 0;
27
28      /* inace, proveravamo da li vazi i jednakost u levih
29         podstabala i desnih podstabala */
30      return (identitet(koren1->levo, koren2->levo)
31              && identitet(koren1->desno, koren2->desno));
32  }
33
34  int main()
35  {
36      int broj;
37      Cvor *koren1, *koren2;
38
39      koren1 = NULL;
40      /* učitavamo elemente prvog stabla */
41      printf("Prvo stablo: ");
42      scanf("%d", &broj);
43      while (broj != 0) {
44          dodaj_u_stablo(&koren1, broj);
45          scanf("%d", &broj);
46      }
47  }

```

```
koren2 = NULL;
49 /* učitavamo elemente drugog stabla */
printf("Drugo stablo: ");
51 scanf("%d", &broj);
while (broj != 0) {
53     dodaj_u_stablo(&koren2, broj);
    scanf("%d", &broj);
55 }

57 /* pozivamo funkciju koja ispituje identitet stabala */
if (identitet(koren1, koren2))
59     printf("Stabla jesu identicna.\n");
else
61     printf("Stabla nisu identicna.\n");

63 /* oslobadjamo memoriju zauzetu stablima */
oslobodi_stablo(&koren1);
65 oslobodi_stablo(&koren2);

67 /* završavamo sa radom programa */
return 0;
69 }
```

### Rešenje 4.20

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

4 /* Uključujemo biblioteku za rad sa stablima */
#include "stabla.h"

6
/* Funkcija kreira novo stablo identicno stablu koje je dato
8 korenom. */
void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
10 {
    /* Izlaz iz rekurzije: ako je stablo prazno nema sta da se
12    kopira */
    if (koren == NULL) {
14        *duplikat = NULL;
        return;
16    }

18    /* Dupliramo koren stabla i postavljamo ga da bude koren novog
    stabla */
20    *duplikat = napravi_cvor(koren->broj);
    prover_i_alokaciju(*duplikat);

22
    /* Rekurzivno dupliramo levo podstablo i njegovu adresu cuvamo
24    u pokazivacu na levo podstablo korena duplikata. */
    kopiraj_stablo(koren->levo, &(*duplikat)->levo);
26 }
```

```

28  /* Rekurzivno dupliramo desno podstablo i njegovu adresu
    cuvamo u pokazivacu na desno podstablo korena duplikata. */
    kopiraj_stablo(koren->desno, &(*duplikat)->desno);
30 }

32 /* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
    dobija modifikacijom prvog stabla */
34 void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
    {
36     /* Ako drugo stablo nije prazno */
    if (koren2 != NULL) {
38         /* dodajemo njegov koren u prvo stablo */
        dodaj_u_stablo(adresa_korena1, koren2->broj);
40
        /* rekurzivno racunamo uniju levog i desnog podstabla drugog
        stabla sa prvim stablom */
42         kreiraj_uniju(adresa_korena1, koren2->levo);
44         kreiraj_uniju(adresa_korena1, koren2->desno);
    }
46 }

48 /* Funkcija izracunava presek dva stabla - rezultujuce stablo se
    dobija modifikacijom prvog stabla */
50 void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
    {
52     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
        */
    if (*adresa_korena1 == NULL)
54         return;

56     /* Kreiramo presek levog i desnog podstabla sa drugim stablom,
    tj. iz levog i desnog podstabla prvog stabla brisemo sve
    one elemente koji ne postoje u drugom stablu */
58     kreiraj_presek(&(*adresa_korena1)->levo, koren2);
60     kreiraj_presek(&(*adresa_korena1)->desno, koren2);

62     /* Ako se koren prvog stabla ne nalazi u drugom stablu tada ga
    uklanjamo iz prvog stabla */
64     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
66         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
    }

68
70 /* Funkcija izracunava razliku dva stabla - rezultujuce stablo
    se dobija modifikacijom prvog stabla */
    void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
72     {
74         /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
            */
        if (*adresa_korena1 == NULL)
76             return;

78         /* Kreiramo razliku levog i desnog podstabla sa drugim

```

```

80     stablom, tj. iz levog i desnog podstabla prvog stabla
      brisemo sve one elemente koji postoje i u drugom stablu */
82     kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
      kreiraj_razliku(&(*adresa_korena1)->desno, koren2);

84     /* Ako se koren prvog stabla nalazi i u drugom stablu tada ga
      uklanjamo iz prvog stabla */
86     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
        obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
88 }

90 int main()
91 {
92     Cvor *koren1;
93     Cvor *koren2;
94     Cvor *pomocni = NULL;
95     int n;
96
97     /* Ucitavamo elemente prvog stabla: */
98     koren1 = NULL;
99     printf("Prvo stablo: ");
100    while (scanf("%d", &n) != EOF) {
101        dodaj_u_stablo(&koren1, n);
102    }

104    /* Ucitavamo elemente drugog stabla: */
105    koren2 = NULL;
106    printf("Drugo stablo: ");
107    while (scanf("%d", &n) != EOF) {
108        dodaj_u_stablo(&koren2, n);
109    }

110    /* Kreiramo uniju stabala: prvo napravimo kopiju prvog stabla
111       kako bi mogli da ga iskoristimo i za preostale operacije */
112    kopiraj_stablo(koren1, &pomocni);
113    kreiraj_uniju(&pomocni, koren2);
114    printf("Unija: ");
115    prikazi_stablo(pomocni);
116    putchar('\n');

118    /* Oslobadjamo stablo za rezultatom operacije */
119    oslobodi_stablo(&pomocni);

120
121    /* Kreiramo presek stabala: prvo napravimo kopiju prvog stabla
122       kako bi mogli da ga iskoristimo i za preostale operacije; */
123    kopiraj_stablo(koren1, &pomocni);
124    kreiraj_presek(&pomocni, koren2);
125    printf("Presek: ");
126    prikazi_stablo(pomocni);
127    putchar('\n');

128
129    /* Oslobadjamo stablo za rezultatom operacije */
130

```

```

132     oslobodi_stablo(&pomocni);

134     /* Kreiramo razliku stabala: prvo napravimo kopiju prvog
        stabla kako bi mogli da ga iskoristimo i za preostale
        operacije; */
136     kopiraj_stablo(koren1, &pomocni);
        kreiraj_razliku(&pomocni, koren2);
138     printf("Razlika: ");
        prikazi_stablo(pomocni);
140     putchar('\n');

142     /* Oslobadjamo stablo za rezultatom operacije */
        oslobodi_stablo(&pomocni);
144
        /* Oslobadjamo i polazna stabla */
146     oslobodi_stablo(&koren2);
        oslobodi_stablo(&koren1);
148
        /* Završavamo sa programom */
150     return 0;
}

```

#### Rešenje 4.21

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
6
7  #define MAX 50
8
9  /* Funkcija koja obilazi stablo sa leva na desno i smesta
        vrednosti cvorova u niz. Povratna vrednost funkcije je broj
        vrednosti koje su smestene u niz. */
11 int kreiraj_niz(Cvor * koren, int a[])
12 {
13     int r, s;
14
15     /* Stablo je prazno - u niz je smesteno 0 elemenata */
17     if (koren == NULL)
        return 0;
19
        /* Dodajemo u niz elemente iz levog podstabla */
21     r = kreiraj_niz(koren->levo, a);
22
23     /* Tekuca vrednost promenljive r je broj elemenata koji su
        upisani u niz i na osnovu nje mozemo odrediti indeks novog
        elementa */
25
27     /* Smestamo vrednost iz korena */

```

```

29     a[r] = koren->broj;

31     /* Dodajemo elemente iz desnog podstabla */
    s = kreiraj_niz(koren->desno, a + r + 1);

33     /* Racunamo indeks na koji treba smestiti naredni element */
    return r + s + 1;
35 }

37 /* Funkcija sortira niz tako sto najpre elemente niza smesti u
39    stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva
    u desno.

41    Ovaj nacin sortiranja primer sortiranja koje nije "u mestu "
43    kao sto je to slucaj sa ostalim prethodno opisanim
    algoritmima sortiranja, jer se sortiranje vrsi u pomocnoj
    dinamičkoj strukturi, a ne razmenom elemenata niza. */
45 void sortiraj(int a[], int n)
{
47     int i;
    Cvor *koren;

49     /* Kreiramo stablo smestanjem elemenata iz niza u stablo */
51     koren = NULL;
    for (i = 0; i < n; i++)
53         dodaj_u_stablo(&koren, a[i]);

55     /* Infiksnim obilaskom stabla elemente iz stabla prepisujemo u
        niz a */
57     kreiraj_niz(koren, a);

59     /* Vise nam stablo nije potrebno i oslobadjamo memoriju */
    oslobodi_stablo(&koren);
61 }

63 int main()
{
65     int a[MAX];
    int n, i;

67     /* Ucitavamo dimenziju i elemente niza */
69     printf("n: ");
    scanf("%d", &n);
71     if (n < 0 || n > MAX) {
        printf("Greska: pogresna dimenzija niza!\n");
73         return 0;
    }

75     printf("a: ");
77     for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
79

```

```

81  /* Pozivamo funkciju za sortiranje */
    sortiraj(a, n);

83  /* Ispisujemo rezultat */
    for (i = 0; i < n; i++)
85      printf("%d ", a[i]);
    printf("\n");

87  /* Prekidamo sa programom */
89  return 0;
}

```

### Rešenje 4.22

```

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

3
   /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  /* a) Funkcija koja izracunava broj cvorova stabla */
   int broj_cvorova(Cvor * koren)
9  {
   /* Ako je stablo prazno, broj cvorova je nula */
11  if (koren == NULL)
    return 0;

13
   /* U suprotnom je broj cvorova stabla jednak zbiru broja
15  cvorova u levom podstablu i broja cvorova u desnom
    podstablu - 1 dodajemo zato sto treba racunati i koren */
17  return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) +
    1;

19 }

21 /* b) Funkcija koja izracunava broj listova stabla */
   int broj_listova(Cvor * koren)
23 {
   /* Ako je stablo prazno, broj listova je nula */
25  if (koren == NULL)
    return 0;

27
   /* Proveravamo da li je tekuci cvor list */
29  if (koren->levo == NULL && koren->desno == NULL)
    /* i ako jeste vracamo 1 - to ce kasnije zbog rekurzivnih
31     poziva uvecati broj listova za 1 */
    return 1;

33
   /* U suprotnom prebrojavamo listove koje se nalaze u
35     podstablima */
    return broj_listova(koren->levo) + broj_listova(koren->desno);

37 }

```

```
39 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
void pozitivni_listovi(Cvor * koren)
41 {
    /* Slucaj kada je stablo prazno */
43     if (koren == NULL)
        return;
45
    /* Ako je cvor list i sadrzi pozitivnu vrednost */
47     if (koren->levo == NULL && koren->desno == NULL
        && koren->broj > 0)
49         /* Stampamo ga */
        printf("%d ", koren->broj);
51
    /* Nastavljamo sa stampanjem pozitivnih listova u podstablama */
53     pozitivni_listovi(koren->levo);
    pozitivni_listovi(koren->desno);
55 }

57 /* d) Funkcija koja izracunava zbir cvorova stabla */
int zbir_cvorova(Cvor * koren)
59 {
    /* Ako je stablo prazno, zbir cvorova je 0 */
61     if (koren == NULL)
        return 0;
63
    /* Inace, zbir cvorova stabla izracunavamo kao zbir korena i
65     svih elemenata u podstablama */
    return koren->broj + zbir_cvorova(koren->levo) +
67         zbir_cvorova(koren->desno);
69 }

71 /* e) Funkcija koja izracunava najveći element stabla. */
Cvor *najveci_element(Cvor * koren)
{
73     /* Ako je stablo prazno, obustavljamo pretragu */
    if (koren == NULL)
75         return NULL;

77     /* Zbog prirode pretrazivackog stabla, sigurni smo da su
        vrednosti vece od korena u desnom podstablu */
79
    /* Ako desnog podstabla nema */
81     if (koren->desno == NULL)
        /* Najveća vrednost je koren */
83         return koren;

85     /* Inace, najveću vrednost trazimo jos desno */
    return najveci_element(koren->desno);
87 }

89 /* f) Funkcija koja izracunava dubinu stabla */
```



```
int dubina_stabla(Cvor * koren)
{
    /* Dubina praznog stabla je 0 */
    if (koren == NULL)
        return 0;

    /* Izracunavamo dubinu levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

    /* Izracunavamo dubinu desnog podstabla */
    int dubina_desno = dubina_stabla(koren->desno);

    /* dubina stabla odgovara vecoj od dubina podstabala - 1
       dodajemo jer racunamo i koren */
    return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* g) Funkcija koja izracunava broj cvorova na i-tom nivou */
int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
{
    /* ideja je da ste spustamo kroz stablo sve dok ne stignemo do
       trazenog nivoa */

    /* Ako nema vise cvorova, ne mozemo da se spustamo niz stablo */
    if (koren == NULL)
        return 0;

    /* Ako smo stigli do trazenog nivoa, vracamo 1 - to ce kasnije
       zbog rekurzivnih poziva uvecati broj pojavljivanja za 1 */
    if (i == 0)
        return 1;

    /* inace, spustamo se jedan nivo nize i u levom i u desnom
       postablu */
    return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
        + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
}

/* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispis_nivo(Cvor * koren, int i)
{
    /* ideja je slicna ideji iz prethodne funkcije */
    /* nema vise cvorova, ne mozemo da se spustamo kroz stablo */
    if (koren == NULL)
        return;

    /* ako smo na trazenom nivou - ispisujemo vrednost */
    if (i == 0) {
        printf("%d ", koren->broj);
        return;
    }
}
```

```
143  /* inace, spustamo se jedan nivo nize i u levom i u desnom
      podstablu */
145  ispis_nivo(koren->levo, i - 1);
      ispis_nivo(koren->desno, i - 1);
147  }

149  /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom
      nivou stabla */
      Cvor *max_nivo(Cvor * koren, int i)
151  {
153      /* Ako je stablo prazno, obustavljamo pretragu */
      if (koren == NULL)
          return NULL;

155      /* Ako smo na trazenom nivou, takodje prekidamo pretragu */
157      if (i == 0)
          return koren;

159      /* Pronalazimo maksimum sa i-tog nivoa levog podstabla */
161      Cvor *a = max_nivo(koren->levo, i - 1);

163      /* Pronalazimo maksimum sa i-tog nivoa desnog podstabla */
      Cvor *b = max_nivo(koren->desno, i - 1);

165      /* Trazimo i vracamo maksimum izracunatih vrednosti */
167      if (a == NULL && b == NULL)
          return NULL;
169      if (a == NULL)
          return b;
171      if (b == NULL)
          return a;
173      return a->broj > b->broj ? a : b;
175  }

177  /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
      int zbir_nivo(Cvor * koren, int i)
179  {
181      /* Ako je stablo prazno, zbir je nula */
      if (koren == NULL)
          return 0;

183      /* Ako smo na trazenom nivou, vracamo vrednost */
185      if (i == 0)
          return koren->broj;

187      /* Inace, spustamo se jedan nivo nize i trazimo sume iz levog
          i desnog podstabla */
189      return zbir_nivo(koren->levo, i - 1) + zbir_nivo(koren->desno,
                                                         i - 1);
191  }

193
```

```

195  /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje
      su manje ili jednake od date vrednosti x */
196  int suma(Cvor * koren, int x)
197  {
      /* Ako je stablo prazno, zbir je nula */
199      if (koren == NULL)
          return 0;

201      /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
203         prirode pretrazivackog stabla treba obici i levo i desno
          podstablo */
205      if (koren->broj < x)
          return koren->broj + suma(koren->levo,
207                                   x) + suma(koren->desno, x);

209      /* Inace, racunamo samo sumu vrednosti iz levog podstabla jer
          medju njima jedino moze biti onih koje zadovoljavaju uslov */
211      return suma(koren->levo, x);
212  }

213  int main(int argc, char **argv)
214  {
215      /* Analiziramo argumente komandne linije */
217      if (argc != 3) {
          fprintf(stderr,
219                 "Greska! Program se poziva sa: ./a.out nivo
          broj_za_pretragu\n");
          exit(EXIT_FAILURE);
221      }
      int i = atoi(argv[1]);
223      int x = atoi(argv[2]);

225      /* Kreiramo stablo */
      Cvor *koren = NULL;
227      int broj;
      while (scanf("%d", &broj) != EOF)
229          dodaj_u_stablo(&koren, broj);

231      /* ispisujemo rezultat rada funkcija */
      printf("broj cvorova: %d\n", br_cvorova(koren));
233      printf("broj listova: %d\n", br_listova(koren));
      printf("pozitivni listovi: ");
235      pozitivni_listovi(koren);
      printf("zbir cvorova: %d\n", suma_cvorova(koren));

237      if (najveci_element(koren) == NULL)
239          printf("najveci element: ne postoji\n");
      else
241          printf("najveci element: %d\n",
          najveci_element(koren)->broj);

243      printf("dubina stabla: %d\n", dubina_stabla(koren));

```

```
245     printf("\n");
246     printf("broj cvorova na %d. nivou: %d\n", i,
247           cvorovi_nivo(koren, i));
248     printf("elementi na %d. nivou: ", i);
249     ispis_nivo(koren, i);
250     printf("\n");
251     if (max_nivo(koren, i) == NULL)
252         printf("Nema elemenata na %d. nivou!\n", i);
253     else
254         printf("maksimalni na %d. nivou: %d\n", i,
255               max_nivo(koren, i)->broj);

257     printf("zbir na %d. nivou: %d\n", i, zbir_nivo(koren, i));
258     printf("zbir elemenata manjih ili jednakih od %d: %d\n", x,
259           suma(koren, x));

261     /* Oslobadjamo memoriju zauzetu stablom */
262     oslobodi_stablo(&koren);

263     /* Prekidamo izvršavanje programa */
264     return 0;
265 }
```

### Rešenje 4.23

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija koja izracunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
9  {
10     /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
12         return 0;
13
14     /* Izracunavamo dubinu levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
16
17     /* Izracunavamo dubinu desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);
19
20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1
21        dodajemo jer racunamo i koren */
22     return dubina_levo >
23           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }
25
26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
```

```
27 void ispisi_nivo(Cvor * koren, int i)
28 {
29     /* Ideja je slicna ideji iz prethodne funkcije */
30
31     /* Nema vise cvorova, ne mozemo da se spustamo kroz stablo */
32     if (koren == NULL)
33         return;
34
35     /* Ako smo na trazenom nivou - ispisujemo vrednost */
36     if (i == 0) {
37         printf("%d ", koren->broj);
38         return;
39     }
40     /* Inace, spustamo se jedan nivo nize i u levom i u desnom
41     podstablu */
42     ispisi_nivo(koren->levo, i - 1);
43     ispisi_nivo(koren->desno, i - 1);
44 }
45
46 /* Funkcija koja ispisuje stablo po nivoima */
47 void ispisi_stablo_po_nivoima(Cvor * koren)
48 {
49     int i;
50
51     /* Prvo izracunavamo dubinu stabla */
52     int dubina;
53     dubina = dubina_stabla(koren);
54
55     /* Ispisujemo nivo po nivo stabla */
56     for (i = 0; i < dubina; i++) {
57         printf("%d. nivo: ", i);
58         ispisi_nivo(koren, i);
59         printf("\n");
60     }
61 }
62
63 int main(int argc, char **argv)
64 {
65     Cvor *koren;
66     int broj;
67
68     /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
69     koren = NULL;
70     while (scanf("%d", &broj) != EOF) {
71         dodaj_u_stablo(&koren, broj);
72     }
73
74     /* Ispisujemo stablo po nivoima */
75     ispisi_stablo_po_nivoima(koren);
76
77     /* Oslobadjamo memoriju zauzetu stablom */
78     oslobodi_stablo(&koren);
79 }
```

```
79      /* Prekidamo izvršavanje programa */
81      return 0;
    }
```

### Rešenje 4.24

### Rešenje 4.25

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
7
   /* Funkcija koja izracunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
9  {
   /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
        return 0;
13
   /* Izracunavamo dubinu levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
17
   /* Izracunavamo dubinu desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);
19
   /* Dubina stabla odgovara vecoj od dubina podstabala - 1
   dodajemo jer racunamo i koren */
21     return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
23 }
25
   /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za
   AVL stablo */
27 int avl(Cvor * koren)
29 {
   int dubina_levo, dubina_desno;
31
   /* Ako je stablo prazno, zaustavljamo brojanje */
33     if (koren == NULL) {
        return 0;
35     }
37
   /* Izracunavamo dubinu levog podstabla korena */
38     dubina_levo = dubina_stabla(koren->levo);
39
   /* Izracunavamo dubinu desnog podstabla korena */
41     dubina_desno = dubina_stabla(koren->desno);
```

```

43  /* Ako je uslov za AVL stablo ispunjen */
44  if (abs(dubina_desno - dubina_levo) <= 1) {
45      /* Racunamo broj avl cvorova u levom i desnom podstablu i
46         uvecavamo za jedan iz razloga sto koren ispunjava uslov */
47      return 1 + avl(koren->levo) + avl(koren->desno);
48  } else {
49      /* Inace, racunamo samo broj avl cvorova u podstablama */
50      return avl(koren->levo) + avl(koren->desno);
51  }
52  }
53
54  int main(int argc, char **argv)
55  {
56      Cvor *koren;
57      int broj;
58
59      /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
60      koren = NULL;
61      while (scanf("%d", &broj) != EOF) {
62          dodaj_u_stablo(&koren, broj);
63      }
64
65      /* Racunamo i ispisujemo broj AVL cvorova */
66      printf("%d\n", avl(koren));
67
68      /* Oslobadjamo memoriju zauzetu stablom */
69      oslobodi_stablo(&koren);
70
71      /* Prekidamo izvršavanje programa */
72      return 0;
73  }

```

### Rešenje 4.26

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija proverava da li je zadato binarno stablo celih
8     pozitivnih brojeva heap. Ideja koju cemo implementirati u
9     osnovi ima pronalazenje maksimalne vrednosti levog i
10    maksimalne vrednosti desnog podstabla - ako je vrednost u
11    korenu veca od izracunatih vrednosti uoceni fragment stabla
12    zadovoljava uslov za heap. Zato ce funkcija vratiti
13    maksimalne vrednosti iz uocenog podstabala ili vrednost -1
14    ukoliko zakljucimo da stablo nije heap. */
15  int heap(Cvor * koren)
16  {

```

```
17  int max_levo, max_desno;

19  /* Prazno sablo je heap. */
   if (koren == NULL) {
21      /* posto je 0 najmanji pozitivan broj, moze nam poslužiti
        kao indikator */
23      return 0;

25  }
   /* Ukoliko je stablo list ... */
27   if (koren->levo == NULL && koren->desno == NULL) {
        /* ... vracamo njegovu vrednost */
29      return koren->broj;
   }

31
   /* Proveravamo svojstvo za levo podstablo. */
33   max_levo = heap(koren->levo);

35   /* Proveravamo svojstvo za desno podstablo. */
   max_desno = heap(koren->desno);
37   /* Ako levo ili desno podstablo uocenog cvora nije heap, onda
        nije ni celo stablo. */
39   if (max_levo == -1 || max_desno == -1) {
        return -1;
41   }

43   /* U suprotnom proveravamo da li svojstvo vazi za uoceni
        cvor. */
45   if (koren->broj > max_levo && koren->broj > max_desno) {
        /* ako vazi, vracamo vrednost korena */
47      return koren->broj;
   }

49
   /* u suprotnom zaključujemo da stablo nije heap */
51   return -1;
   }

53
int main(int argc, char **argv)
55 {
   Cvor *koren;
57   int heap_indikator;

59   /* Kreiramo stablo koje sadrži brojeve 100 19 36 17 3 25 1 2 7
        */
61   koren = NULL;
   koren = napravi_cvor(100);
63   koren->levo = napravi_cvor(19);
   koren->levo->levo = napravi_cvor(17);
65   koren->levo->levo->levo = napravi_cvor(2);
   koren->levo->levo->desno = napravi_cvor(7);
67   koren->levo->desno = napravi_cvor(3);
   koren->desno = napravi_cvor(36);
```



```
69 koren->desno->levo = napravi_cvor(25);
koren->desno->desno = napravi_cvor(1);
71
73 /* pozivamo funkciju kojom proveravamo da li je stablo heap */
heap_indikator = heap(koren);
75
77 /* i ispisujemo rezultat */
if (heap_indikator == -1) {
77     printf("Zadato tablo nije heap\n");
} else {
79     printf("Zadato stablo je heap!\n");
}
81
83 /* Oslobođamo memoriju zauzetu stablom. */
oslobodi_stablo(&koren);
85
87 /* Završavamo sa programom */
return 0;
}
```

## Rešenje 4.27



## 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.

*Test Test 1*

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
Izlaz:
Ispit
Matematika
Programiranje
```

*Test Test 2*

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
2
maksimalano
poena
Izlaz:
```

<i>Test Test 3</i>	<i>Test Test 4</i>
<pre>Poziv: ./a.out ulaz.txt Problem: datoteka ulaz.txt ne postoji Izlaz: -1</pre>	<pre>Poziv: ./a.out Izlaz: -1</pre>

[Rešenje 5.1]

### Zadatak 5.2

Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže 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`.

<i>Test Test 1</i>	<i>Test Test 2</i>
<pre>Ulaz: 2 8 10 3 6 14 13 7 4 0 Izlaz: 20</pre>	<pre>Ulaz: 0 50 14 5 2 4 56 8 52 7 1 0 Izlaz: 50</pre>

[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.

<p><i>Test Test 1</i></p> <pre> 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         </pre>	<p><i>Test Test 2</i></p> <pre> Ulaz:  2 3  0 0 -5  1 2 -4 Izlaz:  2         </pre>	<p><i>Test Test 3</i></p> <pre> Ulaz: -2 Izlaz (na stderr): -1         </pre>
---	---	---

[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.

<p><i>Test Test 1</i></p> <pre> Poziv: ./a.out ulaz.txt test ulaz.txt: Ovo je test primer.           U njemu se rec test javlja           vise puta. testtesttest Izlaz: 5         </pre>	<p><i>Test Test 2</i></p> <pre> Poziv: ./a.out Izlaz (na stderr): -1         </pre>
<p><i>Test Test 3</i></p> <pre> Poziv: ./a.out ulaz.txt foo ulaz.txt: (ne postoji) Izlaz (na stderr): -1         </pre>	<p><i>Test Test 4</i></p> <pre> Poziv: ./a.out ulaz.txt . ulaz.txt: (prazna) Izlaz: 0         </pre>

[Rešenje 5.4]

**Zadatak 5.5 Jelena:** Uključeno rešenje prethodnog zadatka. Dodati rešenje ovog zadatka. 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.

<p><i>Test Test 1</i></p> <pre> 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         </pre>	<p><i>Test Test 2</i></p> <pre> Datoteka: 3           1.2 3.2 1.1 4.3 Izlaz:    -1         </pre>
<p><i>Test Test 3</i></p> <pre> Datoteka: (nema datoteke) Izlaz:    -1         </pre>	<p><i>Test Test 4</i></p> <pre> Datoteka: 0 Izlaz:         </pre>

[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.

<p><i>Test Test 1</i></p> <pre> Ulaz: 1 5 3 6 1 4 7 9 Izlaz: 1         </pre>	<p><i>Test Test 2</i></p> <pre> Ulaz: 2 5 3 6 1 0 4 7 9 Izlaz: 2         </pre>	<p><i>Test Test 3</i></p> <pre> Ulaz: 0 4 2 5 Izlaz: 0         </pre>
<p><i>Test Test 4</i></p> <pre> Ulaz: 3 Izlaz: 0         </pre>	<p><i>Test Test 5</i></p> <pre> Ulaz: -1 4 5 1 7 Izlaz: 0         </pre>	

[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  {
8      printf("-1\n");
9      exit(EXIT_FAILURE);
10 }
11
12 int main(int argc, char *argv[])
13 {
14
15     FILE *ulaz;
16     char **linije;
17     int i, j, n;
18
19     /* Proveravamo argumente komandne linije. */
20     if (argc != 2) {
21         greska();
22     }
23
24     /* Otvaramo datoteku cije ime je navedeno kao argument
25        komandne linije neposredno nakon imena programa koji se
26        poziva. */
27     ulaz = fopen(argv[1], "r");
28     if (ulaz == NULL) {
29         greska();
30     }
31
32     /* Ucitavamo broj linija. */
33     fscanf(ulaz, "%d", &n);
34
35     /* Alociramo memoriju na osnovu ucitanog broja linija. */
36     linije = (char **) malloc(n * sizeof(char *));
37     if (linije == NULL) {
38         greska();
39     }
40     for (i = 0; i < n; i++) {
41         linije[i] = malloc(MAX * sizeof(char));
42         if (linije[i] == NULL) {
43             for (j = 0; j < i; j++) {
44                 free(linije[j]);
45             }
46             free(linije);
47         }
48     }
```

```
47     greska();
48     }
49 }

51 /* Ucitavamo svih n linija iz datoteke. */
52 for (i = 0; i < n; i++) {
53     fscanf(ulaz, "%s", linije[i]);
54 }

55 /* Ispisujemo u odgovarajucem poretку ucitane linije koje
56    zadovoljavaju kriterijum. */
57 for (i = n - 1; i >= 0; i--) {
58     if (isupper(linije[i][0])) {
59         printf("%s\n", linije[i]);
60     }
61 }

62 }

63 /* Oslobadjamo memoriju koju smo dinamicki alocirali. */
64 for (i = 0; i < n; i++) {
65     free(linije[i]);
66 }

67 }

68 free(linije);

69

70 /* Zatvaramo datoteku. */
71 fclose(ulaz);

72

73 /* Zavrsavamo sa programom. */
74 return 0;
75
76
77 }
```

### 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){
11         return koren->broj;
12     }

13
14     return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
15 }
16
```



```

18 int main(){
    Cvor* koren=NULL;
20     int n;
    int nivo;

22     /* Citamo vrednost nivoa */
24     scanf("%d", &nivo);

26

    while(1){

28         /* Citamo broj sa standardnog ulaza */
30         scanf("%d", &n);

32         /* Ukoliko je korisnik uneo 0, prekidamo dalje citanje. */
        if(n==0){
34             break;
        }

36         /* A ako nije, dodajemo procitani broj u stablo. */
38         dodaj_u_stablo(&koren, n);

40     }

42     /* Ispisujemo rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren,nivo));

44     /* Oslobadjamo memoriju */
46     oslobodi_stablo(&koren);

48

    /* Prekidamo izvršavanje programa */
50     return 0;
}

```

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "stabla.h"

5  Cvor* napravi_cvor(int b ) {
    Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
7    if( novi == NULL)
        return NULL;

9

    /* Inicijalizacija polja novog Cvora */
11    novi->broj = b;
    novi->levo = NULL;
13    novi->desno = NULL;

15    return novi;
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
34 void prover_i_alokaciju( Cvor* novi) {
35     if( novi == NULL) {
36         fprintf(stderr, "Malloc greska za nov cvor!\n");
37         exit(EXIT_FAILURE);
38     }
39 }
40
41 void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
42     /* Postojece stablo je prazno*/
43     if( *adresa_korena == NULL){
44         Cvor* novi = napravi_cvor(broj);
45         prover_i_alokaciju(novi);
46         *adresa_korena = novi; /* Kreirani Cvor novi ce biti od
47             sada koren stabla*/
48         return;
49     }
50
51     /* Brojeve smestamo u uredjeno binarno stablo, pa
52     ako je broj koji ubacujemo manji od broja koji je u korenu */
53     if( broj < (*adresa_korena)->broj)
54         /* Dodajemo u levo podstablo */
55         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
56     /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
57     dodajemo nov Cvor desno od korena */
58     else
59         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
60 }

```

```

1 #ifndef __STABLA_H__
2 #define __STABLA_H__ 1
3
4 /* Struktura kojom se predstavlja Cvor stabla */
5 typedef struct dcvor{
6     int broj;
7     struct dcvor* levo, *desno;
8 }

```

```

8 } Cvor;

10 /* Funkcija alocira prostor za novi Cvor stabla, inicijalizuje polja
   strukture i vraća pokazivac na nov Cvor */
12 Cvor* napravi_cvor(int b );

14 /* Oslobadjamo dinamički alociran 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 alociran,
   * 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;
    int i, j;
10    int max_broj_negativnih, max_indeks_kolone;
    int broj_negativnih;

12    /* Ucitavamo dimenzije matrice */
14    scanf("%d", &v);
    if (v < 0 || v > MAX) {
16        fprintf(stderr, "-1\n");
        return 0;
18    }

20    scanf("%d", &k);
    if (k < 0 || k > MAX) {
22        fprintf(stderr, "-1\n");
        return 0;
24    }

```

```
26  /* Ucitavamo elemente matrice */
    for (i = 0; i < v; i++) {
28      for (j = 0; j < k; j++) {
          scanf("%d", &m[i][j]);
30      }
    }

32
    /* Pronalazimo kolonu koja sadrzi najveći broj negativnih
34     elemenata */
    max_indeks_kolone = 0;

36
    max_broj_negativnih = 0;
38    for (i = 0; i < v; i++) {
        if (m[i][0] < 0) {
40            max_broj_negativnih++;
        }
42    }

44
    for (j = 0; j < k; j++) {
46        broj_negativnih = 0;
        for (i = 0; i < v; i++) {
48            if (m[i][j] < 0) {
                broj_negativnih++;
50            }
            if (broj_negativnih > max_broj_negativnih) {
52                max_indeks_kolone = j;
            }
54        }

56    }

58    /* Ispisujemo traženi rezultat */
    printf("%d\n", max_indeks_kolone);

60
    /* Završavamo program */
62    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;
9    int brojac = 0;
    char linija[MAX], *p;
```

```

11  if (argc != 3) {
13      fprintf(stderr, "-1\n");
14      exit(EXIT_FAILURE);
15  }

17  if ((f = fopen(argv[1], "r")) == NULL) {
18      fprintf(stderr, "-1\n");
19      exit(EXIT_FAILURE);
20  }

21  while (fgets(linija, MAX, f) != NULL) {
22      p = linija;
23      while (1) {
24          p = strstr(p, argv[2]);
25          if (p == NULL)
26              break;
27          brojac++;
28          p = p + strlen(argv[2]);
29      }
30  }

31  fclose(f);

32  printf("%d\n", brojac);

33  return 0;
34  }

```

### Rešenje 5.5

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  typedef struct _trougao {
6      double xa, ya, xb, yb, xc, yc;
7  } trougao;
8
9  double duzina(double x1, double y1, double x2, double y2) {
10     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
11 }
12
13 double povrsina(trougao t) {
14     double a = duzina(t.xb, t.yb, t.xc, t.yc);
15     double b = duzina(t.xa, t.ya, t.xc, t.yc);
16     double c = duzina(t.xa, t.ya, t.xb, t.yb);
17     double s = (a + b + c) / 2;
18     return sqrt(s * (s - a) * (s - b) * (s - c));
19 }
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
   free(niz);
72   fclose(f);
}
```

```
74     return 0;
75 }

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"

5  Cvor *napravi_cvor(int broj)
6  {
7
8      /* Dinamicki kreiramo cvor */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

11     /* U slucaju greske ... */
12     if (novi == NULL) {
13         fprintf(stderr, "-1\n");
14         exit(1);
15     }

17     /* Inicijalizacija */
18     novi->vrednost = broj;
19     novi->levi = NULL;
20     novi->desni = NULL;

21     /* Vracamo adresu novog cvora */
22     return novi;
23 }

25 void dodaj_u_stablo(Cvor **koren, int broj)
26 {
27
28     /* Izlaz iz rekurzije: ako je stablo bilo prazno,
29        novi koren je upravo novi cvor */
30     if (*koren == NULL) {
31         *koren = napravi_cvor(broj);
32         return;
33     }

34     /* Ako je stablo neprazno, i koren sadrzi manju vrednost
35        od datog broja, broj se umece u desno podstablo,
36        rekurzivnim pozivom */
37     if ((*koren)->vrednost < broj)
38         dodaj_u_stablo(&(*koren)->desni, broj);
39     /* Ako je stablo neprazno, i koren sadrzi vecu vrednost
40        od datog broja, broj se umece u levo podstablo,
41        rekurzivnim pozivom */
42     else if ((*koren)->vrednost > broj)
43         dodaj_u_stablo(&(*koren)->levi, broj);
44
45 }
46 }
```

```

49 void prikazi_stablo(Cvor * koren)
50 {
51     /* Izlaz iz rekurzije */
52     if (koren == NULL)
53         return;
54
55     prikazi_stablo(koren->levi);
56     printf("%d ", koren->vrednost);
57     prikazi_stablo(koren->desni);
58 }
59
60 Cvor* ucitaj_stablo() {
61     Cvor *koren = NULL;
62     int x;
63     while (scanf("%d", &x) == 1)
64         dodaj_u_stablo(&koren, x);
65     return koren;
66 }
67
68 void oslobodi_stablo(Cvor **koren)
69 {
70
71     /* Izlaz iz rekurzije */
72     if (*koren == NULL)
73         return;
74
75     oslobodi_stablo(&(*koren)->levi);
76     oslobodi_stablo(&(*koren)->desni);
77     free(*koren);
78
79     *koren = NULL;
80 }

```

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura koja predstavlja cvor stabla */
5  typedef struct cvor {
6      int vrednost; /* Vrednost koja se cuva */
7      struct cvor *levi; /* Pokazivac na levo podstablo */
8      struct cvor *desni; /* Pokazivac na desno podstablo */
9  } Cvor;
10
11 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira
12  dinamicki, funkcijom malloc(). U slucaju greske program
13  se prekida i ispisuje se poruka o gresci. U slucaju
14  uspeha inicijalizuje se vrednost datim brojem, a pokazivaci
15  na podstabla se inicijalizuju na NULL. Funkcija vraća
16  adresu novokreiranog cvora */
17 Cvor *napravi_cvor(int broj);
18
19 /* Funkcija dodaje novi cvor u stablo sa datim korenom.

```



```

21     Ukoliko broj vec postoji u stablu, ne radi nista.
    Cvor se kreira funkcijom napravi_cvor(). */
void dodaj_u_stablo(Cvor **koren, int broj);
23
/* Funkcija prikazuje stablo s leva u desno (tj.
25 prikazuje elemente u rastucem poretku) */
void prikazi_stablo(Cvor * koren);
27
/* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
    vraca
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>
#include "stabla.h"
2
4 int f3(Cvor * koren, int n)
{
6     if (koren == NULL || n < 0)
        return 0;
8     if (n == 0) {
        if (koren->levi == NULL && koren->desni != NULL)
10         return 1;
        if (koren->levi != NULL && koren->desni == NULL)
12         return 1;
        return 0;
14     }
    return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
16 }

18 int main()
{
20     Cvor *koren;
    int n;
22
    scanf("%d", &n);
24     koren = ucitaj_stablo();

26     printf("%d\n", f3(koren, n));

28     oslobodi_stablo(&koren);

30     return 0;

```

## ***5 Ispitni rokovi***

---

}
---