

Univerzitet u Beogradu  
Matematički fakultet

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

# **PROGRAMIRANJE 2**

## **Zbirka zadataka sa rešenjima**

**Beograd  
2015.**

Autori:

*dr Milena Vujošević Janičić*, docent na Matematičkom fakultetu u Beogradu

*dr Jelena Graovac*, docent na Matematičkom fakultetu u Beogradu

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Anđelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu

Studentski trg 16, 11000 Beograd

Za izdavača: *prof. dr Zoran Rakić*, dekan

Recenzenti:

*dr Gordana Pavlović-Lažetić*, redovni profesor na Matematičkom fakultetu u Beogradu

*dr Dragan Urošević*, naučni savetnik na Matematičkom institutu SANU

Obrada teksta, crteži i korice: *autori*

Štampa:

Tiraž:

CIP Каталогизација у публикацији

Народна библиотека Србије, Београд

©2015. Milena Vujošević Janičić, Jelena Graovac, Ana Spasić, Mirko Spasić, Anđelka Zečević, Nina Radojičić

Ovo delo zaštićeno je licencom Creative Commons CC BY-NC-ND 4.0 (Attribution-NonCommercial-NoDerivatives 4.0 International License). Detalji licence mogu se videti na veb-adresi <http://creativecommons.org/licenses/by-nc-nd/4.0/>. Dozvoljeno je umnožavanje, distribucija i javno saopštavanje dela, pod uslovom da se navedu imena autora. Upotreba dela u komercijalne svrhe nije dozvoljena. Prerada, preoblikovanje i upotreba dela u sklopu nekog drugog nije dozvoljena.



# 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.

...

*Autori*

# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>2</b>
1.1	Podela koda po datotekama . . . . .	2
1.2	Algoritmi za rad sa bitovima . . . . .	6
1.3	Rekurzija . . . . .	11
1.4	Rešenja . . . . .	20
<b>2</b>	<b>Pokazivači</b>	<b>67</b>
2.1	Pokazivačka aritmetika . . . . .	67
2.2	Višedimenzioni nizovi . . . . .	71
2.3	Dinamička alokacija memorije . . . . .	75
2.4	Pokazivači na funkcije . . . . .	82
2.5	Rešenja . . . . .	84
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>125</b>
3.1	Algoritmi pretrage . . . . .	125
3.2	Algoritmi sortiranja . . . . .	130
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	140
3.4	Rešenja . . . . .	145
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>215</b>
4.1	Liste . . . . .	215
4.2	Stabla . . . . .	226
4.3	Rešenja . . . . .	235
<b>5</b>	<b>Ispitni rokovi</b>	<b>330</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	330
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	332
5.3	Programiranje 2, praktični deo ispita, septembar 2015. . . . .	334
5.4	Rešenja . . . . .	336

# 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 opisuje kompleksan broj njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `void ucitaj_kompleksan_broj(KompleksanBroj * z)` koja učitava kompleksan broj sa standardnog ulaza.
- (c) Napisati funkciju `void ispisi_kompleksan_broj(KompleksanBroj z)` 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 `float realan_deo(KompleksanBroj z)` koja vraća vrednost realnog dela broja.
- (e) Napisati funkciju `float imaginaran_deo(KompleksanBroj z)` koja vraća vrednost imaginarnog dela broja.
- (f) Napisati funkciju `float moduo(KompleksanBroj z)` koja računa moduo kompleksnog broja.
- (g) Napisati funkciju `KompleksanBroj konjugovan(KompleksanBroj z)` koja računa konjugovano-kompleksni broj svog argumenta  $z$ .
- (h) Napisati funkciju `KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)` koja sabira dva kompleksna broja  $z1$  i  $z2$ .

- (i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja oduzima dva kompleksna broja `z1` i `z2`.
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja množi dva kompleksna broja `z1` i `z2`.
- (k) Napisati funkciju `float argument(KompleksanBroj z)` koja računa argument kompleksnog broja `z`.

Napisati program koji testira prethodno napisane funkcije. Program najpre za kompleksan broj `z1` koji se unosi sa standardnog ulaza ispisuje njegov realni deo, imaginarni deo i moduo. Zatim za naredni kompleksan broj `z2` koji se unosi sa standardnog ulaza ispisuje njegov konjugovano-kompleksan broj i argument. Na kraju program ispisuje zbir, razliku i proizvod brojeva `z1` i `z2`.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite realan i imaginaran deo kompleksnog broja: 1 -3
(1.00 - 3.00 i)
Unesite realan i imaginaran deo kompleksnog broja: -1 4
(-1.00 + 4.00 i)
Unesite znak: -
(1.00 - 3.00 i) + (-1.00 + 4.00 i) = (2.00 - 7.00 i)
realan_deo: 2
imaginaran_deo: -7.000000
moduo 7.280110
Njegov konjugovano kompleksan broj: (2.00 + 7.00 i)
Argument kompleksnog broja: - 1.292497
```

[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.

### Primer 1

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

[Rešenje 1.2]

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

- (a) Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20. UPUTSTVO: *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 `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz.
- (c) Napisati funkciju `Polinom ucitaj()` koja učitava polinom sa standardnog ulaza. Za polinom se najpre unosi stepen pa njegovi koeficijenti.
- (d) Napisati funkciju `double izracunaj(const Polinom * p, double x)` za izračunavanje vrednosti polinoma `p` u datoj tački `x` koristeći Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja sabira dva polinoma `p` i `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja množi dva polinoma `p` i `q`.
- (g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja računa izvod polinoma `p`.
- (h) Napisati funkciju `Polinom nIzvod(const Polinom * p, int n)` koja računa `n`-ti izvod polinoma `p`.

Napisati program koji testira prethodno napisane funkcije. Najpre se polinomi `p` i `q` unose sa standardnog ulaza i ispisuju na standardni izlaz u odgovarajućem obliku. Zatim se računa i i ispisuje se zbir i proizvod polinoma `p` i `q`. Označimo izračunati proizvod sa `r`. Nakon toga program računa i ispisuje vrednost polinoma `r` (zaokruženu na dve decimale) u tački koju unosi korisnik. Na kraju se sa standardnog ulaza unosi broj `n`, i ispisuje `n`-ti izvod polinoma `r`.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite polinom p (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
3 1.2 3.5 2.1 4.2
Unesite polinom q (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
2 2.1 0 -3.9
Zbir polinoma je: 1.20x^3+5.60x^2+2.10x+0.30
Proizvod polinoma je polinom r:
2.52x^5+7.35x^4-0.27x^3-4.83x^2-8.19x-16.38
Unesite tacku u kojoj racunate vrednost polinoma r
0
Vrednost polinoma u tacki je -16.38
Unesite izvod polinoma koji zelite:
3
3. izvod polinoma r je: 151.20x^2+176.40x-1.62
```

[Rešenje 1.3]

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

- (a) Definirati strukturu `Razlomak` za reprezentovanje razlomaka.
- (b) Napisati funkciju `Razlomak ucitaj()` za učitavanje razlomaka.
- (c) Napisati funkciju `void ispisi(const Razlomak * r)` koja ispisuje razlomak.
- (d) Napisati funkciju `int brojilac(const Razlomak * r)` koje vraćaju brojilac razlomka `r`.
- (e) Napisati funkciju `int imenilac(const Razlomak * r)` koje vraćaju imenilac razlomka `r`.
- (f) Napisati funkciju `double realna_vrednost(const Razlomak * r)` koja vraća odgovarajuću realnu vrednost razlomka `r`.
- (g) Napisati funkciju `double recipročna_vrednost(const Razlomak * r)` koja izračunava recipročnu vrednost razlomka `r`.
- (h) Napisati funkciju `Razlomak skрати(const Razlomak * r)` koja skraćuje dati razlomak `r`.
- (i) Napisati funkciju `Razlomak saberi(const Razlomak * r1, const Razlomak * r2)` koja sabira dva razlomka `r1` i `r2`.
- (j) Napisati funkciju `Razlomak oduzmi(const Razlomak * r1, const Razlomak * r2)` koja oduzima dva razlomka `r1` i `r2`.
- (k) Napisati funkciju `Razlomak pomnozi(const Razlomak * r1, const Razlomak * r2)` koja množi dva razlomka `r1` i `r2`.
- (l) Napisati funkciju `Razlomak podeli(const Razlomak * r1, const Razlomak * r2)` koja deli dva razlomka `r1` i `r2`.

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`.



### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite imenilac i brojilac prvog razlomka: 1 2
Unesite imenilac i brojilac drugog razlomka: 3 2
1/2 + 3/2 = 2
1/2 - 3/2 = -1
1/2 * 3/2 = 3/4
1/2 / 3/2 = 1/3
```

## 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 2

[illegible]

### Test 3

```
ULAZ:
    0x00FF00FF
IZLAZ:
    00000000111111110000000011111111
```

Test 4

```
ULAZ:
    0xABCD E123
IZLAZ:
    10101011110011011110000100100011
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkcije `count_bits1` i `count_bits2` koje broje bitove sa vrednošću 1 u binarnom zapisu celog broja  $x$ . Prebrojavanje bitova ostvariti na dva načina:

- formiranjem odgovarajuće maske i njenim pomeranjem
- formiranjem odgovarajuće maske i pomeranjem promenljive  $x$ .

Napisati program koji za broj koji unosi u heksadekasnom formatu sa standardnog ulaza računa broj bitova sa vrednošću 1 korišćenjem funkcije `count_bits1` ili funkcije `count_bits2`. Od korisnika sa standardnog ulaza tražiti da izabere koju od ove funkcije treba koristiti u zavisnosti da li unese 1 ili 2 .

### Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite broj: 0x7F
|| Unesite redni broj funkcije: 1
|| Broj jedinica u zapisu je
|| funkcija count_bits1: 7
```

### Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite broj: 0x80
|| Unesite redni broj funkcije: 2
|| Broj jedinica u zapisu je
|| funkcija count_bits2: 1
```

### Primer 3

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite broj: 0x00FF00FF
|| Unesite redni broj funkcije: 2
|| Broj jedinica u zapisu je
|| funkcija count_bits2: 16
```

### Primer 4

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite broj: 0xABCD123
|| Unesite redni broj funkcije: 1
|| Broj jedinica u zapisu je
|| funkcija count_bits1: 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 zadaju u heksadekaskom formatu sa standardnog ulaza.

### Test 1

```
|| ULAZ:
|| 0x7F
|| IZLAZ:
|| Najveci:
|| 11111110000000000000000000000000
|| Najmanji:
|| 00000000000000000000000001111111
```

### Test 2

```
|| ULAZ:
|| 0x80
|| IZLAZ:
|| Najveci:
|| 10000000000000000000000000000000
|| Najmanji:
|| 00000000000000000000000000000001
```

### Test 3

```
|| ULAZ:
|| 0x00FF00FF
|| IZLAZ:
|| Najveci:
|| 11111111111111110000000000000000
|| Najmanji:
|| 00000000000000000111111111111111
```

### Test 4

```
|| ULAZ:
|| 0xFFFFFFFF
|| IZLAZ:
|| Najveci:
|| 11111111111111111111111111111111
|| Najmanji:
|| 11111111111111111111111111111111
```

[Rešenje 1.7]

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



- (b) Napisati funkciju `rotate_right` koja određuje broj koji se dobija rotiranjem `k` puta udesno datog celog neoznačenog broja `x`.
- (c) Napisati funkciju `rotate_right_signed` koja određuje broj koji se dobija rotiranjem `k` puta udesno datog celog broja `x`.

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

### Primer 1

```

INTERAKCIJA SA PROGRAMOM:
Unesite neoznaceni broj x: B10011A7
Unesite neoznaceni broj k: 5
x = 101100010000000000001000110100111
rotate_left(2969571751, 5) = 00100000000000100011010011110110
rotate_right(2969571751, 5) = 00111101100010000000000010001101
rotate_right_signed(2969571751, 5) = 00111101100010000000000010001101
    
```

[Rešenje 1.9]

**Zadatak 1.10** Napisati funkciju `mirror` koja određuje ceo broj čiji je binarni zapis slika u ogledalu binarnog zapisa argumenta funkcije. Napisati i program koji testira datu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu, tako što najpre ispisuje binarnu reprezentaciju unetog broja, a potom i binarnu reprezentaciju broja dobijenog nakon poziva funkcije `mirror` za uneti broj.

### Test 1

```

ULAZ:
255
IZLAZ:
0000000000000000000000001001010101
10101010010000000000000000000000
    
```

### Test 2

```

ULAZ:
-15
IZLAZ:
111111111111111111111111111101011
11010111111111111111111111111111
    
```

[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 1

```

ULAZ:
10
IZLAZ:
0
    
```

### Test 2

```

ULAZ:
2147377146
IZLAZ:
1
    
```

### Test 3

```

ULAZ:
1111111115
IZLAZ:
0
    
```

[Rešenje 1.11]

**Zadatak 1.12** Napisati funkciju koja broji koliko se puta dve uzastopne jedinice pojavljuju u binarnom zapisu celog neoznačenog broja  $x$ . Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Tri uzastopne jedinice sadrže dve uzastopne jedinice dva puta.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 11 IZLAZ: 1           </pre>	<pre> ULAZ: 1024 IZLAZ: 0           </pre>	<pre> ULAZ: 2147377146 IZLAZ: 22           </pre>

[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  $i$  i  $j$ . Pozicije  $i$  i  $j$  se učitavaju kao parametri komandne linije. Smatrati da je krajnji desni bit binarne reprezentacije 0-ti bit. Pri rešavanju nije dozvoljeno koristiti ni pomoćni niz ni aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ .

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 2</i>
<pre> POZIV: ./a.out 1 2 INTERAKCIJA SA PROGRAMOM: 11 13           </pre>	<pre> POZIV: ./a.out 1 2 INTERAKCIJA SA PROGRAMOM: 1024 1024           </pre>	<pre> POZIV: ./a.out 12 12 INTERAKCIJA SA PROGRAMOM: 12345 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 1</i>	<i>Test 2</i>	<i>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 data dva neoznačena broja  $x$  i  $y$  invertuje u podatku  $x$  one bitove koji se poklapaju sa odgovarajućim bitovima u broju  $y$ . Ostali bitovi ostaju nepromenjeni. Napisati program koji tu funkciju testira za brojeve koji se zadaju sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 123 10	ULAZ: 3251 0	ULAZ: 12541 1024
IZLAZ: 4294967285	IZLAZ: 4294967295	IZLAZ: 4294966271

**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. NAPOMENA: *Zadatak rešiti isključivo korišćenjem bitskih operatora.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 123	ULAZ: 3245	ULAZ: 100328
IZLAZ: 0	IZLAZ: 2	IZLAZ: 1

## 1.3 Rekurzija

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

- (a) tako da rešenje bude linearno,
- (b) tako da rešenje bude logaritamsko.

Napisati program koji testira napisane funkciju tako što se sa standardnog ulaza najpre unosi redni broj funkcije koja se primenjuje, a zatim i ceo broj  $x$  i prirodan broj  $k$ , a potom se na standardni izlaz ispisuje rezultat primene odgovarajuće funkcije na unete brojeve.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije (1/2):
1
Unesite broj x:      2
Unesite broj k:     10
1024

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije (1/2):
2
Unesite broj x:      9
Unesite broj k:      4
6561

```

[Rešenje 1.17]

**Zadatak 1.18** Koristeći uzajamnu (posrednu) rekurziju napisati:

- 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 za heksadekadni broj koji se unosi sa standardnog ulaza ispisuje da li je broj njegovih cifara paran ili neparan.

*Test 1*

```

ULAZ:
11
IZLAZ:
Uneti broj ima paran broj cifara.

```

*Test 2*

```

ULAZ:
123
IZLAZ:
Uneti broj ima neparan broj cifara.

```

[Rešenje 1.18]

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

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite n (<= 12): 5
5! = 120

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite n (<= 12): 0
0! = 1

```

[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 funkciju koja računa  $n$ -ti element u nizu  $F$

- (a) iterativno,
- (b) tako da funkcija bude rekurzivna i da koristi navedene rekurentne relacije,
- (c) tako da funkcija bude rekurzivna ali da se problemi manje dimenzije rešavaju samo jedan put.

Napisati program koji testira napisane funkciju tako što se sa standardnog ulaza najpre unosi redni broj funkcije koja se primenjuje, a zatim i vrednosti koeficijenata  $a$  i  $b$  i prirodan broj  $n$ . Na standardni izlaz se ispisuje rezultat primene odabrane funkcije na unete vrednosti koeficijenata  $a$  i  $b$  i prirodan broj  $n$ .

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije koju zelite:
1 - iterativna
2 - rekurzivna
3 - rekurzivna napredna
1
Unesite koeficijente: 2 3
Unesite koji clan niza se racuna: 5
F(5) = 61
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije koju zelite:
1 - iterativna
2 - rekurzivna
3 - rekurzivna napredna
3
Unesite koeficijente: 4 2
Unesite koji clan niza se racuna: 8
F(8) = 31360
```

[Rešenje 1.20]

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

#### Test 1

```
ULAZ:
123
IZLAZ:
6
```

#### Test 2

```
ULAZ:
23156
IZLAZ:
17
```

#### Test 3

```
ULAZ:
1432
IZLAZ:
10
```

#### Test 4

```
ULAZ:
1
IZLAZ:
1
```

#### Test 5

```
ULAZ:
0
IZLAZ:
0
```



[Rešenje 1.21]

**Zadatak 1.22** Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva

- (a) sabirajući elemente počev od početka niza ka kraju niza,
- (b) sabirajući elemente počev od kraja niza ka početku niza.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije, zatim dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojnog niza, a potom i elemente niza. Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim nizom.

*Primer 1*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite redni broj funkcije (1 ili 2): 1
|| Unesite dimenziju niza: 5
|| Unesite elemente niza:
|| 1 2 3 4 5
|| Suma elemenata je 15

```

*Primer 2*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite redni broj funkcije (1 ili 2): 2
|| Unesite dimenziju niza: 4
|| Unesite elemente niza:
|| -5 2 -3 6
|| Suma elemenata je 0

```

[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. Njegovi elementi se unose sve do unosa kraja ulaza (EOF).

*Test 1*

```

|| ULAZ:
|| 3 2 1 4 21
|| IZLAZ:
|| 21

```

*Test 2*

```

|| ULAZ:
|| 2 -1 0 -5 -10
|| IZLAZ:
|| 2

```

*Test 3*

```

|| ULAZ:
|| 1 11 3 5 8 1
|| IZLAZ:
|| 11

```

[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. Prvo se unosi dimenzija nizova, a zatim i njihovi elementi. Nizovi neće imati više od 256 elemenata.

*Primer 1*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite dimenziju nizova: 3
|| Unesite elemente prvog niza:
|| 1 2 3
|| Unesite elemente drugog niza:
|| 1 2 3
|| Skalarni proizvod je 14

```

*Primer 2*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite dimenziju nizova: 2
|| Unesite elemente prvog niza:
|| 3 5
|| Unesite elemente drugog niza:
|| 2 6
|| Skalarni proizvod je 36

```

[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 broj  $x$  i niz  $a$  koji se unose sa standardnog ulaza. Prvo se unosi  $x$ , a zatim elementi niza sve do unosa kraja ulaza. Niz neće imati više od 256 elemenata.

*Primer 1*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ceo broj:
|| 4
|| Unesite elemente niza:
|| 1 2 3 4
|| Broj pojavljivanja je 1

```

*Primer 2*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ceo broj:
|| 11
|| Unesite elemente niza:
|| 3 2 11 14 11 43 1
|| Broj pojavljivanja je 2

```

*Primer 3*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ceo broj:
|| 1
|| Unesite elemente niza:
|| 3 21 5 6
|| Broj pojavljivanja je 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.

## Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite tri cela broja:
1 2 3
Unesite elemente niza:
11 1 2 4 3 6
Uneti brojevi nisu uzastopni clanovi niza.
```

[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 3

```
ULAZ:
    0xFFFFFFFF
IZLAZ:
    32
```

[Rešenje 1.27]

**Zadatak 1.28** Napisati rekurzivnu funkciju koja štampa bitovsku reprezentaciju neoznačenog celog broja, i program koji je testira za vrednost koja se zadaje sa standardnog ulaza.

*Test 2*

```

ULAZ:
    0
IZLAZ:
    00000000000000000000000000000000

```

**Zadatak 1.29** Napisati rekurzivnu funkciju za određivanje najveće cifre u oktalnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. UPUTSTVO: *Binarne cifre grupisati u podgrupe od po tri cifre, počev od bitova najmanje težine.*

### Test 3

ULAZ:	8
IZLAZ:	1

[Rešenje 1.29]

**Zadatak 1.30** Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. UPUTSTVO: *Binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.*

*Test 1*

```

|| ULAZ:
|| 5
|| IZLAZ:
|| 5

```

*Test 2*

```

|| ULAZ:
|| 16
|| IZLAZ:
|| 1

```

*Test 3*

```

|| ULAZ:
|| 18
|| IZLAZ:
|| 2

```

[Rešenje 1.30]

**Zadatak 1.31** Napisati rekurzivnu funkciju **palindrom** koja ispituje da li je data niska palindrom. Napisati program koji testira ovu funkciju na nisci koja se unosi sa standardnog ulaza. Pretpostaviti da niska neće imati više od 31 karaktera.

*Test 1*

```

|| ULAZ:
|| a
|| IZLAZ:
|| da

```

*Test 2*

```

|| ULAZ:
|| aa
|| IZLAZ:
|| da

```

*Test 3*

```

|| ULAZ:
|| aba
|| IZLAZ:
|| da

```

*Test 4*

```

|| ULAZ:
|| programiranje
|| IZLAZ:
|| ne

```

*Test 5*

```

|| ULAZ:
|| anavolimilovana
|| 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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 2 IZLAZ: 1 2 2 1 </pre>	<pre> ULAZ: 3 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1 </pre>	<pre> ULAZ: -5 Duzina permutacije mora biti broj veci od 0 i manji 50! </pre>

[Rešenje 1.32]

\* **Zadatak 1.33** Paskalov trougao sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbira jednog polja levo i jednog polja iznad. Izuzetak su jedinice na krajevima. Vrednosti brojeva Paskalovog trougla odgovaraju binomnim koeficijentima tj. vrednost polja  $(n, k)$ , gde je  $n$  redni broj hipotenuze, a  $k$  redni broj elementa u tom redu (na toj hipotenuzi) odgovara binomnom koeficijentu  $\binom{n}{k}$ , pri čemu brojanje počinje od nule. Na primer, vrednost polja  $(4, 2)$  je 6.

```

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

```

- Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta  $\binom{n}{k}$  koristeći osobine Paskalovog trougla.
- 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 iscrta Paskalov trougao, a zatim štampa sumu elemenata hipotenuze.

Test 1

```

ULAZ:
5 3
IZLAZ:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
8

```

Test 2

```

ULAZ:
6 5
IZLAZ:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
32

```

[Rešenje 1.33]

**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 1

```

ULAZ:
2
IZLAZ:
a a
a b
b a
b b

```

Test 2

```

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>
3
4  /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
   imaginaran deo kompleksnog broja */
5
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 argument
13 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 joj
22 se salje kao argument u obliku (x + i y) Ovoj funkciji se
23 kompleksan broj prenosi po vrednosti (za ispis nije neophodna
24 adresa) */
25 void ispisi_kompleksan_broj(KompleksanBroj z)
26 {
27     printf("(");
28     if (z.real != 0) {
29         printf("%.2f", z.real);
30         if (z.imag > 0)
31             printf(" +");
32     }
33     if (z.imag != 0)
34         printf(" %.2f i ", z.imag);
35
36     if (z.imag == 0 && z.real == 0)
37         printf("0 ");
38 }

```

```
40     printf(" ");
41 }
42
43 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
44 float realan_deo(KompleksanBroj z)
45 {
46     return z.real;
47 }
48
49 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
50 float imaginaran_deo(KompleksanBroj z)
51 {
52     return z.imag;
53 }
54
55 /* Funkcija vraca vrednost modula kompleksnog broja koji joj se salje
56    kao argument */
57 float moduo(KompleksanBroj z)
58 {
59     return sqrt(z.real * z.real + z.imag * z.imag);
60 }
61
62 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
63    odgovara kompleksnom broju poslatom kao argument */
64 KompleksanBroj konjugovan(KompleksanBroj z)
65 {
66     KompleksanBroj z1 = z;
67
68     z1.imag *= -1;
69
70     return z1;
71 }
72
73 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
74    argumenata funkcije */
75 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
76 {
77     KompleksanBroj z = z1;
78
79     z.real += z2.real;
80     z.imag += z2.imag;
81
82     return z;
83 }
84
85 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
86    argumenata funkcije */
87 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
88 {
89     KompleksanBroj z = z1;
90
91     z.real -= z2.real;
```



```
92     z.imag -= z2.imag;
94     return z;
96 }
98 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
    argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
100 {
    KompleksanBroj z;
102
    z.real = z1.real * z2.real - z1.imag * z2.imag;
104     z.imag = z1.real * z2.imag + z1.imag * z2.real;
106
    return z;
108 }
110 /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
    kao argument */
float argument(KompleksanBroj z)
112 {
    return atan2(z.imag, z.real);
114 }
116 int main()
118 {
    char c;
    /* Deklaracija 3 promenljive tipa KompleksanBroj */
    KompleksanBroj z1, z2, z;
122
    /* Ucitavanje znaka na osnovu koga korisnik bira koja aritmeticka
        operacija ce se vrsiti nad kompleksnim brojevima. */
124     printf("Unesite znak (+,-,*):");
    scanf("%c", &c);
126
    /* Ucitavanje prvog kompleksnog broja u promenljivu z1, a potom
        njegovo ispisivanje na standardni izlaz */
128     ucitaj_kompleksan_broj(&z1);
    ispisi_kompleksan_broj(z1);
130     printf("\n");
132
    /* Ucitavanje drugog kompleksnog broja u promenljivu z2, a potom
        njegovo ispisivanje na standardni izlaz */
134     ucitaj_kompleksan_broj(&z2);
    ispisi_kompleksan_broj(z2);
136     printf("\n");
138
    if (c == '+') {
140         /* Sabiranje kompleksnih brojeva */
        z = saberi(z1, z2);
        printf("\n");
142         ispisi_kompleksan_broj(z1);
```

```

144     printf(" + ");
        ispisi_kompleksan_broj(z2);
146     printf(" = ");
        ispisi_kompleksan_broj(z);
148     printf("\n");
    } else if (c == '-') {
150         /* Oduzimanje kompleksnih brojeva */
        z = oduzmi(z1, z2);
152         printf("\n");
        ispisi_kompleksan_broj(z1);
154         printf(" - ");
        ispisi_kompleksan_broj(z2);
156         printf(" = ");
        ispisi_kompleksan_broj(z);
158         printf("\n");
    } else {
160         /* Mnozenje kompleksnih brojeva */
        z = mnozi(z1, z2);
162         printf("\n");
        ispisi_kompleksan_broj(z1);
164         printf(" * ");
        ispisi_kompleksan_broj(z2);
166         printf(" = ");
        ispisi_kompleksan_broj(z);
168     }

170     /* Ispisuje se na standardni izlaz realan, imaginaran deo i moduo
        kompleksnog broja z1 */
172     printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
        realan_deo(z), imaginaran_deo(z), moduo(z));
174     printf("\n");

176     /* Racunanje i ispisivanje konjugovano kompleksan broj od z2 */
    printf("\nNjegov konjugovano kompleksan broj: ");
178     ispisi_kompleksan_broj(konjugovan(z));
    printf("\n");

180     /* Testiranje funkcije koja racuna argument kompleksnih brojeva */
182     printf("\nArgument kompleksnog broja %.f\n", argument(z));

184     return 0;
}

```

## Rešenje 1.2

Datoteka 1.1: *complex.h*

```

1  /*
3  Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
    deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavlje

```

```

5      nikada ne treba da sadrzi definicije funkcija. Da bi neki program
      mogao da koristi ove brojeve i funkcije iz ove biblioteke,
      neophodno je da ukljuci ovo zaglavlje. */
7
9      /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i time
      onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci. Niska
      posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
11     narednoj pretprocesorskoj define direktivi. */
      #ifndef _COMPLEX_H
13     #define _COMPLEX_H

15     /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
      koje se koriste u definicijama funkcija navedenim u complex.c */
17     #include <stdio.h>
      #include <math.h>
19
21     /* Struktura KompleksanBroj */
      typedef struct {
23         float real;
25         float imag;
27     } KompleksanBroj;

29     /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
      definisane u complex.c */

31     /* Funkcija ucitava sa standardnog ulaza realan i imaginaran deo
      kompleksnog broja i smesta ih u strukturu cija adresa je argument
      funkcije */
      void ucitaj_kompleksan_broj(KompleksanBroj * z);
33
35     /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
      se salje kao argument u obliku (x + y i) */
      void ispisi_kompleksan_broj(KompleksanBroj z);
37
39     /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
      float realan_deo(KompleksanBroj z);

41     /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
      float imaginaran_deo(KompleksanBroj z);
43
45     /* Funkcija vraca vrednost modula kompleksnog broja koji joj se salje
      kao argument */
      float moduo(KompleksanBroj z);
47
49     /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
      odgovara kompleksnom broju poslatom kao argument */
      KompleksanBroj konjugovan(KompleksanBroj z);
51
53     /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
      argumenata funkcije */
      KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
55

```

```

57  /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
    argumenata funkcije */
    KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
59
    /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
    argumenata funkcije */
61  KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
63
    /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
    kao argument */
65  float argument(KompleksanBroj z);
67
    /* Kraj zakljucanog dela */
69  #endif

```

Datoteka 1.2: *complex.c*

```

1  /* Ukljucuje se zaglavlje za rad sa kompleksnim brojevima, jer je
    neophodno da bude poznata definicija tipa KompleksanBroj.
3   Takodje, time su ukljucena zaglavlja standardne biblioteke koja
    su navedena u complex.h */
5  #include "complex.h"
7
    void ucitaj_kompleksan_broj(KompleksanBroj * z)
9  {
    /* Ucitavanje vrednosti sa standardnog ulaza */
11  printf("Unesite realan i imaginaran deo kompleksnog broja: ");
    scanf("%f", &z->real);
13  scanf("%f", &z->imag);
    }
15
    void ispisi_kompleksan_broj(KompleksanBroj z)
17  {
    /* Ispisuje se otvorena zagrada */
    printf("(");
19  /* Zatim se razlikuju dva slucaja
    1) realni deo kompleksnog broja razlicit od nule: tada se realni
    deo ispisuje na standardni izlaz, nakon cega se ispisuje znak +
21  ili - u zavisnosti da li je imaginarni deo pozitivan ili
    negativan, a potom i apsolutna vrednost imaginarnog dela
23  kompleksnog broja
    2) realni deo kompleksnog broja je nula: tada se samo ispisuje
    imaginaran deo, s tim sto se ukoliko su oba dela nula ispisuje
    samo 0, bez decimalnih mesta */
25
    if (z.real != 0) {
27  printf("%.2f", z.real);

    if (z.imag > 0)
31  printf(" + %.2f i", z.imag);
    else if (z.imag < 0)
33  printf(" - %.2f i", z.imag);
    }
    printf(")");
}

```

```
35     printf(" - %.2f i", -z.imag);
36 } else{
37     if (z.imag == 0)
38         printf("0");
39     else
40         printf("%.2f i", z.imag);
41 }

42
43 /* Ispisuje se otvorena zagrada */
44 printf(")");
45 }

46
47 float realan_deo(KompleksanBroj z)
48 {
49     /* Vraca se vrednost realnog dela kompleksnog broja */
50     return z.real;
51 }

52
53 float imaginaran_deo(KompleksanBroj z)
54 {
55     /* Vraca se vrednost imaginarnog dela kompleksnog broja */
56     return z.imag;
57 }

58
59 float moduo(KompleksanBroj z)
60 {
61     /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
62     return sqrt(z.real * z.real + z.imag * z.imag);
63 }

64
65 KompleksanBroj konjugovan(KompleksanBroj z)
66 {
67     /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
68        se promeni znak imaginarnom delu kompleksnog broja */
69     KompleksanBroj z1 = z;
70     z1.imag *= -1;
71     return z1;
72 }

73
74 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
75 {
76     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
77        broj ciji je realan deo zbir realnih delova kompleksnih brojeva
78        z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
79        brojeva z1 i z2 */
80     KompleksanBroj z = z1;
81
82     z.real += z2.real;
83     z.imag += z2.imag;
84
85     return z;
86 }
```

```

87 KomplexanBroj oduzmi(KomplexanBroj z1, KomplexanBroj z2)
89 {
    /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
91    broj ciji je realan deo razlika realnih delova kompleksnih
    brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
93    kompleksnih brojeva z1 i z2 */
    KomplexanBroj z = z1;

95    z.real -= z2.real;
97    z.imag -= z2.imag;

99    return z;
101 }

KomplexanBroj mnozi(KomplexanBroj z1, KomplexanBroj z2)
103 {
    /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
105    broj ciji se realan i imaginaran deo racunaju po formuli za
    mnozenje kompleksnih brojeva z1 i z2 */
107    KomplexanBroj z;

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

111    return z;
113 }

float argument(KomplexanBroj z)
115 {
117     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
    iz biblioteke math.h */
119     return atan2(z.imag, z.real);
121 }

```

Datoteka 1.3: *main.c*

```

/*****
2 Ovaj program koristi korektno definisanu biblioteku kompleksnih
brojeva. U zaglavlju complex.h nalazi se definicija kompleksnog broja
4 i popis deklaracija podrzanih funkcija, a u complex.c se nalaze
njihove definicije.

6
Kompilacija programa se najjednostavnije postize naredbom
8 gcc -Wall -lm -o complex complex.c main.c

10 Kompilacija se moze uraditi i na sledeci nacin:
gcc -Wall -c -o complex.o complex.c
12 gcc -Wall -c -o main.o main.c
gcc -lm -o complex complex.o main.o
14

```

```

16 | Napomena: Prethodne komande se koriste kada se sva tri navedena
    | dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
    | complex.c complex.h) nalazi u direktorijumu header_dir prevodjenje
18 | se vrši dodavanjem opcije opcije -I header_dir
    | gcc -I header_dir -Wall -lm -o complex complex.c main.c
20 | *****/
    |
22 | #include <stdio.h>
24 | /* Uključuje se zaglavlje neophodno za rad sa kompleksnim brojevima
    | */
    | #include "complex.h"
26 |
    | /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
28 |    | polarni oblik */
    | int main()
30 | {
    |     KompleksanBroj z;
32 |
    |     /* Učitavamo kompleksan broj */
34 |     ucitaj_kompleksan_broj(&z);
    |
36 |     printf("Polarni oblik kompleksnog broja je %.2f * e^i * %.2f\n",
    |           moduo(z), argument(z));
38 |
    |     return 0;
40 | }

```

### Rešenje 1.3

Datoteka 1.4: *polinom.h*

```

    | #ifndef _POLINOM_H
    | #define _POLINOM_H
2 |
    | #include <stdio.h>
    | #include <stdlib.h>
4 |
    | /* Maksimalni stepen polinoma */
6 |
    | #define MAX_STEPEN 20
8 |
    |
10 | /* Polinome predstavljamo strukturom koja cuva koeficijente (koef[i]
    |    | je koeficijent uz clan x^i) i stepen polinoma */
12 |
    | typedef struct {
14 |     double koef[MAX_STEPEN + 1];
    |     int stepen;
16 | } Polinom;
    |
18 | /* Funkcija koja ispisuje polinom na stdout u citljivom obliku

```

```

20     Polinom prenosimo po adresi, da bi uštedeli kopiranje cele
    strukture, vec samo prenosimo adresu na kojoj se nalazi polinom
    kog ispisujemo */
22 void ispisi(const Polinom * p);

24 /* Funkcija koja ucitava polinom sa tastature */
    Polinom ucitaj();

26
28 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
    algoritmom */
    double izracunaj(const Polinom * p, double x);

30
32 /* Funkcija koja sabira dva polinoma */
    Polinom saberi(const Polinom * p, const Polinom * q);

34 /* Funkcija mnozi dva polinoma p i q */
    Polinom pomnozi(const Polinom * p, const Polinom * q);

36
38 /* Funkcija racuna izvod polinoma p */
    Polinom izvod(const Polinom * p);

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

```

Datoteka 1.5: *polinom.c*

```

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

4
void ispisi(const Polinom * p)
6 {
    /* Ispisivanje polinoma pocinje od najviseg stepena kao najnizem da
8     bi polinom bio ispisan na prirodan nacin. Ispisuju se samo oni
    koeficijenti koji su razliciti od nule. Ispred pozitivnih
10    koeficijenata je potrebno ispisati znak + (osim u slucaju
    koeficijenta uz najvisi stepen). */
12    int i;
    for (i = p->stepen; i >= 0; i--) {
14
16        if (p->koef[i]) {
            if (p->koef[i] >= 0 && i != p->stepen)
                putchar('+');
18            if (i > 1)
                printf("%.2fx^%d", p->koef[i], i);
20            else if (i == 1)
                printf("%.2fx", p->koef[i]);
22            else
                printf("%.2f", p->koef[i]);
24        }
    }
}

```



```

    }
26     putchar('\n');
    }
28
    Polinom ucitaj()
30 {
    int i;
32     Polinom p;

34     /* Ucitavanje stepena polinoma */
    scanf("%d", &p.stepen);

36     /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
38     dozvoljenog opsega */
    while (p.stepen > MAX_STEPEN || p.stepen < 0) {
40         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
        scanf("%d", &p.stepen);
42     }

44     /* Unose se koeficijenti polinoma */
    for (i = p.stepen; i >= 0; i--)
46         scanf("%lf", &p.koef[i]);
    return p;
48 }

50 double izracunaj(const Polinom * p, double x)
52 {
    /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
    /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
54     svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
    vrednost odgovarajuceg koeficijenta */
56     double rezultat = 0;
    int i = p->stepen;
58     for (; i >= 0; i--)
        rezultat = rezultat * x + p->koef[i];
60     return rezultat;
    }

62
    Polinom saberi(const Polinom * p, const Polinom * q)
64 {
    Polinom rez;
66     int i;

68     /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
    stepenom */
70     rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;

72     /* Racunaju se svi koeficijenti rezultujucega polinoma: tako sto
    se sabiraju koeficijenti na odgovarajucim pozicijama u dva
74     polinoma koja sabiramo. Ukoliko je pozicija za koju se racuna
    koeficijent veca od stepena nekog od polaznih polinoma
76     podrazumeva se koeficijent jednak koeficijentu uz odgovarajuci

```

```

78     stepen iz drugog polinoma */
    for (i = 0; i <= rez.stepen; i++)
        rez.koef[i] =
80            (i > p->stepen ? 0 : p->koef[i]) + (i >
                                                    q->stepen ? 0 : q->
82                koef[i]);

    return rez;
84 }

86 Polinom pomnozi(const Polinom * p, const Polinom * q)
87 {
88     int i, j;
89     Polinom r;

90     /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
91        */
92     r.stepen = p->stepen + q->stepen;
93     if (r.stepen > MAX_STEPEN) {
94         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
95         exit(EXIT_FAILURE);
96     }
97 }

98     /* Svi koeficijenti rezultujuceg polinoma se inicijalizuju na nulu
99        */
100     for (i = 0; i <= r.stepen; i++)
101         r.koef[i] = 0;

102     /* U svakoj iteraciji odgovarajuci koeficijent rezultata se uvecava
103        za proizvod odgovarajucih koeficijenata iz polaznih polinoma */
104     for (i = 0; i <= p->stepen; i++)
105         for (j = 0; j <= q->stepen; j++)
106             r.koef[i + j] += p->koef[i] * q->koef[j];

107     return r;
108 }

109 Polinom izvod(const Polinom * p)
110 {
111     int i;
112     Polinom r;

113     /* Izvod polinoma ce imati stepen za jedan stepen manji od stepena
114        polaznog polinoma. Ukoliko je stepen polinoma p vec nula, onda
115        je rezultujući polinom nula (izvod od konstante je nula). */
116     if (p->stepen > 0) {
117         r.stepen = p->stepen - 1;

118         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
119            polaznog polinoma */
120         for (i = 0; i <= r.stepen; i++)
121             r.koef[i] = (i + 1) * p->koef[i + 1];
122     }
123 }

```

```

130     } else
        r.koef[0] = r.stepen = 0;

132     return r;
134 }

134 Polinom nIzvod(const Polinom * p, int n)
136 {
138     int i;
138     Polinom r;

140     /* Provera da li n nenegativan */
142     if (n < 0) {
144         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
146         exit(EXIT_FAILURE);
148     }

146     /* Multi izvod je bas taj polinom */
148     if (n == 0)
150         return *p;

150     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove funkcija
152        za racunanje prvog izvoda polinoma */
152     r = izvod(p);
154     for (i = 1; i < n; i++)
156         r = izvod(&r);

156     return r;
158 }

```

Datoteka 1.6: *main.c*

```

1  #include <stdio.h>
2  #include "polinom.h"
3
4  int main(int argc, char **argv)
5  {
6      Polinom p, q, r;
7      double x;
8      int n;
9
10     /* Unos polinoma p */
11     printf
12         ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
13          najveceg stepena do nultog):\n");
14     p = ucitaj();
15
16     /* Ispis polinoma p */
17     ispisi(&p);
18
19     /* Unos polinoma q */

```

```

19     printf
        ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
        od najveceg stepena do nultog):\n");
21     q = ucitaj();

23     /* Polinomi se sabiraju i ispisuje se izracunati zbir */
    r = saberi(&p, &q);
25     printf("Zbir polinoma je: ");
    ispisi(&r);

27     /* Polinomi se mnoze i ispisuje se izracunati proizvod */
29     r = pomnozi(&p, &q);
    printf("Prozvod polinoma je polinom r:\n");
31     ispisi(&r);

33     printf("Unesite tacku u kojoj racunate vrednost polinoma r\n");
    scanf("%lf", &x);

35     /* Ispisuje se vrednost polinoma u unetoj tacki */
37     printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&r, x));

39     /* Racunanje n-tog izvoda polinoma p i ispisivanje dobijenog
        polinoma */
41     printf("Unesite izvod polinoma koji zelite:\n");
    scanf("%d", &n);
43     r = nIzvod(&p, n);
    printf("%d. izvod polinoma r je: ", n);
45     ispisi(&r);

47     return 0;
}

```

### Rešenje 1.5

```

#include <stdio.h>

2
/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
4     celog broja u memoriji. Bitove koji predstavljaju binarnu
    reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
6     najvece tezine ka 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 svim ostalim
16     mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto

```

```

18     se jedini bit jedinica pomera udesno, kako bi se ocitao naredni
20     bit broja x koji je argument funkcije. Odgovarajuci karakter,
22     ('0' ili '1'), ispisuje se na standardnom izlazu. Neophodno je
24     da promenljiva maska bude deklarirana kao neoznaceni ceo broj
26     kako bi se siftovanjem u desno vrsilo logicko siftovanje
28     (popunjavanje nulama) a ne aritmeticko siftovanje (popunjavanje
30     znakom broja). */
32     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
34         putchar(x & maska ? '1' : '0');
36
38     putchar('\n');
39 }
40
41 int main()
42 {
43     int broj;
44     scanf("%x", &broj);
45     print_bits(broj);
46
47     return 0;
48 }

```

## Rešenje 1.6

```

1  #include <stdio.h>
2
3  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
4     celog broja u memoriji */
5  void print_bits(int x)
6  {
7     unsigned velicina = sizeof(int) * 8;
8     unsigned maska;
9
10     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
12         putchar(x & maska ? '1' : '0');
14
16     putchar('\n');
17 }
18
19 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
20    kreiranjem odgovarajuce maske i njenim pomeranjem */
21 int count_bits1(int x)
22 {
23     int br = 0;
24     unsigned wl = sizeof(unsigned) * 8 - 1;
25
26     /* Formiranje se maska cija binarna reprezentacija izgleda
27        100000...0000000, koja služi za ocitavanje bita najveće težine.
28        U svakoj iteraciji maska se pomera u desno za 1 mesto, i
29        ocitavamo sledeći bit. Petlja se završava kada više nema

```

```

27     jedinica tj. kada maska postane nula. */
    unsigned maska = 1 << wl;
29     for (; maska != 0; maska >>= 1)
        x & maska ? br++ : 1;

31
    return br;
33 }

35 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
    formiranjem odgovarajuće maske i pomeranjem promenljive x */
37 int count_bits2(int x)
{
39     int br = 0;
    unsigned wl = sizeof(int) * 8 - 1;

41
    /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1
43     vrsila bi aritmeticko pomeranje u desno, tj. popunjavanje bita
    najveće težine bitom znaka. U tom slučaju nikad ne bi bio
45     ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
    petlji. Zbog toga se koristi pomeranje broja x ulevo i maska
47     koja ocitava bit najveće težine. */

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

53     return br;
}

55

57 int main()
{
59     int x, i;
    printf("Unesite broj:\n");
61     scanf("%x", &x);
    printf("Unesite redni broj funkcije:\n");
63     scanf("%d", &i);
    printf("Broj jedinica u zapisu je\n");
65     if (i == 1)
        printf("funkcija count_bits1: %d\n", count_bits1(x));
67     else
        printf("funkcija count_bits2: %d\n", count_bits2(x));
69     return 0;
}

```

### Rešenje 1.7

```

1 #include <stdio.h>

3 /* Funkcija vraca najveći neoznaceni broj sastavljen od istih bitova
    koji se nalaze u binarnoj reprezentaciji vrednosti promenljive x */

```

```
5 unsigned najveći(unsigned x)
6 {
7     unsigned velicina = sizeof(unsigned) * 8;
8
9     /* Formira se maska 100000...00000000 */
10    unsigned maska = 1 << (velicina - 1);
11
12    /* Rezultat se inicijalizuje vrednoscu 0 */
13    unsigned rezultat = 0;
14
15    /* Promenljiva x se pomera u levo sve dok postoje jedinice u njoj
16       binarnoj reprezentaciji (tj. sve dok je promenljiva x razlicita
17       od nule). */
18    for (; x != 0; x <<= 1) {
19        /* Za svaku jedinicu koja se koriscenjem maske detektuje na
20           poziciji najveće težine u binarnoj reprezentaciji promenljive
21           x, potiskuje se jedna nova jedinicu sa leva u rezultat */
22        if (x & maska) {
23            rezultat >>= 1;
24            rezultat |= maska;
25        }
26    }
27
28    return rezultat;
29 }
30
31 /* Funkcija vraca najmanji neoznaceni broj sastavljen od istih bitova
32    koji se nalaze u binarnoj reprezentaciji vrednosti promenljive x */
33 unsigned najmanji(unsigned x)
34 {
35     /* Rezultat se inicijalizuje vrednoscu 0 */
36     unsigned rezultat = 0;
37
38     /* Promenljiva x se pomera u desno sve dok postoje jedinice u
39        njoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
40        razlicita od nule). */
41    for (; x != 0; x >>= 1) {
42        /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
43           detektuje na poziciji najmanje težine u binarnoj
44           reprezentaciji promenljive x, potiskuje se jedna nova jedinicu
45           sa desna u rezultat */
46        if (x & 1) {
47            rezultat <<= 1;
48            rezultat |= 1;
49        }
50    }
51
52    return rezultat;
53 }
54
55 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
56    celog broja u memoriji */
```

```

57 void print_bits(int x)
58 {
59     unsigned velicina = sizeof(int) * 8;
60     unsigned maska;
61
62     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
63         putchar(x & maska ? '1' : '0');
64
65     putchar('\n');
66 }
67
68 int main()
69 {
70     int broj;
71     scanf("%x", &broj);
72
73     printf("Najveci:\n");
74     print_bits(najveci(broj));
75
76     printf("Najmanji:\n");
77     print_bits(najmanji(broj));
78
79     return 0;
80 }

```

### Rešenje 1.8

```

1  #include <stdio.h>
2
3
4  /*****
5   Funkcija postavlja na nulu n bitova pocev od pozicije p.
6   Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
7   se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
8   1010 1110 0111 1010 1011 1100 1101 1110 1000 0010 0111 */
9   unsigned reset(unsigned x, unsigned n, unsigned p)
10  {
11      /*****
12       Cilj je anulirati samo zeljene bitove, a da ostali
13       ostanu nepromenjeni. Maska koja ce se koristiti je ona cija
14       binarna reprezentacija ima n bitova
15       postavljenih na 0 pocev od pozicije p, dok su svi ostali
16       postavljeni na 1.
17
18       Na primer, za n=5 i p=10 cilj je maska oblika
19       1111 1111 1111 1111 1111 1000 0011 1111
20       To se postize na sledeci nacin:
21       ~0                1111 1111 1111 1111 1111 1111 1111 1111
22       (~0 << n)         1111 1111 1111 1111 1111 1111 1110 0000
23       ~(~0 << n)        0000 0000 0000 0000 0000 0000 0001 1111
24       ~(~0 << n) << ( p-n+1)) 0000 0000 0000 0000 0000 0111 1100 0000

```



```

26 ~((~0 << n) << (p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111
*****/
   unsigned maska = ~(~0 << n) << (p - n + 1));
28
   return x & maska;
30 }

32
33 /*****
34  Funkcija postavlja na 1 n bitova pocev od pozicije p.
35  Pozicije se broje pocev od pozicije najnižeg bita, pri čemu
36  se broji od nule . Npr, za n=5, p=10
37  1010 1011 1100 1101 1110 1010 1110 0111
38  1010 1011 1100 1101 1110 1111 1110 0111
39  *****/
40 unsigned set(unsigned x, unsigned n, unsigned p)
41 {
42
43  /*****
44   Cilj je samo određenih n bitova postaviti na 1, dok
45   ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
46   formira se maska oblika
47   0000 0000 0000 0000 0000 0111 1100 0000
48   prateći vrlo sličan postupak kao za prethodnu funkciju
49  *****/
50  unsigned maska = ~(~0 << n) << (p - n + 1);
51
52  return x | maska;
53 }
54
55 /*****
56  Funkcija vraća celobrojno polje bitova, desno poravnato, koje
57  predstavlja n bitova pocev od pozicije p u binarnoj
58  reprezentaciji broja x, pri čemu se pozicija broji sa desna
59  ulevo, gde je početna pozicija 0. Na primer za n = 5 i p = 10
60  i broj čija je binarna reprezentacija:
61  1010 1011 1100 1101 1110 1010 1110 0111
62  traži se
63  0000 0000 0000 0000 0000 0000 0000 1011
64  *****/
65  unsigned get_bits(unsigned x, unsigned n, unsigned p)
66  {
67
68  /*****
69   Kreira se maska kod koje su poslednjih n bitova 1, a
70   ostali su 0. Na primer za n=5
71   0000 0000 0000 0000 0000 0000 0001 1111
72  *****/
73  unsigned maska = ~(~0 << n);
74
75  /* Najpre se vrednost promenljive x pomera u desno tako da traženo
76

```

```

78     polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
       zeljenih n i funkcija vraća tako dobijenu vrednost */
80     return maska & (x >> (p - n + 1));
}

82
/* Funkcija vraća broj x kome su n bitova počev od pozicije p
84     postavljeni na vrednosti n bitova najniže težine binarne
       reprezentacije broja y */
86 unsigned set_n_bits(unsigned x, unsigned n, unsigned p, unsigned y)
{
88     /******
       Kreira se maska kod koje su poslednjih n bitova 1, a
90     ostali su 0. Na primer za n=5
       0000 0000 0000 0000 0000 0000 0001 1111
92     *****/
       unsigned last_n_1 = ~(~0 << n);
94     /******
       Kao što je i u funkciji reset, i ovde se kreira masku koja ima n
96     bitova postavljenih na 0 počevši od pozicije p, dok su
       ostali bitovi 1. Na primer za n=5 i p =10
98     1111 1111 1111 1111 1111 1000 0011 1111
       *****/
100    unsigned middle_n_0 = ~(~(~0 << n) << (p - n + 1));

102    /* U promenljivu x_reset se smesta vrednost dobijena kada se u
       binarnoj reprezentaciji vrednosti promenljive x resetuje n
104    bitova na pozicijama počev od p */
       unsigned x_reset = x & middle_n_0;

106
       /* U promenljivu y_shift_middle se smesta vrednost dobijena od
       binarne reprezentacije vrednosti promenljive y čijih je n bitova
108     najniže težine pomera tako da stoje počev od pozicije p. Ostali
       bitovi su nule. (y & last_n_1) Resetuju se svi bitovi osim
110     najnižih n */
       unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);

112
       return x_reset ^ y_shift_middle;
114     }
}

116
/* Funkcija invertuje bitove u zapisu broja x počevši od pozicije p
       njih n */
118 unsigned invert(unsigned x, unsigned n, unsigned p)
{
120     /******
       Formira se maska sa n jedinica počev od pozicije p.
122     Na primer za n=5 i p=10
       0000 0000 0000 0000 0000 0111 1100 0000
124     *****/
       unsigned maska = ~(~0 << n) << (p - n + 1);
126
128

```

```
130  /* Operator ekskluzivno ili invertuje sve bitove gde je
    odgovarajući bit maske 1. Ostali bitovi ostaju nepromenjeni. */
132  return maska ^ x;
134
136  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
    celog broja u memoriji */
138  void print_bits(int x)
140  {
    unsigned velicina = sizeof(int) * 8;
    unsigned maska;

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

144    putchar('\n');
146  }
148
150  int main()
152  {
    unsigned x, p, n, y;
    printf("Unesite neoznaceni broj x:\n");
    scanf("%u", &x);
    printf("Unesite neoznaceni broj n:\n");
    scanf("%u", &n);
    printf("Unesite neoznaceni broj p:\n");
    scanf("%u", &p);
    printf("Unesite neoznaceni broj y:\n");
    scanf("%u", &y);

162    /* Stampaње broja x i y */
    printf("x = %u = ", x);
    print_bits(x);

166    printf("y = %u = ", y);
    print_bits(y);

168    printf("reset(%u,%u,%u) = ", x, n, p);
    print_bits(reset(x, n, p));

170    printf("set(%u,%u,%u) = ", x, n, p);
    print_bits(set(x, n, p));

172    printf("get_bits(%u,%u,%u) = ", x, n, p);
    print_bits(get_bits(x, n, p));

174    printf("set_n_bits(%u,%u,%u,%u) = ", x, n, p, y);
    print_bits(set_n_bits(x, n, p, y));
180  }
```

```

182 printf("invert(%u,%u,%5u) = ", x, n, p);
    print_bits(invert(x, n, p));
184 return 0;
}

```

### Rešenje 1.9

```

1  #include <stdio.h>

3  /*****
   Funkcija binarnu reprezentaciju svog argumenta x rotira u
   5  levo za n mesta i vraća odgovarajući neoznačen ceo broj čija
   je binarna reprezentacija dobijena nakon rotacije.
   7  Na primer za n=5 i x čija je interna reprezentacija
   1010 1011 1100 1101 1110 0001 0010 0011
   9  funkcija vraća neoznačen ceo broj čija je binarna
   reprezentacija:
   11  0111 1001 1011 1100 0010 0100 0111 0101
   *****/
13  unsigned rotate_left(int x, unsigned n)
   {
15     unsigned first_bit;
   /* Maska koja ima samo najviši bit postavljen na 1 neophodna da bi
   17     pre siftovanja u levo za 1 najviši bit bio sačuvan. */
   unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
   19     int i;

21     /* n puta se vrši rotaciju za jedan bit u levo. U svakoj iteraciji
   se odredi prvi bit, a potom se pomera binarna reprezentacija
   23     trenutne vrednosti promenljive x u levo za 1. Nakon toga, potom
   najniži bit se postavlja na vrednost koju je imao prvi bit koji
   25     je istisnut siftovanjem */
   for (i = 0; i < n; i++) {
27         first_bit = x & first_bit_mask;
         x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
29     }
   return x;
31 }

33 /*****
   Funkcija neoznačen broj x rotira u desno za n.
   35  Na primer za n=5 i x čija je binarna reprezentacija
   1010 1011 1100 1101 1110 0001 0010 0011
   37  funkcija vraća neoznačen ceo broj čija je binarna
   reprezentacija:
   39  0001 1101 0101 1110 0110 1111 0000 1001
   *****/
41  unsigned rotate_right(unsigned x, unsigned n)
   {
43     unsigned last_bit;

```

```

45     int i;

46     /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
47        iteraciji se odredjuje bit najmanje tezine broja x, zatm tako
48        odredjeni bit se siftuje u levo tako da najnizi bit dode do
49        pozicije najviseg bita. Zatim, nakon siftovanja binarne
50        reprezentacije trenutne vrednosti promenljive x za 1 u desno,
51        najvisi bit se postavlja na vrednost vec zapamcenog bita koji je
52        bio na poziciji najmanje tezine. */
53     for (i = 0; i < n; i++) {
54         last_bit = x & 1;
55         x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
56     }
57
58     return x;
59 }

60 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
61    argument funkcije x oznaceni ceo broj */
62 int rotate_right_signed(int x, unsigned n)
63 {
64     unsigned last_bit;
65     int i;

66
67
68     /* U svakoj iteraciji se odredjuje bit najmanje tezine i smesta u
69        promenljivu last_bit. Kako je x oznacen ceo broj, tada se
70        prilikom siftovanja u desno vrši aritmeticki sift i cuva se znak
71        broja. Dakle, razlikuju se dva slucaja u zavisnosti od znaka od
72        x. Nije dovoljno da se ova provera izvrši pre petlje, s obzirom
73        da rotiranjem u desno na mesto najviseg bita moze doci i 0 i 1,
74        nezavisno od pocetnog znaka broja smestenog u promenljivu x. */
75     for (i = 0; i < n; i++) {
76         last_bit = x & 1;

77
78         if (x < 0)
79             /******
80                Siftovanjem u desno broja koji je negativan dobija se 1
81                kao bit na najvisoj poziciji. Na primer ako je x
82                1010 1011 1100 1101 1110 0001 0010 001b
83                (sa b je oznacen ili 1 ili 0 na najnizjoj poziciji)
84                Onda je sadržaj promenljive last_bit:
85                0000 0000 0000 0000 0000 0000 0000 000b
86                Nakon siftovanja sadržaja promenljive x za 1 u desno
87                1101 0101 1110 0110 1111 0000 1001 0001
88                Kako bi umesto 1 na najvisoj poziciji u trenutnoj
89                binarnoj reprezentaciji x bilo postavljeno b nije
90                dovoljno da se siftuje na najvisu poziciju jer bi se
91                time dobile 0, a u ovom slucaju su potrebne jedinice
92                zbog bitovskog & zato se prvo vrši komplementiranje, a
93                zatim siftovanje
94                ~last_bit << (sizeof(int)*8 -1)

```

```

97         B000 0000 0000 0000 0000 0000 0000 0000
gde B oznacava ~b.
Potom se ponovo vrsi komplementiranje kako bi se b
99 nalazilo na najvisoj poziciji i sve jedinice na ostalim
pozicijama
101 ~(~last_bit << (sizeof(int)*8 -1))
b111 1111 1111 1111 1111 1111 1111 1111
103 *****/
x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
105 else
x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
107 }

109 return x;
}

111

113 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
celog broja u memoriji */
115 void print_bits(int x)
{
117     unsigned velicina = sizeof(int) * 8;
unsigned maska;
119     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
putchar(x & maska ? '1' : '0');

121     putchar('\n');
123 }

125 int main()
{
127     unsigned x, k;
printf("Unesite neoznacen ceo broj x:");
129     scanf("%x", &x);
printf("Unesite neoznacen ceo broj k:");
131     scanf("%x", &k);
printf("x = ", "");
133     print_bits(x);
printf("rotate_left(%u,%u) = ", x, k);
135     print_bits(rotate_left(x, k));

137     printf("rotate_right(%u,%u) = ", x, k);
print_bits(rotate_right(x, k));

139     printf("rotate_right_signed(%u,%u) = ", x, k);
141     print_bits(rotate_right_signed(x, k));

143     return 0;
}

```

## Rešenje 1.10

```

1 #include <stdio.h>

3 /******
   Funkcija vraća vrednost cija je binarna reprezentacija slika
   u ogledalu binarne reprezentacije broja x koji se prosledjuje
   kao argument funkcije. Na primer za x
   cija binarna reprezentacija izgleda ovako
   101010111100110111100100100100011
   funkcija treba da vrati broj cija binarna reprezentacija
   izgleda:
   11000100100001111011001111010101
   *****/
13 unsigned mirror(unsigned x)
14 {
15     unsigned najnizi_bit;
16     unsigned rezultat = 0;
17
18     int i;
19     /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
       vrednosti broja x se određuje i pamti u promenljivoj
       najnizi_bit, nakon čega se na promenljivu x primeni siftovanje u
       desno. */
23     for (i = 0; i < sizeof(x) * 8; i++) {
24         najnizi_bit = x & 1;
25         x >>= 1;
26         /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
           postavljeni bitovi dobijaju veću poziciju. Novi bit se
           postavlja na najnižu poziciju */
29         rezultat <<= 1;
30         rezultat |= najnizi_bit;
31     }
32     return rezultat;
33 }

35
36 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji */
37 void print_bits(int x)
38 {
39     unsigned velicina = sizeof(int) * 8;
40     unsigned maska;
41     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
42         putchar(x & maska ? '1' : '0');
43
44     putchar('\n');
45 }

47 int main()
48 {
49     int broj;
50     scanf("%x", &broj);
51

```

```

53  /* Ispisuje se binarna reprezentaciju unetog broja */
    print_bits(broj);
55
57  /* Ispisuje se binarna reprezentaciju broja dobijenog pozivom
    funkcije mirror */
    print_bits(mirror(broj));
59
    return 0;
61 }

```

### Rešenje 1.11

```

#include <stdio.h>
2
/* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
   jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
4  int Broj01(unsigned int n)
6  {
8      int broj_nula, broj_jedinica;
      unsigned int maska;
10
      broj_nula = 0;
      broj_jedinica = 0;
12
14     /* Maska je inicijalizovana tako da moze da analizira bit najvece
       tezine */
       maska = 1 << (sizeof(unsigned int) * 4 - 1);
16
18     /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
       iteraciji pomera u desno pa ce jedini bit koji je postavljen na
       1 biti na svim pozicijama u binarnoj reprezentaciji maske */
       while (maska != 0) {
22
24         /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili
           1 i uveca se odgovarajuci brojac */
           if (n & maska) {
26             broj_jedinica++;
           } else {
28             broj_nula++;
           }
30
           /* Pomera se maska u desnu stranu */
           maska = maska >> 1;
32     }
34
36     /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
       suprotnom vraca 0 */
       return (broj_jedinica > broj_nula) ? 1 : 0;
38

```



```
    }  
40  
    int main()  
42    {  
        unsigned int n;  
44  
        /* Sa standardnog ulaza se unosi neoznaceni ceo broj */  
46        scanf("%u", &n);  
48  
        printf("%d\n", Broj01(n));  
50  
        return 0;  
    }
```

## Rešenje 1.12

```
#include <stdio.h>  
2  
int broj_parova(unsigned int x)  
4    {  
6        int broj_parova;  
        unsigned int maska;  
8  
        /* Vrednost promenljive koja predstavlja broj parova se  
10        inicijalizuje na 0 */  
        broj_parova = 0;  
12  
        /* Postavlja se maska tako da moze da procitamo da li su dva  
14        najmanja bita u zapisu broja x 11 */  
        /* Binarna reprezentacija broja 3 je 000...00011 */  
16        maska = 3;  
18  
        while (x != 0) {  
20            /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par  
            */  
            if ((x & maska) == maska) {  
22                broj_parova++;  
            }  
24  
            /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji  
26            proveravao sledeci par bitova. Pomeranjem u desno bit najvece  
            tezine se popunjava nulom jer je x neoznaceni broj. */  
28            x = x >> 1;  
        }  
30  
        return broj_parova;  
32    }  
34
```

```

36 int main()
37 {
38     unsigned int x;
39
40     scanf("%u", &x);
41
42     printf("%d\n", broj_parova(x));
43
44     return 0;
45 }

```

### Rešenje 1.14

```

1  #include <stdio.h>
2
3  /*
4   Niska koja se formiramo je duzine (sizeof(unsigned int)*8)/4 +1
5   jer su za svaku heksadekadnu cifru potrebne 4 binarne cifre i
6   jedna dodatna pozicija za terminirajucu nulu.
7
8   Prethodni izraz je identican sa sizeof(unsigned int)*2+1. */
9
10 #define MAX_DUZINA sizeof(unsigned int)*2 +1
11
12 void prevod(unsigned int x, char s[])
13 {
14
15     int i;
16     unsigned int maska;
17     int vrednost;
18
19     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
20     maska za citanje 4 uzastopne cifre */
21     maska = 15;
22
23     /******
24     Broj se posmatra od pozicije najmanje tezine ka poziciji
25     najvece tezine. Na primer za broj
26     00000000001101000100001111010101
27     u prvom koraku se citaju bitovi izdvojeni sa <...>:
28     0000000000110100010000111101<0101>
29     u drugom koraku:
30     000000000011010001000011<1101>0101
31     u trecem koraku:
32     00000000001101000100<0011>11010101 i tako redom
33
34     Indeks i oznacava poziciju na koju se smesta vrednost.
35
36     */
37     for (i = MAX_DUZINA - 2; i >= 0; i--) {

```

```

40      /* Vrednost izdvojene cifre */
      vrednost = x & maska;

42      /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
      dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
44      10 do 15 odgovarajuci karakter se dobija tako sto se prvo
      oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na tako
46      dobijenu vrednost doda ASCII kod 'A' (time se dobija
      odgovarajuće slovo 'A', 'B', ... 'F') */
48      if (vrednost < 10) {
          s[i] = vrednost + '0';
50      } else {
          s[i] = vrednost - 10 + 'A';
52      }

54      /* Primenljiva x se pomera za 4 bita u desnu stranu i time ce u
      narednoj iteraciji biti posmatrane sledeće 4 cifre */
56      x = x >> 4;
    }

    s[MAX_DUZINA - 1] = '\0';
60 }

62 int main()
{
64     unsigned int x;
66     char s[MAX_DUZINA];

68     scanf("%u", &x);

70     prevod(x, s);

72     printf("%s\n", s);

74     return 0;
}

```

### Rešenje 1.17

```

#include <stdio.h>

2

4 /******
   Linearno resenje se zasniva na cinjenici:
   x^0 = 1 x^k = x * x^(k-1)
6   *****/
int stepen(int x, int k)
8 {
10     if (k == 0)
        return 1;

```

```

12     return x * stepen(x, k - 1);
13 }
14
15 /*****
16     Celo telo funkcije se moze ovako kratko zapisati
17     return k == 0 ? 1 : x * stepen(x,k-1);
18
19     Druga verzija prethodne funkcije. Obratiti paznju na
20     efikasnost u odnosu na prvu verziju!
21     Logaritamsko resenje je zasnovano na cinjenicama:
22         x^0 =1;
23         x^k = x * (x^2 )^(k/2) , za neparno k
24         x^k = (x^2)^(k/2) , za parno k
25     Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
26     doslo do rezultata, i stoga je efikasnije.
27 *****/
28 int stepen2(int x, int k)
29 {
30     if (k == 0)
31         return 1;
32
33     /* Ako je stepen paran */
34     if ((k % 2) == 0)
35         return stepen2(x * x, k / 2);
36     /* Inace (ukoliko je stepen neparan) */
37     return x * stepen2(x * x, k / 2);
38 }
39
40 int main()
41 {
42     int x, k, ind;
43     /*Unosi se redni broj funkcije koja ce se primeniti */
44     printf("Unesite redni broj funkcije (1/2):\n");
45     scanf("%d", &ind);
46     /* Unose se vrednosti za x i k */
47     printf("Unesite broj x:\n");
48     scanf("%d", &x);
49     printf("Unesite broj k:\n");
50     scanf("%d", &k);
51
52     /* Ispisivanje vrednsoti koju vraca odgovarajuca funkcija */
53     if(x == 1)
54         printf("%d\n", stepen(x, k));
55     else
56         printf("%d\n",stepen2(x, k));
57
58     return 0;
59 }

```

Rešenje 1.18

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
   (posrednu) rekurziju. */
7
8 /* Deklaracija funkcije neparan mora da bude navedena jer se ta
   funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
   definicije. Funkcija je mogla biti deklarirana i u telu funkcije
   paran. */
9
10 unsigned neparan(unsigned n);
11
12 /* Funkcija vraca 1 ako broj n ima paran broj cifara inace vraca 0.
   */
13 unsigned paran(unsigned n)
14 {
15     if (n <= 9)
16         return 0;
17     else
18         return neparan(n / 10);
19 }
20
21 /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace vraca
   0. */
22 unsigned neparan(unsigned n)
23 {
24     if (n <= 9)
25         return 1;
26     else
27         return paran(n / 10);
28 }
29
30 /* Glavna funkcija za testiranje */
31 int main()
32 {
33     int n;
34     scanf("%d", &n);
35     printf("Uneti broj ima %s paran broj cifara.\n",
36           (paran(n) == 1 ? "" : "ne"));
37     return 0;
38 }
```

### Rešenje 1.19

```
1 #include <stdio.h>
2
3 /* Pomocna funkcija koja izracunava n! * result. Koristi repnu
   rekurziju. Result je argument u kome se akumulira do tada
```

```
5         izracunatu vrednost faktorijela. Kada dodje do izlaza iz
        rekurziije iz rekurziije potrebno je da vratimo result. */
7     int faktorijelRepna(int n, int result)
8     {
9         if (n == 0)
10             return result;
11
12         return faktorijelRepna(n - 1, n * result);
13     }
14
15     /* U sledece dve funkcije je prikazan postupak oslobadjanja od repne
        rekurziije koja postoji u funkciji faktorijelRepna.
16
17     Funkcija se transformise tako sto rekurzivni poziv zemeni sa
        naredbama kojima se vrednost argumenta funkcije postavlja na
18     vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
        goto naredbe za vracanje na pocetak tela funkcije. */
19
20     int faktorijelRepna_v1(int n, int result)
21     {
22     pocetak:
23         if (n == 0)
24             return result;
25
26         result = n * result;
27         n = n - 1;
28         goto pocetak;
29     }
30
31     /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
        praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
        iterativno resenje bez bezuslovnih skokova: */
32
33     int faktorijelRepna_v2(int n, int result)
34     {
35         while (n != 0) {
36             result = n * result;
37             n = n - 1;
38         }
39
40         return result;
41     }
42
43     /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
        broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
        ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
        udobnosti korisnika, jer je sasvim prirodno da za faktorijel
        zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
        sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
        faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
        funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
        poziv faktorijelRepna sa pozivom faktorijelRepna_v1, a zatim sa
        pozivom funkcije faktorijelRepna_v2. */
44
```

```
int faktorijel(int n)
57 {
    return faktorijelRepna(n, 1);
59 }

int main()
{
63     int n;

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

69     while (1);

71     return 0;
}
```

### Rešenje 1.20

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
5  int F_iterativna(int n, int a, int b)
   {
7      int i;
      int F_0 = 0;
9      int F_1 = 1;
      int tmp;

11     if (n == 0)
13         return 0;

15     for (i = 2; i <= n; i++) {
        tmp = a * F_1 + b * F_0;
17         F_0 = F_1;
        F_1 = tmp;
19     }

21     return F_1;
   }
23
   /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
25 int F_rekurzivna(int n, int a, int b)
   {
27     /* Izlaz iz rekurzije */
      if (n == 0 || n == 1)
29         return n;

31     /* Rekurzivni pozivi */
```

```

33     return a * F_rekurzivna(n - 1, a, b) + b * F_rekurzivna(n - 2,
34                                     a, b);
35 }
36
37 /* NAPOMENA: U slucaju da se rekurzijom problem svodi na vise manjih
38 podproblema koji se mogu preklapati, postoji opasnost da se
39 pojedini podproblemi manjih dimenzija resavaju veci broj puta.
40 Npr. F(20) = a*F(19) + b*F(18), a F(19) = a*F(18) + b*F(17), tj.
41 problem fibonacci(18) se resava dva puta! Problemi manjih
42 dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
43 problem je kombinacija rekurzije sa dinamicnim programiranjem.
44 Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
45 memoriji (obicno u nizovima ili matricama), odakle se koriste ako
46 tokom resavanja ponovo budu potrebni.
47
48 U narednoj funkciji vec izracunati clanovi niza se cuvaju u
49 statickom nizu celih brojeva, jer taj niz onda nece biti smesten
50 na stek, kao sto je slucaj sa lokalnim promenljivama, vec u
51 statickoj memoriji odakle ce biti dostupan svim pozivima
52 rekurzivne funkcije. */
53
54 /* c) Funkcija racuna n-ti fibonaccijev broj - napredna rekurzivna
55 verzija */
56 int F_napredna(int n, int a, int b)
57 {
58     /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
59     staticke promenljive na podrazumevane vrednosti. Stoga, elemente
60     celobrojnog niza inicijalizuje na 0 */
61     static int f[20];
62
63     /* Ako je podproblem vec ranije resen, koristi se resenje koje je
64     vec izracunato i */
65     if (f[n] != 0)
66         return f[n];
67
68     /* Izlaz iz rekurzije */
69     if (n == 0 || n == 1)
70         return f[n] = n;
71
72     /* Rekurzivni pozivi */
73     return f[n] =
74         a * F_napredna(n - 1, a, b) + b * F_napredna(n - 2, a, b);
75 }
76
77 int main()
78 {
79     int n, a, b, ind;
80
81     /*Unosi se redni broj funkcije koja ce se primeniti */
82     printf("Unesite redni broj funkcije koju zelite:\n");
83     printf("1 - iterativna\n");
84     printf("2 - rekurzivna\n");

```



```

85     printf("3 - rekurzivna napredna\n");
86     scanf("%d", &ind);
87
88     /* Ucitavanje koeficijenata a i b */
89     printf("Unesite koeficijente:\n");
90     scanf("%d%d", &a, &b);
91
92     /* Ucitavanje prirodnog broja n */
93     printf("Unesite koji clan niza se racuna:\n");
94     scanf("%d", &n);
95
96     /* Testirati program za razlicite vrednosti promenljive n. Na
97        primer za n=20, 30, 40, 50, 55, 60 ... */
98
99     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
100        funkcije F_iterativna, F_rekurzivna ili F_napredna */
101     if(ind == 0)
102         printf("F(%d) = %d\n", n, F_iterativna(n, a, b));
103     else if(ind == 1)
104         printf("F(%d) = %d\n", n, F_rekurzivna(n, a, b));
105     else
106         printf("F(%d) = %d\n", n, F_napredna(n, a, b));
107
108     return 0;
109 }

```

### Rešenje 1.21

```

#include <stdio.h>
2
3 int zbir_cifara(unsigned int x)
4 {
5     /* Izlazak iz rekurzije: ako je broj jednocifren */
6     if (x < 10)
7         return x;
8
9     /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
10        poslednje cifre + poslednja cifra tog broja */
11     return zbir_cifara(x / 10) + x % 10;
12 }
13
14 int main()
15 {
16     unsigned int x;
17
18     /* Ucitava se ceo broj sa ulaza */
19     scanf("%u", &x);
20
21     /* Ispisuje se zbir cifara ucitanog broja */
22     printf("%d\n", zbir_cifara(x));

```

```

24     return 0;
    }

```

### Rešenje 1.22

```

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

4  /*****
   Ako je n<=0, onda je suma(a,0) = 0
6   Ako je n>0, onda je suma(a,n) = a[n-1] + suma(a,n-1)
   Suma celog niza je jednaka sumi prvih n-1 elementa uvecenoj
8   za poslednji element celog niza.
   *****/
10 int sumaNiza(int *a, int n)
11 {
12     if (n <= 0)
13         return 0;
14
15     return a[n - 1] + sumaNiza(a, n - 1);
16 }

18 /*****
   Funkcija napisana na drugi nacin:
20 n<=0, suma(a,0) = 0
   n >0, suma(a,n) = a[0] + suma(a+1,n-1)
22 Suma celog niza je jednaka zbiru prvog elementa niza i sume
   preostalih n-1 elementa.
24 *****/
25 int sumaNiza2(int *a, int n)
26 {
27     if (n <= 0)
28         return 0;
29
30     return a[0] + sumaNiza2(a + 1, n - 1);
31 }

32
33 int main()
34 {
35     int a[MAX_DIM];
36     int n, i = 0, ind;

38     /* Ucitava se redni broj funkcije */
39     printf("Unesite redni broj funkcije (1 ili 2):\n");
40     scanf("%d", &ind);

42     /* Ucitava se broj elemenata niza */
43     printf("Unesite dimenziju niza:");
44     scanf("%d", &n);

46     /* Ucitava se n elemenata niza. */

```

```

printf("Unesite elemente niza:");
48 for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
50
/* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
52 funkcije sumaNiza, odnosno sumaNiza2 */
if(ind == 1)
54     printf("Suma elemenata je %d\n", sumaNiza(a, n));
else
56     printf("Suma elemenata je %d\n", sumaNiza2(a, n));
58 return 0;
}

```

### Rešenje 1.23

```

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

/* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
4 dimenzije n */
int maksimum_niza(int niz[], int n)
6 {
    /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je
8    ujedno i jedini element niza */
    if (n == 1)
10        return niz[0];

    /* Resavanje problema manje dimenzije */
12
14    int max = maksimum_niza(niz, n - 1);

    /* Na osnovu poznatog resenja problema dimenzije n-1, resava se
16    problem dimenzije n */
18    return niz[n - 1] > max ? niz[n - 1] : max;
}

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

    /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
26    Promenljiva i predstavlja indeks tekućeg broja. */
    int i = 0;
    while (scanf("%d", &brojevi[i]) != EOF) {
28        i++;
30    }
    n = i;
32

    /* Stampa se maksimum unetog niza brojeva */
34    printf("%d\n", maksimum_niza(brojevi, n));
}

```

```

36     return 0;
    }

```

### Rešenje 1.24

```

1  #include <stdio.h>
2  #define MAX_DIM 256
3
4  int skalarno(int a[], int b[], int n)
5  {
6      /* Izlazak iz rekurzije */
7      if (n == 0)
8          return 0;
9
10     /******
11     Na osnovu resenja problema dimenzije n-1, resava se problem
12     dimenzije n primenom definicije skalarnog proizvoda
13     a*b = a[0]*b[0] + a[1]*b[1] + ... + a[n-2]*a[n-2] + a[n-1]*a[n-1]
14     Dakle,
15     skalarno(a,b,n) = skalarno(a, b, n - 1) + a[n-1]*a[n-1]
16     *****/
17     else
18         return a[n - 1] * b[n - 1] + skalarno(a, b, n - 1);
19 }
20
21 int main()
22 {
23     int i, a[MAX_DIM], b[MAX_DIM], n;
24
25     /* Unosi se dimenzija nizova. */
26     printf("Unesite dimenziju nizova:");
27     scanf("%d", &n);
28
29     /* A zatim i elementi nizova. */
30     printf("Unesite elemente prvog niza:");
31     for (i = 0; i < n; i++)
32         scanf("%d", &a[i]);
33
34     printf("Unesite elemente drugog niza:");
35     for (i = 0; i < n; i++)
36         scanf("%d", &b[i]);
37
38     /* Ispisuje se rezultat skalarnog proizvoda dva učitana niza. */
39     printf("Skalarni proizvod je %d\n", skalarno(a, b, n));
40
41     return 0;
42 }

```

### Rešenje 1.25

```

#include <stdio.h>
#define MAX_DIM 256

int br_pojave(int x, int a[], int n)
{
    /* Izlazak iz rekurzije: za niz duzine jedan broj pojava broja x
       u nizu je 1 ukoliko je jedini element a[0] bas x ili 0 inace */
    if (n == 1)
        return a[0] == x ? 1 : 0;

    /* U promenljivu bp se smesta broj pojava broja x u prvih n-1
       elemenata niza a. Ukupan broj pojavljivanja broja x u celom nizu
       a je jednak bp uvecanom za jedan ukoliko je se na poziciji n-1 u
       nizu a nalazi broj x */
    int bp = br_pojave(x, a, n - 1);
    return a[n - 1] == x ? 1 + bp : bp;
}

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

    /* Unosi se ceo broj */
    printf("Unesite ceo broj:");
    scanf("%d", &x);

    /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz.
       Promenljiva i predstavlja indeks tekuceg broja */
    printf("Unesite elemente niza:");
    i = 0;
    while (scanf("%d", &a[i]) != EOF) {
        i++;
    }
    n = i;

    /* Ispisuje se broj pojavljivanja */
    printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
    return 0;
}

```

### Rešenje 1.26

```

#include <stdio.h>
#define MAX_DIM 256

int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
{
    /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i
       vraca se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi

```

```

8      niza */
9      if (n < 3)
10         return 0;

12     /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
13        dovoljno je da su oni poslednja tri clana niza ili da se oni
14        rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
15     return ((a[n - 3] == x) && (a[n - 2] == y)
16            && (a[n - 1] == z))
17            || tri_uzastopna_clana(x, y, z, a, n - 1);
18 }

20 int main()
21 {
22     int x, y, z, a[MAX_DIM];
23     int n;

24     /* Ucitavaju se tri cela broja za koje se ispituje da li su
25        uzastopni clanovi niza */
26     printf("Unesite tri cela broja:");
27     scanf("%d%d%d", &x, &y, &z);

30     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
31        Promenljiva i predstavlja indeks tekuceg broja. */
32     printf("Unesite elemente niza:");
33     int i = 0;
34     while (scanf("%d", &a[i]) != EOF) {
35         i++;
36     }
37     n = i;

38     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana ispisuje
39        se odgovarajuca poruka */
40     if (tri_uzastopna_clana(x, y, z, a, n))
41         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
42     else
43         printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
44

46     return 0;
47 }

```

### Rešenje 1.27

```

1  #include <stdio.h>
2
3  /*****
4   Funkcija koja broji bitove svog argumenta
5
6   ako je x ==0, onda je count(x) = 0
7   inace count(x) = najvisi_bit +count(x<<1)
8
9   *****/

```

```

10      Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
      siftovanjem sa desne strane uvek dopisuju 0, argument x ce u
      nekom rekurzivnom pozivu biti bas 0 i izacicemo iz rekurzije.
12  *****/
      int count(int x)
14  {
      /* Izlaz iz rekurzije */
16      if (x == 0)
          return 0;
18
      /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x je
20      postavljen na 1. Koriscenjem odgovarajuce maske proverava se
      vrednost najviseg bita. Rezultat koliko ima jedinica u ostatku
22      binarnog zapisa broja x se uvecava za 1. Najvisi bit je 0. Stoga
      je broj jedinica u zapisu x isti kao broj jedinica u zapisu
24      broja x<<1, jer se siftovanjem u levo sa desne strane dopisuju 0.
      Za rekurzivni poziv se salje vrednost koja se dobija kada se x
26      siftuje u levo. Napomena: argument funkcije x je oznacen ceo
      broj, usled cega se ne koristi siftovanje udesno, jer funkciji
28      moze biti prosleden i negativan broj. Iz tog razloga, odlucujemo
      se da proveramo najvisi, umesto najnizeg bita */
30      if (x & (1 << (sizeof(x) * 8 - 1)))
          return 1 + count(x << 1);
32      else
          return count(x << 1);
34      /******
      Krace napisano
36      return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1);
      *****/
38  }

40  int main()
42  {
      int x;
      /* Sa standardnog ulaza se ucitava ceo broj */
44      scanf("%x", &x);
      /* Na standardni izlaz se ispisuje rezultat poziva funkcije count
46      za uneti broj */
      printf("%d\n", count(x));
48
      return 0;
50  }

```

### Rešenje 1.29

```

      #include <stdio.h>
2
      /* Rekurzivna funkcija za odredjivanje najvece oktalne cifre u
      broju */
4
      int max_oktalna_cifra(unsigned x)
6  {

```

```

8      /* Izlazak iz rekurzije */
      if (x == 0)
          return 0;
10     /* Odredjivanje poslednje oktalne cifre u broju */
      int poslednja_cifra = x & 7;
12     /* 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;

24     /* Ucitava se neoznaceni ceo broj */
      scanf("%u", &x);

26     /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
28     printf("%d\n", max_oktalna_cifra(x));
      return 0;
30 }

```

### Rešenje 1.30

```

2      #include <stdio.h>

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

20 int main()
21 {
22     unsigned x;

```



```

24  /* Ucitava se neoznaceni broj */
    scanf("%u", &x);
26
    /* Ispisuje se vrednost najveće heksadekadne cifre unetog broja */
28  printf("%d\n", max_heksadekadna_cifra(x));
    return 0;
30 }

```

### Rešenje 1.31

```

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

6  int palindrom(char s[], int n)
{
8     /* Izlaz iz rekurzije - trivijalno, niska dužine 0 ili 1 je
        palindrom */
10    if ((n == 1) || (n == 0))
        return 1;
12    /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
        poslednji karakter i da je palindrom niska koja nastaje kada se
14    polaznoj nisci otklone prvi i poslednji karakter */
    return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
16 }

18 int main()
{
20     char s[MAX_DIM];
    int n;

22     /* Ucitavanje niske sa standardnog ulaza */
24     scanf("%s", s);

26     /* Odredjuje se dužina niske */
    n = strlen(s);

28     /* Ispisuje se poruka da li je niska palindrom ili nije */
30     if (palindrom(s, n))
        printf("da\n");
32     else
        printf("ne\n");
34
    return 0;
36 }

```

### Rešenje 1.32

```
#include <stdio.h>
2 #include <stdlib.h>
#define MAX_DUZINA_NIZA 50

4 void ispisiNiz(int a[], int n)
6 {
    int i;

8     for (i = 1; i <= n; i++)
10         printf("%d ", a[i]);
    printf("\n");
12 }

14 /* Funkcija proverava da li se x vec nalazi u permutaciji na
    prethodnih 1...n mesta */
16 int koriscen(int a[], int n, int x)
18 {
    int i;
    for (i = 1; i <= n; i++)
20         if (a[i] == x)
            return 1;

22     return 0;
24 }

26 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[] je niz
    u koji smesta permutacije m - oznacava da se na m-tu poziciju u
28 permutaciji smesta jedan od preostalih celih brojeva n- je
    velicina skupa koji se permutuje Funkciju se poziva sa argumentom
30 m=1 jer formiranje permutacije pocinje od 1. pozicije. Stoga, nece
    se koristiti a[0]. */
32 void permutacija(int a[], int m, int n)
34 {
    int i;

36     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
        premasila velicinu skupa, onda se svi brojevi vec nalaze u
38 permutaciji i ispisuje se permutacija. */
    if (m > n) {
40         ispisiNiz(a, n);
        return;
42     }

44     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
        mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
46 Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
        ovako postavljenom pocetku permutacije. Kada se to zavrsi, vrsi
48 se provera da li postoji jos neki broj koji moze da se stavi na
        m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
50 funkcija zavrшава sa radom. Ukoliko takav broj postoji, onda se
        ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
```

```

52     ali sada sa drugacije postavljenim prefiksom. */
54
55     for (i = 1; i <= n; i++) {
56         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
57            pozicije, onda se on postavlja na poziciju m i poziva se
58            funkcija da napravi permutaciju za jedan vecu duzine, tj. m+1.
59            Inace, nastavlja se dalje, trazeci broj koji se nije pojavio
60            do sada u permutaciji. */
61         if (!koriscen(a, m - 1, i)) {
62             a[m] = i;
63             /* Poziva se ponovo funkcija da dopuni ostatak permutacije
64                posle upisivanja i na poziciju m. */
65             permutacija(a, m + 1, n);
66         }
67     }
68 }
69
70 int main(void)
71 {
72     int n;
73     int a[MAX_DUZINA_NIZA];
74
75     /* Ucitavanje broja n i provera da li je u odgovarajucem opsegu */
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
85     permutacija(a, 1, n);
86
87     exit(EXIT_SUCCESS);
88 }

```

### Rešenje 1.33

```

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

```

```

12         binomniKoeficijent(n - 1, k) : 1;
13     }
14
15     /*****
16     Iterativno izracunavanje datog binomnog koeficijenta.
17
18     int binomniKoeficijent (int n, int k) {
19         int i, j, b;
20         for (b=i=1, j=n; i<=k; b =b * j-- / i++)
21             ;
22         return b;
23     }
24
25     Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
26     resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
27     *****/
28
29     /* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
30     zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
31     ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
32     veca od sume elemenata prethodne hipotenuze. */
33     int sumaElemenataHipotenuze(int n)
34     {
35         return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
36     }
37
38     int main()
39     {
40         int n, k, i, d, r;
41
42         /* Ucitavanje brojeva d i r */
43         scanf("%d %d", &d, &r);
44
45         /* Ispisivanje Paskalovog trougla */
46         putchar('\n');
47         for (n = 0; n <= d; n++) {
48             for (i = 0; i < d - n; i++)
49                 printf(" ");
50             for (k = 0; k <= n; k++)
51                 printf("%4d", binomniKoeficijent(n, k));
52             putchar('\n');
53         }
54
55         /* Provera da li je r nenegativan */
56         if (r < 0) {
57             fprintf(stderr,
58                 "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
59             );
60             exit(EXIT_FAILURE);
61         }
62
63         /* Ispisivanje sume elemenata hipotenuze */

```

```
64     printf("%d\n", sumaElemenataHipotenuze(r));  
66     exit(EXIT_SUCCESS);  
    }
```

## 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 SA PROGRAMOM:  
|| 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 SA PROGRAMOM:  
|| Unesite dimenziju niza: 0  
|| Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.1]

**Zadatak 2.2** Dat je niz realnih brojeva dimenzije  $n$ . Korišćenjem pokazivačke sintakse, napisati:

- (a) funkciju `zbir` koja izračunava zbir elemenata niza,
- (b) funkciju `proizvod` koja izračunava proizvod elemenata niza,
- (c) funkciju `min_element` koja izračunava najmanji element niza,
- (d) 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 SA PROGRAMOM:
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 SA PROGRAMOM:
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 SA PROGRAMOM:
Unesite dimenziju niza: 5
Unesite elemente niza:
1 2 3 4 5
Transformisan niz je:
2 3 3 3 4
```

### Primer 2

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

### Primer 3

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju niza: 0
Greska: neodgovarajuca dimenzija niza.
```

### Primer 4

```
INTERAKCIJA SA PROGRAMOM:
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 argumenate 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 koje od ova dva rešenja treba koristiti prilikom ispisa.

### Primer 1

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

INTERAKCIJA SA PROGRAMOM:
Broj 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:
. p 2 t -
```

### Primer 2

```
Poziv: ./a.out

INTERAKCIJA SA PROGRAMOM:
Broj 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:
.
```

[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.

### Primer 1

```
Poziv: ./a.out a b 11 212

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

### Primer 2

```
Poziv: ./a.out

INTERAKCIJA SA PROGRAMOM:
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

ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj ima
reci koje imaju 1 karakter

INTERAKCIJA SA PROGRAMOM:
Broj reci ciji je broj karaktera 1 je 3.
```

### Primer 2

```
POZIV: ./a.out ulaz.txt

ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj ima
reci koje imaju 1 karakter

INTERAKCIJA SA PROGRAMOM:
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 SA PROGRAMOM:
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

ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj ima reci
koje se završavaju na ke

INTERAKCIJA SA PROGRAMOM:
Broj reci koje se završavaju na ke je 2.
```

### Primer 2

```
POZIV: ./a.out ulaz.txt sa -p

ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj ima reci
koje pocinju sa sa

INTERAKCIJA SA PROGRAMOM:
Broj reci koje pocinju na sa je 3.
```

*Primer 3*

```

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

```

*Primer 4*

```

Poziv: ./a.out ulaz.txt
ULAZ.TXT
Ovo je sadrzaj ulaza.
INTERAKCIJA SA PROGRAMOM:
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. Na standardni izlaz ispisati učitane matricu, a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 -2 3
4 -5 6
7 -8 9
Trag matrice je 5.
Euklidska norma matrice je 16.88.
Vandijagonalna norma matrice je 11.

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 0
Greska: neodgovarajuca dimenzija matrice.

```

[Rešenje 2.8]

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

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

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

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrica: 3
Unesite elemente prve matrice, vrstu po vrstu:
1 2 3
1 2 3
1 2 3
Unesite elemente druge matrice, vrstu po vrstu:
1 2 3
1 2 3
1 2 3
Matrice su jednake.
Zbir matrica je:
2 4 6
2 4 6
2 4 6
Proizvod matrica je:
6 12 8
6 12 8
6 12 8
```

[Rešenje 2.9]

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

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

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

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

#### Primer 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

INTERAKCIJA SA PROGRAMOM:
Relacija nije refleksivna.
Relacija nije simetricna.
Relacija jeste tranzitivna.
Refleksivno zatvorenje relacije:
1 0 0 0
0 1 1 0
0 0 1 0
0 0 0 1
Simetricno zatvorenje relacije:
1 0 0 0
0 1 1 0
0 1 1 0
0 0 0 0
Refleksivno-tranzitivno zatvorenje relacije:
1 0 0 0
0 1 1 0
0 0 1 0
0 0 0 1
```

[Rešenje 2.10]

**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 rezultat primene prethodno napisanih funkcija.

*Primer 1*

```

Poziv: ./a.out 3

INTERAKCIJA SA PROGRAMOM:
Unesite elemente matrice dimenzije 3:
1 2 3
-4 -5 -6
7 8 9
Najveci element sporedne dijagonale je 7.
Indeks kolone sa najmanjim elementom je 2.
Indeks vrste sa najvećim elementom je 2.
Broj negativnih elemenata matrice je 3.

```

*Primer 2*

```

Poziv: ./a.out 4

INTERAKCIJA SA PROGRAMOM:
Unesite elemente matrice dimenzije 4:
-1 -2 -3 -4
-5 -6 -7 -8
-9 -10 -11 -12
-13 -14 -15 -16
Najveci element sporedne dijagonale je -4.
Indeks kolone sa najmanjim elementom je 3.
Indeks vrste sa najvećim elementom je 0.
Broj negativnih elemenata matrice je 16.

```

[Rešenje 2.11]

**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.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 4
Unesite elemente matrice, vrstu po vrstu:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
Matrica je ortonormirana.

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 2 3
5 6 7
1 4 2
Matrica nije ortonormirana.

```

[Rešenje 2.12]

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

- (a) Napisati funkciju koja učitava elemente matrice sa standardnog ulaza
- (b) Napisati 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. Na standardni izlaz spiralno ispisati elemente učitane matrice.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona matrice:
3 3
Unesite elemente matrice, vrstu po vrstu:
1 2 3
4 5 6
7 8 9
Spiralno ispisana matrica:
1 2 3 6 9 8 7 4 5
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona matrice:
3 4
Unesite elemente matrice, vrstu po vrstu:
1 2 3 4
5 6 7 8
9 10 11 12
Spiralno ispisana matrica:
1 2 3 4 8 12 11 10 9 5 6 7
```

[Rešenje 2.13]

**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.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju kvadratne matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 2 3
4 5 6
7 8 9
Unesite stepen koji se racuna: 8
8. stepen matrice je:
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.

### Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:
| Unesite dimenziju niza: 3
| Unesite elemente niza:
| 1 -2 3
| Niz u obrnutom poretku je: 3 -2 1
```

### Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:
| Unesite dimenziju niza: -1
| malloc(): neuspela alokacija memorije.
```

[Rešenje 2.15]

**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.

Od korisnika sa ulaza tražiti da izabere način realokacije memorije.

### Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:
| Unesite zeljeni nacin realokacije (M ili R):
| M
| Unesite brojeve, nulu za kraj:
| 1 -2 3 -4 0
| Niz u obrnutom poretku je: -4 3 -2 1
```

### Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:
| Unesite zeljeni nacin realokacije (M ili R):
| R
| Unesite brojeve, nulu za kraj:
| 6 -1 5 -2 4 -3 0
| Niz u obrnutom poretku je: -3 4 -2 5 -1 6
```

[Rešenje 2.16]

**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 50 karaktera i da ne sadrže praznine). Na standardni izlaz ispisati nisku koja se dobija njihovim nadovezivanjem. Za rezultujuću nisku dinamički alocirati memoriju.

### Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:
| Unesite dve niske karaktera:
| Jedan Dva
| Nadovezane niske: JedanDva
```

### Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:
| Unesite dve niske karaktera:
| Ana Marija
| Nadovezane niske: AnaMarija
```

[Rešenje 2.17]

**Zadatak 2.18** Napisati program koji sa standardnog ulaza učitava matricu realnih 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.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona matrice:
2 3
Unesite elemente matrice, vrstu po vrstu:
1.2 2.3 3.4
4.5 5.6 6.7
Trag unete matrice je 6.80.
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona matrice:
2 2
Unesite elemente matrice, vrstu po vrstu:
-0.1 -0.2
-0.3 -0.4
Trag unete matrice je -0.50.
```

[Rešenje 2.18]

**Zadatak 2.19** Napisati biblioteku za rad sa celobrojnim matricama.

- (a) Napisati funkciju `int **alociraj_matricu(int n, int m)` koja dinamički alokira memoriju potrebnu za matricu dimenzija  $n \times m$ .
- (b) Napisati funkciju `int **alociraj_kvadratnu_matricu(int n)` koja alokira memoriju za kvadratnu matricu dimenzije  $n$ .
- (c) Napisati funkciju `int **deallociraj_matricu(int **A, int n)` koja dealocira memoriju matrice sa  $n$  vrsta. Povratna vrednost ove funkcije treba da bude "prazna" matrica.
- (d) Napisati funkciju `void ucitaj_matricu(int **A, int n, int m)` koja učitava već alociranu matricu dimenzija  $n \times m$  sa standardnog ulaza.
- (e) Napisati funkciju `void ucitaj_kvadratnu_matricu(int **A, int n)` koja učitava već alociranu kvadratnu matricu dimenzije  $n$  sa standardnog ulaza.
- (f) Napisati funkciju `void ispisi_matricu(int **A, int n, int m)` koja ispisuje matricu dimenzija  $n \times m$  na standardnom izlazu.
- (g) Napisati funkciju `void ispisi_kvadratnu_matricu(int **A, int n)` koja ispisuje kvadratnu matricu dimenzije  $n$  na standardnom izlazu.
- (h) Napisati funkciju `int ucitaj_matricu_iz_datoteke(int **A, int n, int m, FILE * f)` koja učitava već alociranu matricu dimenzija  $n \times m$  iz već otvorene datoteke  $f$ . U slučaju neuspešnog učitavanja vratiti vrednost različitu od 0.



- (i) Napisati funkciju `int ucitaj_kvadratnu_matricu_iz_datoteke(int **A, int n, FILE * f)` koja učitava već alociranu kvadratnu matricu dimenzije  $n$  iz već otvorene datoteke  $f$ . U slučaju neuspješnog učitavanja vratiti vrednost različitu od 0.
- (j) Napisati funkciju `int upisi_matricu_u_datoteku(int **A, int n, int m, FILE * f)` koja upisuje matricu dimenzija  $n \times m$  u već otvorenu datoteku  $f$ . U slučaju neuspješnog upisivanja vratiti vrednost različitu od 0.
- (k) Napisati funkciju `int upisi_kvadratnu_matricu_u_datoteku(int **A, int n, FILE * f)` koja upisuje kvadratnu matricu dimenzije  $n$  u već otvorenu datoteku  $f$ . U slučaju neuspješnog upisivanja vratiti vrednost različitu od 0.

Napisati programe koji testiraju napisanu biblioteku.

- (1) Program učitava dimenzije nekvadratne matrice sa standardnog ulaza, a zatim i samu matricu. Potom, matricu upisati u datoteku *matrica.txt*.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesi broj vrsta matrice: 3
Unesi broj kolona matrice: 4
Unesi elemente matrice po vrstama:
1 2 3 4
5 6 7 8
9 10 11 12

MATRICA.TXT
1 2 3 4
5 6 7 8
9 10 11 12
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesi broj vrsta matrice: 5
Unesi broj kolona matrice: 0
Neodgovarajce dimenzije matrice
```

- (2) Program prima kao prvi argument komandne linije putanju do datoteke u kojoj se redom nalazi dimenzija kvadratne matrice i sama matrica, koju treba ispisati na standardnom izlazu.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> Poziv: ./a.out ulaz.txt  ULAZ.TXT 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  IzLAZ: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 </pre>	<pre> Poziv: ./a.out ulaz.txt  ULAZ.TXT dimenzija: 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  IzLAZ: Neispravan pocetak fajla </pre>	<pre> Poziv: ./a.out  IzLAZ: Koriscenje programa: ./a.out datoteka </pre>

[Rešenje 2.19]

**Zadatak 2.20** Data je celobrojna matrica dimenzije  $n \times m$ . Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu). Napisati program koji testira napisanu funkciju. 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. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

### *Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona matrice:
2 3
Unesite elemente matrice, vrstu po vrstu:
1 -2 3
-4 5 -6
Elementi ispod glavne dijagonale matrice:
1
-4 5

```

[Rešenje 2.20]

**Zadatak 2.21** Za zadatak 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. Ukoliko ima više takvih, ispisati prvu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona matrice:
2 3
Unesite elemente matrice, vrstu po vrstu:
1 2 3
4 5 6
Kolona pod rednim brojem 3 ima najveći zbir.
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona matrice:
2 4
Unesite elemente matrice, vrstu po vrstu:
1 2 3 4
8 7 6 5
Kolona pod rednim brojem 1 ima najveći zbir.
```

**Zadatak 2.22** Data je realna kvadratna matrica dimenzije  $n$ .

- (a) Napisati funkciju koja izračunava zbir apsolutnih vrednosti matrice ispod sporedne dijagonale.
- (b) 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.

### Primer 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
```

```
INTERAKCIJA SA PROGRAMOM:
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
```

[Rešenje 2.22]

**Zadatak 2.23** Napisati program koji na osnovu dve realne 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.

### Primer 1

```
POZIV: ./a.out matrice.txt
```

```
MATRICE.TXT
3
1.1 -2.2 3.3
-4.4 5.5 -6.6
7.7 -8.8 9.9
-1.1 2.2 -3.3
4.4 -5.5 6.6
-7.7 8.8 -9.9
```

```
INTERAKCIJA SA PROGRAMOM:
```

```
1.1 -2.2 3.3
-1.1 2.2 -3.3
-4.4 5.5 -6.6
4.4 -5.5 6.6
7.7 -8.8 9.9
-7.7 8.8 -9.9
```

**Zadatak 2.24** Na ulazu se zadaje niz celih brojeva čiji se unos završava nulom. Napisati funkciju koja od zadatog niza formira matricu tako da prva vrsta odgovara unetom nizu, a svaka naredna se dobija cikličkim pomeranjem elemenata niza za jednu poziciju ulevo. Napisati program koji testira ovu funkciju. Rezultujuću matricu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
```

```
Unesite elemente niza, nulu za kraj:
```

```
1 2 3 0
```

```
Trazena matrica je:
```

```
1 2 3
```

```
2 3 1
```

```
3 1 2
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
```

```
Unesite elemente niza, nulu za kraj:
```

```
-5 -2 -4 -1 0
```

```
Trazena matrica je:
```

```
-5 -2 -4 -1
```

```
-2 -4 -1 -5
```

```
-4 -1 -5 -2
```

```
-1 -5 -2 -4
```

**Zadatak 2.25** 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.*

### Primer 1

```
SLICICE.TXT
```

```
3 Brazil
```

```
6 Nemacka
```

```
2 Kamerun
```

```
1 Brazil
```

```
2 Engleska
```

```
4 Engleska
```

```
5 Brazil
```

```
INTERAKCIJA SA PROGRAMOM:
```

```
Petru ukupno nedostaje 7 sličica.
```

```
Reprezentacija za koju je sakupio
```

```
najmanji broj sličica je Brazil.
```

**\*\* Zadatak 2.26** 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.

### Primer 1

```
TEMENA.TXT
-1 -1
1 -1
1 1
-1 1
```

```
INTERAKCIJA SA PROGRAMOM:
U datoteci su zadata temena cetvorougla.
Obim je 8.
Povrsina je 4.
```

### Primer 2

```
TEMENA.TXT
-1.75 -1.5
3 1.5
2.2 3.1
-2 4
-4.1 1
```

```
INTERAKCIJA SA PROGRAMOM:
U datoteci su zadata temena petougla.
Obim je 18.80.
Povrsina je 22.59.
```

## 2.4 Pokazivači na funkcije

**Zadatak 2.27** 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`).

### Primer 1

```
Poziv: ./a.out sin

INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala:
-0.5 1
Koliko tacaka ima na ekvidistantnoj
mrezi (ukljucujuci krajeve intervala)?
4
x sin(x)
-----
| -0.50000 | -0.47943 |
| 0.00000 | 0.00000 |
| 0.50000 | 0.47943 |
| 1.00000 | 0.84147 |
-----
```

### Primer 2

```
Poziv: ./a.out cos

INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala:
0 2
Koliko tacaka ima na ekvidistantnoj
mrezi (ukljucujuci krajeve intervala)?
4
x cos(x)
-----
| 0.00000 | 1.00000 |
| 0.66667 | 0.78589 |
| 1.33333 | 0.23524 |
| 2.00000 | -0.41615 |
-----
```

[Rešenje 2.27]

**Zadatak 2.28** Napisati funkciju koja izračunava limes funkcije  $f(x)$  u tački  $a$ . Adresa funkcije  $f$  čiji se limes računa se prenosi kao parametar funkciji za računanje limesa. Limes se računa sledećom aproksimacijom (vrednosti  $n$  i  $a$  uneti sa standardnog ulaza kao i ime funkcije):

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

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ime funkcije, n i a:
|| tan 10000 1.570795
|| Limes funkcije tan je -10134.46.
```

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ime funkcije, n i a:
|| cos 5000 0.25
|| Limes funkcije cos je 0.97.
```

**Zadatak 2.29** 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.

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ime funkcije, n, a i b:
|| cos 6000 -1.5 3.5
|| Vrednost integrala je 0.645931.
```

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ime funkcije, n, a i b:
|| sin 10000 -5.2 2.1
|| Vrednost integrala je 0.973993.
```

**Zadatak 2.30** 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 i  $n$ , a na standardni izlaz ispisuje vrednost odgovarajućeg integrala.

## Primer 1

```

INTERAKCIJA SA PROGRAMOM:
Unesite ime funkcije, n, a i b:
sin 100 -1.0 3.0
Vrednost integrala je 1.530295.

```

## Primer 2

```

INTERAKCIJA SA PROGRAMOM:
Unesite ime funkcije, n, a i b:
tan 5000 -4.1 -2.3
Vrednost integrala je -0.147640.

```

## 2.5 Rešenja

## Rešenje 2.1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 100
5
6  /* Funkcija obrce elemente niza koriscenjem indekse sintakse */
7  void obrni_niz_v1(int a[], int n)
8  {
9      int i, j;
10
11     for (i = 0, j = n - 1; i < j; i++, j--) {
12         int t = a[i];
13         a[i] = a[j];
14         a[j] = t;
15     }
16 }
17
18 /* Funkcija obrce elemente niza koriscenjem pokazivacke sintakse */
19 void obrni_niz_v2(int *a, int n)
20 {
21     /* Pokazivaci na elemente niza */
22     int *prvi, *poslednji;
23
24     /* Vrsi se obrtanje niza */
25     for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
26         int t = *prvi;
27
28         /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
29          vrednost koja se nalazi na adresi na koju pokazuje pokazivac
30          "poslednji". Nakon toga se pokazivac "prvi" uvecava za jedan
31          sto za posledicu ima da "prvi" pokazuje na sledeci element u
32          nizu */
33         *prvi++ = *poslednji;
34
35         /* Vrednost promenljive "t" se postavlja na adresu na koju
36          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 nizu
    */
39  *poslednji-- = t;
    }

41
42  /******
43  Drugi nacin za obrtanje niza

44
45  for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
46      prvi++, poslednji--) {
47      int t = *prvi;
48      *prvi = *poslednji;
49      *poslednji = t;
50  }
51  /******
52  */
53  }

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

58
59      /* Broj elemenata niza a */
60      int n;

61
62      /* Pokazivac na elemente niza */
63      int *p;

64
65      printf("Unesite dimenziju niza: ");
66      scanf("%d", &n);

67
68      /* Proverava se da li je doslo do prekoracenja ogranicenja
69      dimenzije */
70      if (n <= 0 || n > MAX) {
71          fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72          exit(EXIT_FAILURE);
73      }

74
75      printf("Unesite elemente niza:\n");
76      for (p = a; p - a < n; p++)
77          scanf("%d", p);

78
79      obrni_niz_v1(a, n);

80
81      printf("Nakon obrtanja elemenata, niz je:\n");

82
83      for (p = a; p - a < n; p++)
84          printf("%d ", *p);
85      printf("\n");

86
87      obrni_niz_v2(a, n);

```



```
89 printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
91 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 */
double proizvod(double *a, int n)
18 {
20    double p = 1;

22    for (; n; n--)
        p *= *(a + n - 1));

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;
    int i;

34    /* Ispituje se da li se medju ostalim elementima niza nalazi
36    minimalni */
    for (i = 1; i < n; i++)
38        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 prekoracenja ogranicenja
        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 %.3f.\n", zbir(a, n));
    printf("Proizvod elemenata niza je %.3f.\n", proizvod(a, n));
82     printf("Minimalni element niza je %.3f.\n", min(a, n));
    printf("Maksimalni element niza je %.3f.\n", max(a, n));
84
    return 0;
86 }
```

### Rešenje 2.3

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX 100
4
5 /* Funkcija povecava za jedan sve elemente u prvoj polovini niza a
6    smanjuje za jedan sve elemente u drugoj polovini niza. Ukoliko niz
7    ima neparan broj elemenata, srednji element ostaje nepromenjen */
8 void povecaj_smanji(int *a, int n)
9 {
10     int *prvi = a;
11     int *poslednji = a + n - 1;
12
13     while (prvi < poslednji) {
14
15         /* Povecava se vrednost elementa na koji pokazuje pokazivac prvi
16            */
17         (*prvi)++;
18
19         /* Pokazivac prvi se pomera na sledeci element */
20         prvi++;
21
22         /* Smanjuje se vrednost elementa na koji pokazuje pokazivac
23            poslednji */
24         (*poslednji)--;
25
26         /* Pokazivac poslednji se pomera na prethodni element */
27         poslednji--;
28     }
29
30     /******
31     Drugi nacin:
32     while (prvi < poslednji) {
33         (*prvi++)++;
34         (*poslednji--)--;
35     }
36     *****/
37 }
38
39 int main()
40 {
41     int a[MAX];
42     int n;
43     int *p;
44
45     printf("Unesite dimenziju niza: ");
46     scanf("%d", &n);
47
48     /* Proverava se da li je doslo do prekoracenja ogranicenja
49        dimenzije */
50     if (n <= 0 || n > MAX) {
51         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
52         exit(EXIT_FAILURE);
53     }
```

```

    }

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

57     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

```

1  #include <stdio.h>

3  int main(int argc, char *argv[])
4  {
5      int i;
6      char tip_ispisa;

7

8      printf("Broj argumenata komandne linije je %d.\n", argc);

9

10     printf("Kako zelite da ispisete argumente, koriscenjem"
11            " indeksne ili pokazivacke sintakse (I ili P)? ");
12     scanf("%c", &tip_ispisa);

13

14     printf("Argumenti komandne linije su:\n");
15     if (tip_ispisa == 'I') {
16         /* Ispisuju se argumenti komandne linije koriscenjem indeksne
17            sintakse */
18         for (i = 0; i < argc; i++)
19             printf("%d %s\n", i, argv[i]);
20     } else if (tip_ispisa == 'P') {
21         /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
22            sintakse */
23         i = argc;
24         for (; argc > 0; argc--)
25             printf("%d %s\n", i - argc, *argv++);

26         /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje na
27            polje u memoriji koje se nalazi iza poslednjeg argumenta
28            komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
29            broja argumenta komandne linije to sada moze ponovo da se
30            postavi "argv" da pokazuje na nulti argument komandne linije
31            */
32         argv = argv - i;

```

```

33     argc = i;
34 }
35
36 printf("Pocetna slova argumenata komandne linije:\n");
37 if (tip_ispisa == 'I') {
38     /* koristeći indeksnu sintaksu */
39     for (i = 0; i < argc; i++)
40         printf("%c ", argv[i][0]);
41     printf("\n");
42 } else if (tip_ispisa == 'P') {
43     /* koristeći pokazivačku sintaksu */
44     for (i = 0; i < argc; i++)
45         printf("%c ", **argv++);
46     printf("\n");
47 }
48
49 return 0;
50 }

```

## Rešenje 2.5

```

1  #include <stdio.h>
2  #include <string.h>
3  #define MAX 100
4
5  /* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
6     cita sprede i odpozadi */
7  int palindrom(char *niska)
8  {
9      int i, j;
10     for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
11         if (*(niska + i) != *(niska + j))
12             return 0;
13     return 1;
14 }
15
16 int main(int argc, char **argv)
17 {
18     int i, n = 0;
19
20     /* Multi argument komandne linije je ime izvrsnog programa */
21     for (i = 1; i < argc; i++)
22         if (palindrom(*(argv + i)))
23             n++;
24
25     printf
26         ("Broj argumenata komandne linije koji su palindromi je %d.\n",
27         n);
28     return 0;
29 }

```

## Rešenje 2.6

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Implementacija funkcije strlen() iz standardne biblioteke */
7  int duzina(char *s)
8  {
9      int i;
10     for (i = 0; *(s + i); i++);
11     return i;
12 }
13
14 int main(int argc, char **argv)
15 {
16     char rec[MAX_KARAKTERA];
17     int br = 0, n;
18     FILE *in;
19
20     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska */
21     if (argc < 3) {
22         printf("Greska: ");
23         printf("Nedovoljan broj argumenata komandne linije.\n");
24         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
25             argv[0]);
26         exit(EXIT_FAILURE);
27     }
28
29     /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
30        komandne linije. */
31     in = fopen(*(argv + 1), "r");
32     if (in == NULL) {
33         fprintf(stderr, "Greska: ");
34         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
35         exit(EXIT_FAILURE);
36     }
37
38     n = atoi(*(argv + 2));
39
40     /* Broje se reci cija je duzina jednaka broju zadatom drugim
41        argumentom komandne linije */
42     while (fscanf(in, "%s", rec) != EOF)
43         if (duzina(rec) == n)
44             br++;
45
46     printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);
47
48     /* Zatvara se datoteka */
49     fclose(in);
50 }
```

```
50     return 0;
    }
```

## Rešenje 2.7

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

#define MAX_KARAKTERA 100

/* Implementacija funkcije strcpy() iz standardne biblioteke */
void kopiranje_niske(char *dest, char *src)
{
    int i;
    for (i = 0; *(src + i); i++)
        *(dest + i) = *(src + i);
}

/* Implementacija funkcije strcmp() iz standardne biblioteke */
int poredjenje_niski(char *s, char *t)
{
    int i;
    for (i = 0; *(s + i) == *(t + i); i++)
        if (*(s + i) == '\0')
            return 0;
    return *(s + i) - *(t + i);
}

/* Implementacija funkcije strlen() iz standardne biblioteke */
int duzina_niske(char *s)
{
    int i;
    for (i = 0; *(s + i); i++);
    return i;
}

/* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
   sufiks niske zadate prvi argumentom funkcije */
int sufiks_niske(char *niska, char *sufiks)
{
    int duzina_sufiksa = duzina_niske(sufiks);
    int duzina_niske_pom = duzina_niske(niska);
    if (duzina_sufiksa <= duzina_niske_pom &&
        poredjenje_niski(niska + duzina_niske_pom -
                        duzina_sufiksa, sufiks) == 0)
        return 1;
    return 0;
}

/* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
   prefiks niske zadate prvi argumentom funkcije */
```

```
int prefiks_niske(char *niska, char *prefiks)
{
    int i;
    int duzina_prefiksa = duzina_niske(prefiks);
    int duzina_niske_pom = duzina_niske(niska);
    if (duzina_prefiksa <= duzina_niske_pom) {
        for (i = 0; i < duzina_prefiksa; i++)
            if (*(prefiks + i) != *(niska + i))
                return 0;
        return 1;
    } else
        return 0;
}

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

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

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

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



```
98     fclose(in);  
    return 0;  
100 }
```

## Rešenje 2.8

```
1  #include <stdio.h>  
  #include <math.h>  
3  #include <stdlib.h>  
  
5  #define MAX 100  
  
7  /* Funkcija izracunava trag matrice */  
  int trag(int M[][MAX], int n)  
9  {  
    int trag = 0, i;  
11   for (i = 0; i < n; i++)  
        trag += M[i][i];  
13   return trag;  
  }  
  
15  
  /* Funkcija izracunava euklidsku normu matrice */  
17  double euklidska_norma(int M[][MAX], int n)  
  {  
19     double norma = 0.0;  
    int i, j;  
  
21     for (i = 0; i < n; i++)  
23         for (j = 0; j < n; j++)  
            norma += M[i][j] * M[i][j];  
  
25     return sqrt(norma);  
27  }  
  
29  /* Funkcija izracunava gornju vandijagonalnu normu matrice */  
  int gornja_vandijagonalna_norma(int M[][MAX], int n)  
31  {  
    int norma = 0;  
    int i, j;  
  
35     for (i = 0; i < n; i++) {  
        for (j = i + 1; j < n; j++)  
37            norma += abs(M[i][j]);  
    }  
  
39     return norma;  
41  }  
  
43  int main()  
  {  
45     int A[MAX][MAX];
```

```

47     int i, j, n;

49     printf("Unesite dimenziju matrice: ");
    scanf("%d", &n);

51     /* Provera prekoracenja dimenzije matrice */
    if (n > MAX || n <= 0) {
53         fprintf(stderr, "Greska: neodgovarajuca dimenzija matrice.\n");
        exit(EXIT_FAILURE);
55     }

57     printf("Unesite elemente matrice, vrstu po vrstu:\n ");
    for (i = 0; i < n; i++)
59         for (j = 0; j < n; j++)
            scanf("%d", &A[i][j]);

61     /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
    for (i = 0; i < n; i++) {
63         /* Ispis elemenata i-te vrste */
        for (j = 0; j < n; j++)
65             printf("%d ", A[i][j]);
        printf("\n");
67     }

69     /*
71     Ispisuju se elementi matrice koriscenjem pokazivacke sintakse.
    Kod ovako definisane matrice, elementi su uzastopno smesteni u
73     memoriju, kao na traci. To znaci da su svi elementi prve vrste
    redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
75     prve vrste smesten je prvi element druge vrste za kojim slede
    svi elementi te vrste i tako dalje redom.

77     for( i = 0; i < n ; i++) {
79         for ( j=0 ; j<n ; j++)
            printf("%d ", (*(A+i)+j));
81         printf("\n");
        }

83     *****/

85     /* Ispisuje se rezultat na standardni izlaz */
    int tr = trag(A, n);
87     printf("Trag matrice je %d.\n", tr);

89     printf("Euklidska norma matrice je %.2f.\n", euklidska_norma(A, n))
    ;
    printf("Vandijagonalna norma matrice je = %d.\n",
91         gornja_vandijagonalna_norma(A, n));

93     return 0;
}

```

## Rešenje 2.9

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 100
5
   /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
   standardnog ulaza */
7  void ucitaj_matricu(int m[][MAX], int n)
9  {
   int i, j;
11
   for (i = 0; i < n; i++)
13     for (j = 0; j < n; j++)
        scanf("%d", &m[i][j]);
15 }

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

23     for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
25         printf("%d ", m[i][j]);
        printf("\n");
27     }
   }

29
31 /* Funkcija proverava da li su zadate kvadratne matrice a i b
   dimenzije n jednake */
   int jednake_matrice(int a[][MAX], int b[][MAX], int n)
33 {
   int i, j;
35
   for (i = 0; i < n; i++)
37     for (j = 0; j < n; j++)
        if (a[i][j] != b[i][j])
39         return 0;

41     /* Prosla je provera jednakosti za sve parove elemenata koji su na
       istim pozicijama. To znaci da su matrice jednake */
43     return 1;
   }

45
47 /* Funkcija izracunava zbir dve kvadratne matrice */
   void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
   {
49     int i, j;
```

```

51     for (i = 0; i < n; i++)
52         for (j = 0; j < n; j++)
53             c[i][j] = a[i][j] + b[i][j];
54 }
55
56 /* Funkcija izracunava proizvod dve kvadratne matice */
57 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
58 {
59     int i, j, k;
60
61     for (i = 0; i < n; i++)
62         for (j = 0; j < n; j++) {
63             /* Mnozi se i-ta vrsta prve sa j-tom kolonom druge matrice */
64             c[i][j] = 0;
65             for (k = 0; k < n; k++)
66                 c[i][j] += a[i][k] * b[k][j];
67         }
68 }
69
70 int main()
71 {
72     /* Matrice ciji se elementi zadaju sa ulaza */
73     int a[MAX][MAX], b[MAX][MAX];
74
75     /* Matrice zbira i proizvoda */
76     int zbir[MAX][MAX], proizvod[MAX][MAX];
77
78     /* Dimenzija matrica */
79     int n;
80
81     printf("Unesite dimenziju matrica:\n");
82     scanf("%d", &n);
83
84     /* Proverava se da li je doslo do prekoracenja dimenzije */
85     if (n > MAX || n <= 0) {
86         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
87         fprintf(stderr, "matrica.\n");
88         exit(EXIT_FAILURE);
89     }
90
91     printf("Unesite elemente prve matrice, vrstu po vrstu:\n");
92     ucitaj_matricu(a, n);
93     printf("Unesite elemente druge matrice, vrstu po vrstu:\n");
94     ucitaj_matricu(b, n);
95
96     /* Izracunava se zbir i proizvod matrica */
97     saberi(a, b, zbir, n);
98     pomnozi(a, b, proizvod, n);
99
100    /* Ispisuje se rezultat */
101    if (jednake_matrice(a, b, n) == 1)
        printf("Matrice su jednake.\n");

```

```
103     else
104         printf("Matrice nisu jednake.\n");
105
106     printf("Zbir matrica je:\n");
107     ispisi_matricu(zbir, n);
108
109     printf("Proizvod matrica je:\n");
110     ispisi_matricu(proizvod, n);
111
112     return 0;
113 }
```

### Rešenje 2.10

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

#define MAX 64

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

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

    return 1;
}

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

    /* Prepisuju se vrednosti elemenata pocetne matrice */
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            zatvorenje[i][j] = m[i][j];

    /* Na glavnoj dijagonali se postavljaju jedinice */
    for (i = 0; i < n; i++)
        zatvorenje[i][i] = 1;
}
```

```

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

46    /* Obilaze se elementi ispod glavne dijagonale matrice i upoređuju
    se sa njima simetricnim elementima */
48    for (i = 0; i < n; i++)
        for (j = 0; j < i; j++)
50            if (m[i][j] != m[j][i])
52                return 0;

54    return 1;
}

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

64    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
66            zatvorenje[i][j] = m[i][j];

68    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
70            if (zatvorenje[i][j] == 1)
72                zatvorenje[j][i] = 1;
}

74

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

84    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
86            /* Ispituje se da li postoji element koji narusava *
            tranzitivnost */
            for (k = 0; k < n; k++)
88                if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)

```

```
90         return 0;
92     return 1;
93 }
94
95 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
96    relacije koriscenjem Varsalovog algoritma */
97 void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
98 {
99     int i, j, k;
100
101     /* Prepisuju se vrednosti elemenata pocetne matrice */
102     for (i = 0; i < n; i++)
103         for (j = 0; j < n; j++)
104             zatvorenje[i][j] = m[i][j];
105
106     /* Odredjuje se reflektivno zatvorenje matrice */
107     for (i = 0; i < n; i++)
108         zatvorenje[i][i] = 1;
109
110     /* Primenom Varsalovog algoritma odredjuje se tranzitivno
111        zatvorenje matrice */
112     for (k = 0; k < n; k++)
113         for (i = 0; i < n; i++)
114             for (j = 0; j < n; j++)
115                 if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
116                     && (zatvorenje[i][j] == 0))
117                     zatvorenje[i][j] = 1;
118 }
119
120 /* Funkcija ispisuje elemente matrice */
121 void pisi_matricu(int m[][MAX], int n)
122 {
123     int i, j;
124
125     for (i = 0; i < n; i++) {
126         for (j = 0; j < n; j++)
127             printf("%d ", m[i][j]);
128         printf("\n");
129     }
130 }
131
132 int main(int argc, char *argv[])
133 {
134     FILE *ulaz;
135     int m[MAX][MAX];
136     int pomocna[MAX][MAX];
137     int n, i, j;
138
139     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
140        */
```

```
142     if (argc < 2) {
143         printf("Greska: ");
144         printf("Nedovoljan broj argumenata komandne linije.\n");
145         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
146         exit(EXIT_FAILURE);
147     }
148
149     /* Otvara se datoteka za citanje */
150     ulaz = fopen(argv[1], "r");
151     if (ulaz == NULL) {
152         fprintf(stderr, "Greska: ");
153         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
154         exit(EXIT_FAILURE);
155     }
156
157     /* Ucitava se dimenzija matrice */
158     fscanf(ulaz, "%d", &n);
159
160     /* Proverava se da li je doslo do prekoracenja dimenzije */
161     if (n > MAX || n <= 0) {
162         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
163         fprintf(stderr, "matrice.\n");
164         exit(EXIT_FAILURE);
165     }
166
167     /* Ucitava se element po element matrice */
168     for (i = 0; i < n; i++)
169         for (j = 0; j < n; j++)
170             fscanf(ulaz, "%d", &m[i][j]);
171
172     /* Ispisuje se rezultat */
173     printf("Relacija %s refleksivna.\n",
174           refleksivnost(m, n) == 1 ? "jeste" : "nije");
175
176     printf("Relacija %s simetricna.\n",
177           simetricnost(m, n) == 1 ? "jeste" : "nije");
178
179     printf("Relacija %s tranzitivna.\n",
180           tranzitivnost(m, n) == 1 ? "jeste" : "nije");
181
182     printf("Refleksivno zatvorenje relacije:\n");
183     ref_zatvorenje(m, n, pomocna);
184     pisi_matricu(pomocna, n);
185
186     printf("Simetricno zatvorenje relacije:\n");
187     sim_zatvorenje(m, n, pomocna);
188     pisi_matricu(pomocna, n);
189
190     printf("Refleksivno-tranzitivno zatvorenje relacije:\n");
191     ref_tran_zatvorenje(m, n, pomocna);
192     pisi_matricu(pomocna, n);
```



```
194  /* Zatvara se datoteka */  
    fclose(ulaz);  
196  return 0;  
}
```

### Rešenje 2.11

```
#include <stdio.h>  
2  #include <stdlib.h>  
  
4  #define MAX 32  
  
6  /* Funkcija izracunava najveći element na sporednoj dijagonali. Za  
   elemente sporedne dijagonale vazi da je zbir indeksa vrste i  
   indeksa kolone jednak n-1 */  
8  int max_sporedna_dijagonala(int m[][MAX], int n)  
10 {  
    int i;  
12    int max_na_sporednoj_dijagonali = m[0][n - 1];  
  
14    for (i = 1; i < n; i++)  
        if (m[i][n - 1 - i] > max_na_sporednoj_dijagonali)  
16            max_na_sporednoj_dijagonali = m[i][n - 1 - i];  
  
18    return max_na_sporednoj_dijagonali;  
}  
20  
/* Funkcija izracunava indeks kolone najmanjeg elementa */  
22 int indeks_min(int m[][MAX], int n)  
{  
24    int i, j;  
    int min = m[0][0], indeks_kolone = 0;  
  
26    for (i = 0; i < n; i++)  
        for (j = 0; j < n; j++)  
28            if (m[i][j] < min) {  
30                min = m[i][j];  
                indeks_kolone = j;  
32            }  
  
34    return indeks_kolone;  
}  
36  
/* Funkcija izracunava indeks vrste najvećeg elementa */  
38 int indeks_max(int m[][MAX], int n)  
{  
40    int i, j;  
    int max = m[0][0], indeks_vrste = 0;  
42  
    for (i = 0; i < n; i++)
```

```
44     for (j = 0; j < n; j++)
45         if (m[i][j] > max) {
46             max = m[i][j];
47             indeks_vrste = i;
48         }
49     return indeks_vrste;
50 }

52 /* Funkcija izracunava broj negativnih elemenata matrice */
53 int broj_negativnih(int m[][MAX], int n)
54 {
55     int i, j;
56     int broj_negativnih = 0;

57     for (i = 0; i < n; i++)
58         for (j = 0; j < n; j++)
59             if (m[i][j] < 0)
60                 broj_negativnih++;

61     return broj_negativnih;
62 }

64 }

66 int main(int argc, char *argv[])
67 {
68     int m[MAX][MAX];
69     int n;
70     int i, j;

71     /* Proverava se broj argumenata komandne linije */
72     if (argc < 2) {
73         printf("Greska: ");
74         printf("Nedovoljan broj argumenata komandne linije.\n");
75         printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
76         exit(EXIT_FAILURE);
77     }

78     /* Ucitava se vrednost dimenzije i proverava se njena korektnost */
79     n = atoi(argv[1]);

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

85     /* Ucitava se matrica */
86     printf("Unesite elemente matrice dimenzije %d:\n", n);
87     for (i = 0; i < n; i++)
88         for (j = 0; j < n; j++)
89             scanf("%d", &m[i][j]);

90     printf("Najveci element sporedne dijagonale je %d.\n",
```

```

96         max_sporedna_dijagonala(m, n));
98     printf("Indeks kolone sa najmanjim elementom je %d.\n",
           indeks_min(m, n));
100
102     printf("Indeks vrste sa najvećim elementom je %d.\n",
           indeks_max(m, n));
104
106     printf("Broj negativnih elemenata matrice je %d.\n",
           broj_negativnih(m, n));
108
109     return 0;
110 }

```

### Rešenje 2.12

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 32
5
   /* Funkcija ucitava elemente kvadratne matrice sa standardnog ulaza
      */
7  void ucitaj_matricu(int m[][MAX], int n)
   {
9      int i, j;
11
12     for (i = 0; i < n; i++)
13         for (j = 0; j < n; j++)
14             scanf("%d", &m[i][j]);
15 }
16
17 /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
   */
18 void ispis_matricu(int m[][MAX], int n)
   {
19     int i, j;
21
22     for (i = 0; i < n; i++) {
23         for (j = 0; j < n; j++)
24             printf("%d ", m[i][j]);
25         printf("\n");
26     }
27 }
28
29 /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
   da li je normirana i ortogonalna. Matrica je normirana ako je
   proizvod svake vrste matrice sa samom sobom jednak jedinici.
   Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
   vrste matrice jednak nuli */
31 int ortonormirana(int m[][MAX], int n)
32 {
33

```

```
{
35     int i, j, k;
36     int proizvod;
37
38     /* Ispituje se uslov normiranosti */
39     for (i = 0; i < n; i++) {
40         proizvod = 0;
41
42         for (j = 0; j < n; j++)
43             proizvod += m[i][j] * m[i][j];
44
45         if (proizvod != 1)
46             return 0;
47     }
48
49     /* Ispituje se uslov ortogonalnosti */
50     for (i = 0; i < n - 1; i++) {
51         for (j = i + 1; j < n; j++) {
52
53             proizvod = 0;
54
55             for (k = 0; k < n; k++)
56                 proizvod += m[i][k] * m[j][k];
57
58             if (proizvod != 0)
59                 return 0;
60         }
61     }
62
63     /* Ako su oba uslova ispunjena, matrica je ortonormirana */
64     return 1;
65 }
66
67 int main()
68 {
69     int A[MAX][MAX];
70     int n;
71
72     printf("Unesite dimenziju matrice: ");
73     scanf("%d", &n);
74
75     if (n > MAX || n <= 0) {
76         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
77         fprintf(stderr, "matrice.\n");
78         exit(EXIT_FAILURE);
79     }
80
81     printf("Unesite elemente matrice, vrstu po vrstu:\n");
82     ucitaj_matricu(A, n);
83
84     printf("Matrica %s ortonormirana.\n",
85           ortonormirana(A, n) ? "je" : "nije");
```

```
    return 0;
87 }
```

### Rešenje 2.13

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX_V 10
5  #define MAX_K 10
7
   /* Funkcija proverava da li su ispisani svi elementi iz matrice,
      odnosno da li se narušio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
   {
11     return !(top <= bottom && left <= right);
   }
13
   /* Funkcija spiralno ispisuje elemente matrice */
15 void ispis_i_matricu_spiralno(int a[][MAX_K], int n, int m)
   {
17     int i, j, top, bottom, left, right;
19     top = left = 0;
      bottom = n - 1;
21     right = m - 1;
23     while (!krajIspisa(top, bottom, left, right)) {
25         for (j = left; j <= right; j++)
            printf("%d ", a[top][j]);
27
      /* Spusta se prvi red */
29     top++;
31     if (krajIspisa(top, bottom, left, right))
        break;
33
      for (i = top; i <= bottom; i++)
35         printf("%d ", a[i][right]);
37
      /* Pomera se desna kolona za naredni krug ispisa blize levom
         kraju */
39     right--;
41     if (krajIspisa(top, bottom, left, right))
        break;
43
      /* Ispisuje se donja vrsta */
45     for (j = right; j >= left; j--)
        printf("%d ", a[bottom][j]);
```

```

47      /* Podize se donja vrsta za naredni krug ispisa */
48      bottom--;
49
50      if (krajIspisa(top, bottom, left, right))
51          break;
52
53      /* Ispisuje se prva kolona */
54      for (i = bottom; i >= top; i--)
55          printf("%d ", a[i][left]);
56
57      /* Priprema se leva kolona za naredni krug ispisa */
58      left++;
59  }
60  putchar('\n');
61 }
62
63 /* Funkcija učitava matricu */
64 void ucitaj_matricu(int a[][MAX_K], int n, int m)
65 {
66     int i, j;
67
68     for (i = 0; i < n; i++)
69         for (j = 0; j < m; j++)
70             scanf("%d", &a[i][j]);
71 }
72
73 int main()
74 {
75     int a[MAX_V][MAX_K];
76     int m, n;
77
78     printf("Unesite broj vrsta i broj kolona matrice:\n");
79     scanf("%d %d", &n, &m);
80
81     if (n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
82         fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
83         fprintf(stderr, "matrice.\n");
84         exit(EXIT_FAILURE);
85     }
86
87     printf("Unesite elemente matrice, vrstu po vrstu:\n");
88     ucitaj_matricu(a, n, m);
89
90     printf("Spiralno ispisana matrica: ");
91     ispisi_matricu_spiralno(a, n, m);
92
93     return 0;
94 }

```

## Rešenje 2.15

```

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

3
4
5 int main()
6 {
7     int *p = NULL;
8     int i, n;

9     printf("Unesite dimenziju niza: ");
10    scanf("%d", &n);

11
12    /* Alocira se prostor za n celih brojeva */
13    if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
14        fprintf(stderr, "malloc(): ");
15        fprintf(stderr, "greska pri alokaciji memorije.\n");
16        exit(EXIT_FAILURE);
17    }

18    printf("Unesite elemente niza: ");
19    for (i = 0; i < n; i++)
20        scanf("%d", &p[i]);

21
22    printf("Niz u obrnutom poretku je: ");
23    for (i = n - 1; i >= 0; i--)
24        printf("%d ", p[i]);
25    printf("\n");

26
27    /* Oslobadja se prostor rezervisan funkcijom malloc() */
28    free(p);

29
30    return 0;
31 }

```

### Rešenje 2.16

```

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

4
5
6 int main()
7 {
8     /* Adresa prvog alociranog bajta */
9     int *a = NULL;

10    /* Velicina alocirane memorije */
11    int alocirano;

12
13    /* Broj elemenata niza */
14    int n;

```

```
16  /* Broj koji se ucitava sa ulaza */
17  int x;
18  int i;
19  int *b = NULL;
20  char realokacija;
21
22  /* Inicijalizacija */
23  alocirano = n = 0;
24
25  printf("Unesite zeljeni nacin realokacije (M ili R):\n");
26  scanf("%c", &realokacija);
27
28  printf("Unesite brojeve, nulu za kraj:\n");
29  scanf("%d", &x);
30
31  while (x != 0) {
32      if (n == alocirano) {
33          alocirano = alocirano + KORAK;
34
35          if (realokacija == 'M') {
36              /* Vrsi se realokacija memorije sa novom velicinom */
37              b = (int *) malloc(alocirano * sizeof(int));
38
39              if (b == NULL) {
40                  fprintf(stderr, "malloc(): ");
41                  fprintf(stderr, "greska pri alokaciji memorije.\n");
42                  free(a);
43                  exit(EXIT_FAILURE);
44              }
45
46              /* Svih n elemenata koji pocinju na adresi a prepisujemo na
47               novu adresu b */
48              for (i = 0; i < n; i++)
49                  b[i] = a[i];
50
51              free(a);
52
53              /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
54               bloka koji je prilikom alokacije zapamcen u promenljivoj b
55               */
56              a = b;
57          } else if (realokacija == 'R') {
58
59              /* Zbog funkcije realloc je neophodno da i u prvoj iteraciji
60               "a" bude inicijalizovano na NULL */
61
62              a = (int *) realloc(a, alocirano * sizeof(int));
63              if (a == NULL) {
64                  fprintf(stderr, "realloc(): ");
65                  fprintf(stderr, "greska pri alokaciji memorije.\n");
66                  exit(EXIT_FAILURE);
67              }
68          }
69      }
70      x--;
71  }
```



```

68     }
69     }
70     a[n++] = x;
71
72     scanf("%d", &x);
73 }
74
75 printf("Niz u obrnutom poretku je: ");
76 for (n--; n >= 0; n--)
77     printf("%d ", a[n]);
78 printf("\n");
79
80 /* Oslobadja se dinamicki alocirana memorija */
81 free(a);
82
83 exit(EXIT_SUCCESS);
84 }

```

### Rešenje 2.17

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX 1000
6
7  /* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
8     nadovezivanja niski. Adresa niza se vraca kao povratna vrednost.
9     */
10 char *nadovezi(char *s, char *t)
11 {
12     char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
13                               * sizeof(char));
14
15     /* Proverava se da li je memorija uspesno alocirana */
16     if (p == NULL) {
17         fprintf(stderr, "malloc(): ");
18         fprintf(stderr, "greska pri alokaciji memorije.\n");
19         exit(EXIT_FAILURE);
20     }
21
22     /* Kopiraju se i nadovezuju niske karaktera */
23     strcpy(p, s);
24     strcat(p, t);
25
26     return p;
27 }
28
29 int main()
30 {

```

```

30 char *s = NULL;
31 char s1[MAX], s2[MAX];
32
33 printf("Unesite dve niske karaktera:\n");
34 scanf("%s", s1);
35 scanf("%s", s2);
36
37 /* Poziva se funkcija koja nadovezuje niske */
38 s = nadovezi(s1, s2);
39
40 /* Prikazuje se rezultat */
41 printf("Nadovezane niske: %s\n", s);
42
43 /* Oslobadja se memorija alocirana u funkciji nadovezi() */
44 free(s);
45
46 return 0;
47 }

```

### Rešenje 2.18

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  int main()
6  {
7      int i, j;
8
9      /* Pokazivac na dinamicki alociran niz pokazivaca na vrste matrice
10       */
11     double **A = NULL;
12
13     /* Broj vrsta i broj kolona */
14     int n = 0, m = 0;
15
16     /* Trag matrice */
17     double trag = 0;
18
19     printf("Unesite broj vrsta i broj kolona matrice:\n ");
20     scanf("%d%d", &n, &m);
21
22     /* Dinamicki se alocira prostor za n pokazivaca na double */
23     A = malloc(sizeof(double *) * n);
24
25     /* Provera se da li je doslo do greske pri alokaciji */
26     if (A == NULL) {
27         fprintf(stderr, "malloc(): ");
28         fprintf(stderr, "greska pri alokaciji memorije.\n");
29         exit(EXIT_FAILURE);
30     }

```

```

31  /* Dinamicki se alokira prostor za elemente u vrstama */
    for (i = 0; i < n; i++) {
33      A[i] = malloc(sizeof(double) * m);

35      /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
         potrebno je osloboditi svih i-1 prethodno alociranih vrsta, i
         alocirani niz pokazivaca */
37      if (A[i] == NULL) {
39          for (j = 0; j < i; j++)
              free(A[j]);
41          free(A);
              exit(EXIT_FAILURE);
43      }
    }

45     printf("Unesite elemente matrice, vrstu po vrstu:\n");
47     for (i = 0; i < n; i++)
         for (j = 0; j < m; j++)
49         scanf("%lf", &A[i][j]);

51     /* Izracunava se trag matrice, odnosno suma elemenata na glavnoj
         dijagonali */
53     trag = 0.0;

55     for (i = 0; i < n; i++)
         trag += A[i][i];

57     printf("Trag unete matrice je %.2f.\n", trag);

59     /* Oslobadja se prostor rezervisan za svaku vrstu */
61     for (j = 0; j < n; j++)
         free(A[j]);

63     /* Oslobadja se memorija za niz pokazivaca na vrste */
65     free(A);

67     return 0;
}

```

### Rešenje 2.19

Datoteka 2.1: *matrica.h*

```

1  #ifndef _MATRICA_H_
    #define _MATRICA_H_ 1
3
    /* Funkcija dinamicki alokira memoriju za matricu dimenzija n x m */
5  int **alociraj_matricu(int n, int m);

```

```

7  /* Funkcija dinamički alokira memoriju za kvadratnu matricu dimenzije
   n */
9  int **alociraj_kvadratnu_matricu(int n);

11 /* Funkcija dealocira memoriju za matricu sa n vrsta */
   int **deallociraj_matricu(int **matrica, int n);
13
15 /* Funkcija učitava već alociranu matricu dimenzija n x m sa
   standardnog ulaza */
   void ucitaj_matricu(int **matrica, int n, int m);
17
19 /* Funkcija učitava već alociranu kvadratnu matricu dimenzije n sa
   standardnog ulaza */
   void ucitaj_kvadratnu_matricu(int **matrica, int n);
21
23 /* Funkcija ispisuje matricu dimenzija n x m na standardnom izlazu */
   void ispisi_matricu(int **matrica, int n, int m);

25 /* Funkcija ispisuje kvadratnu matricu dimenzije n na standardnom
   izlazu */
27 void ispisi_kvadratnu_matricu(int **matrica, int n);

29 /* Funkcija učitava već alociranu matricu dimenzija n x m iz datoteke
   f */
31 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
   ;

33 /* Funkcija učitava već alociranu kvadratnu matricu dimenzije n iz
   datoteke f */
35 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
                                           FILE * f);
37
39 /* Funkcija upisuje matricu dimenzija n x m u datoteku f */
   int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);

41 /* Funkcija upisuje kvadratnu matricu dimenzije n u datoteku f */
   int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
                                           FILE * f);
43
45 #endif

```

Datoteka 2.2: *matrica.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "matrica.h"

5  int **alociraj_matricu(int n, int m)
   {
7      int **matrica = NULL;
       int i, j;

```

```

9
11 /* Alocira se prostor za niz vrsti matrice */
12 matrica = (int **) malloc(n * sizeof(int *));
13 /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije ce
14 biti NULL, sto mora biti provereno u main funkciji */
15 if (matrica == NULL)
16     return NULL;
17
18 /* Alocira se prostor za svaku vrstu matrice */
19 for (i = 0; i < n; i++) {
20     matrica[i] = (int *) malloc(m * sizeof(int));
21     /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
22        prethodno alocirani resursi, i povratna vrednost je NULL */
23     if (matrica[i] == NULL) {
24         for (j = 0; j < i; j++)
25             free(matrica[j]);
26         free(matrica);
27         return NULL;
28     }
29     return matrica;
30 }
31
32 int **alociraj_kvadratnu_matricu(int n)
33 {
34     /* Alociranje matrice dimenzije n x n */
35     return alociraj_matricu(n, n);
36 }
37
38 int **dealociraj_matricu(int **matrica, int n)
39 {
40     int i;
41     /* Oslobadja se prostor rezervisan za svaku vrstu */
42     for (i = 0; i < n; i++)
43         free(matrica[i]);
44     /* Oslobadja se memorija za niz pokazivaca na vrste */
45     free(matrica);
46
47     /* Matrica postaje prazna, tj. nealocirana */
48     return NULL;
49 }
50
51 void ucitaj_matricu(int **matrica, int n, int m)
52 {
53     int i, j;
54     /* Elementi matrice se ucitavaju po vrstama */
55     for (i = 0; i < n; i++)
56         for (j = 0; j < m; j++)
57             scanf("%d", &matrica[i][j]);
58 }
59
60 void ucitaj_kvadratnu_matricu(int **matrica, int n)

```

```
61 {
62     /* Ucitavanje matrice n x n */
63     ucitaj_matricu(matrica, n, n);
64 }
65
66 void ispisi_matricu(int **matrica, int n, int m)
67 {
68     int i, j;
69     /* Ispis po vrstama */
70     for (i = 0; i < n; i++) {
71         for (j = 0; j < m; j++)
72             printf("%d ", matrica[i][j]);
73         printf("\n");
74     }
75 }
76
77 void ispisi_kvadratnu_matricu(int **matrica, int n)
78 {
79     /* Ispis matrice n x n */
80     ispisi_matricu(matrica, n, n);
81 }
82
83 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
84 {
85     int i, j;
86     /* Elementi matrice se ucitavaju po vrstama */
87     for (i = 0; i < n; i++)
88         for (j = 0; j < m; j++)
89             /* Ako je nemoguće učitati sledeći element, povratna vrednost
90              funkcije je 1, kao indikator neuspesnog učitavanja */
91             if (fscanf(f, "%d", &matrica[i][j]) != 1)
92                 return 1;
93
94     /* Uspesno učitana matrica */
95     return 0;
96 }
97
98 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
99                                         FILE * f)
100 {
101     /* Ucitavanje matrice n x n iz datoteke */
102     return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
103 }
104
105 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
106 {
107     int i, j;
108     /* Ispis po vrstama */
109     for (i = 0; i < n; i++) {
110         for (j = 0; j < m; j++)
111             /* Ako je nemoguće ispisati sledeći element, povratna vrednost
112              funkcije je 1, kao indikator neuspesnog ispisa */
```

```

113     if (fprintf(f, "%d ", matrica[i][j]) <= 0)
114         return 1;
115     fprintf(f, "\n");
116 }
117
118 /* Uspesno upisana matrica */
119 return 0;
120 }
121
122 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n, FILE * f
123 )
124 {
125     /* Ispis matrice n x n u datoteku */
126     return upisi_matricu_u_datoteku(matrica, n, n, f);
127 }

```

Datoteka 2.3: *main\_a.c*

```

#include <stdio.h>
2 #include <stdlib.h>
#include "matrica.h"
4
int main()
6 {
    int **matrica = NULL;
8     int n, m;
    FILE *f;
10
    /* Ucitavanje dimenzije matrice */
12     printf("Unesi broj vrsta matrice: ");
    scanf("%d", &n);
14     printf("Unesi broj kolona matrice: ");
    scanf("%d", &m);
16
    /* Provera dimenzija matrice */
18     if (n <= 0 || m <= 0) {
        fprintf(stderr, "Neodgovarajce dimenzije matrice\n");
20         exit(EXIT_FAILURE);
    }
22
    /* Alokacija matrice i provera alokacije */
24     matrica = alociraj_matricu(n, m);
    if (matrica == NULL) {
26         fprintf(stderr, "Neuspesna alokacija matrice\n");
        exit(EXIT_FAILURE);
28     }

30     /* Ucitavanje matrice sa standardnog ulaza */
    printf("Unesi elemente matrice po vrstama:\n");
32     ucitaj_matricu(matrica, n, m);

```

```

34  /* Otvaranje fajla za upis matrice */
    if ((f = fopen("matrica.txt", "w")) == NULL) {
36      fprintf(stderr, "fopen() error\n");
      matrica = dealociraj_matricu(matrica, n);
38      exit(EXIT_FAILURE);
    }

40

    /* Upis matrice u fajl */
42    if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
      fprintf(stderr, "Neuspesno upisivanje matrice u datoteku\n");
44      matrica = dealociraj_matricu(matrica, n);
      exit(EXIT_FAILURE);
46    }

48    /* Zatvaranje fajla */
    fclose(f);

50

    /* Dealokacija matrice */
52    matrica = dealociraj_matricu(matrica, n);

54    return 0;
}

```

Datoteka 2.4: *main\_b.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "matrica.h"

5  int main(int argc, char **argv)
   {
7      int **matrica = NULL;
      int n;
9      FILE *f;

11     /* Provera argumenata komandne linije */
      if (argc != 2) {
13         fprintf(stderr, "Koriscenje programa: %s datoteka\n", argv[0]);
         exit(EXIT_FAILURE);
15     }

17     /* Otvaranje fajla za citanje */
      if ((f = fopen(argv[1], "r")) == NULL) {
19         fprintf(stderr, "fopen() error\n");
         exit(EXIT_FAILURE);
21     }

23     /* Ucitavanje dimenzije matrice */
      if (fscanf(f, "%d", &n) != 1) {
25         fprintf(stderr, "Neispravan pocetak fajla\n");
         exit(EXIT_FAILURE);

```



```

27     }

29     /* Provera dimenzije matrice */
30     if (n <= 0) {
31         fprintf(stderr, "Neodgovarajca dimenzija matrice\n");
32         exit(EXIT_FAILURE);
33     }

35     /* Alokacija matrice i provera alokacije */
36     matrica = alociraj_kvadratnu_matricu(n);
37     if (matrica == NULL) {
38         fprintf(stderr, "Neuspesna alokacija matrice\n");
39         exit(EXIT_FAILURE);
40     }

41     /* Ucitavanje matrice iz datoteke */
42     if (ucitaj_kvadratnu_matricu_iz_datoteke(matrica, n, f) != 0) {
43         fprintf(stderr, "Neuspesno ucitavanje matrice iz datoteke\n");
44         matrica = dealociraj_matricu(matrica, n);
45         exit(EXIT_FAILURE);
46     }

47     /* Zatvaranje fajla */
48     fclose(f);

49     /* Ispis matrice na standardnom izlazu */
50     ispisi_kvadratnu_matricu(matrica, n);

51     /* Dealokacija matrice */
52     matrica = dealociraj_matricu(matrica, n);

53     return 0;
54 }

```

### Rešenje 2.20

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "matrica.h"

5

6  /* Funkcija ispisuje elemente matrice ispod glavne dijagonale */
7  void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
8  {
9      int i, j;

10

11     for (i = 0; i < n; i++) {
12         for (j = 0; j <= i; j++)
13             printf("%d ", M[i][j]);
14         printf("\n");
15     }

```

```

17 }
18
19 int main()
20 {
21     int m, n;
22     int **matrica = NULL;
23
24     printf("Unesite broj vrsta i broj kolona matrice:\n ");
25     scanf("%d %d", &n, &m);
26
27     /* Alocira se matrica */
28     matrica = alociraj_matricu(n, m);
29     /* Provera alokacije */
30     if (matrica == NULL) {
31         fprintf(stderr, "Neuspesna alokacija matrice\n");
32         exit(EXIT_FAILURE);
33     }
34
35     printf("Unesite elemente matrice, vrstu po vrstu:\n");
36     ucitaj_matricu(matrica, n, m);
37
38     printf("Elementi ispod glavne dijagonale matrice:\n");
39     ispisi_elemente_ispod_dijagonale(matrica, n, m);
40
41     /* Oslobadjanje memorije */
42     matrica = dealociraj_matricu(matrica, n);
43
44     return 0;
45 }

```

## Rešenje 2.22

```

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            if (i < j)
13                a[i][j] /= 2;
14            else if (i > j)
15                a[i][j] *= 2;
16 }
17
18 /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
19    sporedne dijagonale. Element se nalazi ispod sporedne dijagonale

```

```
20     ukoliko je zbir indeksa vrste i indeksa kolone elementa veci od
      n-1 */
22 float zbir_ispod_sporedne_dijagonale(float **m, int n)
23 {
24     int i, j;
      float zbir = 0;
26
      for (i = 0; i < n; i++)
28         for (j = 0; j < n; j++)
29             if (i + j > n - 1)
30                 zbir += fabs(m[i][j]);
32
      return zbir;
33 }
34
35 /* Funkcija ucitava elemente kvadratne matrice dimenzije n iz zadate
      datoteke */
36 void ucitaj_matricu(FILE * ulaz, float **m, int n)
37 {
38     int i, j;
39
40     for (i = 0; i < n; i++)
41         for (j = 0; j < n; j++)
42             fscanf(ulaz, "%f", &m[i][j]);
43 }
44
45 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
      standardni izlaz */
46 void ispisi_matricu(float **m, int n)
47 {
48     int i, j;
49
50     for (i = 0; i < n; i++) {
51         for (j = 0; j < n; j++)
52             printf("%.2f ", m[i][j]);
53         printf("\n");
54     }
55 }
56
57 /* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
58 float **alociraj_memoriju(int n)
59 {
60     int i, j;
      float **m;
62
63     m = (float **) malloc(n * sizeof(float *));
64     if (m == NULL) {
65         fprintf(stderr, "malloc(): Neuspela alokacija\n");
66         exit(EXIT_FAILURE);
67     }
68
69     for (i = 0; i < n; i++) {
```

```
72     m[i] = (float *) malloc(n * sizeof(float));

74     if (m[i] == NULL) {
75         printf("malloc(): neuspela alokacija memorije!\n");
76         for (j = 0; j < i; j++)
77             free(m[j]);
78         free(m);
79         exit(EXIT_FAILURE);
80     }
81 }
82 return m;
83 }

84 /* Funckija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
85    n */
86 void oslobodi_memoriju(float **m, int n)
87 {
88     int i;
89
90     for (i = 0; i < n; i++)
91         free(m[i]);
92     free(m);
93 }

94

95 int main(int argc, char *argv[])
96 {
97     FILE *ulaz;
98     float **a;
99     int n;

100

101     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
102        */
103     if (argc < 2) {
104         printf("Greska: ");
105         printf("Nedovoljan broj argumenata komandne linije.\n");
106         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
107         exit(EXIT_FAILURE);
108     }

109

110     /* Otvara se datoteka za citanje */
111     ulaz = fopen(argv[1], "r");
112     if (ulaz == NULL) {
113         fprintf(stderr, "Greska: ");
114         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
115         exit(EXIT_FAILURE);
116     }

117

118     /* Cita se dimenzija matrice */
119     fscanf(ulaz, "%d", &n);

120

121     /* Alocira se memorija */
122     a = alociraj_memoriju(n);
```

```

124  /* Ucitavaju se elementi matrice */
    ucitaj_matricu(ulaz, a, n);
126
    float zbir = zbir_ispod_sporedne_dijagonale(a, n);
128
    /* Poziva se funkcija za transformaciju matrice */
130    izmeni(a, n);
132
    /* Ispisuje se rezultat */
    printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
134    printf("je %.2f.\n", zbir);
136
    printf("Transformisana matrica je:\n");
    ispisi_matricu(a, n);
138
    /* Oslobadja se memorija */
140    oslobodi_memoriju(a, n);
142
    /* Zatvara se datoteka */
    fclose(ulaz);
144
    return 0;
146 }

```

### Rešenje 2.27

```

1  #include <stdio.h>
3  #include <stdlib.h>
   #include <math.h>
5  #include <string.h>
7
   /* Funkcija tabela() prihvata granice intervala a i b, broj
      ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
9  funkciju koja prihvata double argument, i vraca double vrednost.
      Za tako datu funkciju ispisuju se njene vrednosti u intervalu
11 [a,b] u n ekvidistantnih tacaka intervala */
   void tabela(double a, double b, int n, double (*fp) (double))
13 {
    int i;
15    double x;
17
    printf("-----\n");
    for (i = 0; i < n; i++) {
19        x = a + i * (b - a) / (n - 1);
        printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
21    }
    printf("-----\n");
23 }

```

```
25 double sqr(double a)
26 {
27     return a * a;
28 }
29
30 int main(int argc, char *argv[])
31 {
32     double a, b;
33     int n;
34
35     char ime_fje[6];
36
37     /* Pokazivac na funkciju koja ima jedan argument tipa double i
38        povratnu vrednost istog tipa */
39     double (*fp) (double);
40
41     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
42        */
43     if (argc < 2) {
44         printf("Greska: ");
45         printf("Nedovoljan broj argumenata komandne linije.\n");
46         printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
47             argv[0]);
48         exit(EXIT_FAILURE);
49     }
50
51     /* Niska ime_fje sadrzi ime trazene funkcije koja je navedena u
52        komandnoj liniji */
53     strcpy(ime_fje, argv[1]);
54
55     /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
56        */
57     if (strcmp(ime_fje, "sin") == 0)
58         fp = &sin;
59     else if (strcmp(ime_fje, "cos") == 0)
60         fp = &cos;
61     else if (strcmp(ime_fje, "tan") == 0)
62         fp = &tan;
63     else if (strcmp(ime_fje, "atan") == 0)
64         fp = &atan;
65     else if (strcmp(ime_fje, "acos") == 0)
66         fp = &acos;
67     else if (strcmp(ime_fje, "asin") == 0)
68         fp = &asin;
69     else if (strcmp(ime_fje, "exp") == 0)
70         fp = &exp;
71     else if (strcmp(ime_fje, "log") == 0)
72         fp = &log;
73     else if (strcmp(ime_fje, "log10") == 0)
74         fp = &log10;
75     else if (strcmp(ime_fje, "sqrt") == 0)
76         fp = &sqrt;
```

```
77     else if (strcmp(ime_fje, "floor") == 0)
78         fp = &floor;
79     else if (strcmp(ime_fje, "ceil") == 0)
80         fp = &ceil;
81     else if (strcmp(ime_fje, "sqr") == 0)
82         fp = &sqr;
83     else {
84         printf("Program jos uvek ne podrzava trazenu funkciju!\n");
85         exit(EXIT_SUCCESS);
86     }
87
88     printf("Unesite krajeve intervala:\n");
89     scanf("%lf %lf", &a, &b);
90
91     printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
92     printf("(ukljucujuci krajeve intervala)?\n");
93     scanf("%d", &n);
94
95     /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
96        uneti broj veci od 2 */
97     if (n < 2) {
98         fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
99         exit(EXIT_FAILURE);
100     }
101
102     /* Ispisuje se ime funkcije */
103     printf("      x %10s(x)\n", ime_fje);
104
105     /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
106        komandne linije */
107     tabela(a, b, n, fp);
108     exit(EXIT_SUCCESS);
109 }
```

## Glava 3

# Algoritmi pretrage i sortiranja

### 3.1 Algoritmi pretrage

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

- (a) Napisati funkciju `linarna_pretraga` koja vrši linearnu pretragu niza celih brojeva `a`, dužine `n`, tražeći u njemu broj `x`.
- (b) Napisati funkciju `binarna_pretraga` koja vrši binarnu pretragu sortiranog niza `a`, dužine `n`, tražeći u njemu broj `x`.
- (c) Napisati funkciju `interpolaciona_pretraga` koja vrši interpolacionu pretragu sortiranog niza `a`, dužine `n`, tražeći u njemu broj `x`.

Napisati i program koji generiše rastući niz slučajnih brojeva dimenzije `n` i pozivajući napisane funkcije traži broj `x`. Programu se kao prvi argument komandne linije prosleđuje prirodan broj `n` koji nije veći od 1000000 i broj `x` kao drugi argument komandne linije. Potrebna vremena za izvršavanje ovih funkcija dopisati u datoteku `vremena.txt`.



*Test 1*

<pre> Poziv: ./a.out 1000000 23542 IZLAZ: Linearna pretraga: Element nije u nizu Binarna pretraga: Element nije u nizu Interpolaciona pretraga: Element nije u nizu </pre>	<pre> VREMENA.TXT Dimenzija niza: 1000000 Linearna: 3615091 ns Binarna: 1536 ns Interpolaciona: 558 ns </pre>
--	---

*Test 2*

<pre> Poziv: ./a.out 100000 37842 IZLAZ: Linearna pretraga: Element nije u nizu Binarna pretraga: Element nije u nizu Interpolaciona pretraga: Element nije u nizu </pre>	<pre> VREMENA.TXT Dimenzija niza: 1000000 Linearna: 3615091 ns Binarna: 1536 ns Interpolaciona: 558 ns  Dimenzija niza: 100000 Linearna: 360803 ns Binarna: 1187 ns Interpolaciona: 628 ns </pre>
---	---

[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, svođenjem pretrage na prefiks i na sufiks niza. Prvo se unosi broj koji se traži, a zatim sortirani elementi niza sve do kraja ulaza. Pretpostaviti da niz brojeva koji se unosi neće biti duži od 1024 elemenata.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite trazeni 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.

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite trazeni broj: 14
Unesite sortiran niz elemenata:
10 32 35 43 66 89 100
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 studenta po broju indeksa rastuće.

Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks ili prezime studenta čije informacije se potom prikazuju na ekranu. U slučaju više studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti `-indeks` ili `-prezime`. U slučaju neuspešnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složeno-sti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

#### Primer 1

```
Poziv: ./a.out datoteka.txt -indeks
```

```
DATOTEKA.TXT
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 SA PROGRAMOM:

```
Unesite indeks studenta
cije informacije zelite:
20140076
Indeks: 20140076,
Ime i prezime: Sonja Stevanovic
```

#### Primer 2

```
Poziv: ./a.out datoteka.txt -prezime
```

```
DATOTEKA.TXT
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 SA PROGRAMOM:

```
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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>POZIV: ./a.out dat.txt -x</code>	<code>POZIV: ./a.out dat.txt</code>	<code>POZIV: ./a.out dat.txt -y</code>
<code>DAT.TXT</code>	<code>DAT.TXT</code>	<code>DAT.TXT</code>
<code>12 53</code>	<code>12 53</code>	<code>12 53</code>
<code>2.342 34.1</code>	<code>2.342 34.1</code>	<code>2.342 34.1</code>
<code>-0.3 23</code>	<code>-0.3 23</code>	<code>-0.3 0.23</code>
<code>-1 23.1</code>	<code>-1 2.1</code>	<code>-1 2.1</code>
<code>123.5 756.12</code>	<code>123.5 756.12</code>	<code>123.5 756.12</code>
<code>IZLAZ:</code>	<code>IZLAZ:</code>	<code>IZLAZ:</code>
<code>-0.3 23</code>	<code>-1 2.1</code>	<code>-0.3 0.23</code>

[Rešenje 3.5]

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

*Test 1*

```

IZLAZ:
1.57031

```

[Rešenje 3.6]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>ULAZ:</code>	<code>ULAZ:</code>	<code>ULAZ:</code>
<code>-151 -44 5 12 13 15</code>	<code>-100 -15 -11 -8 -7 -5</code>	<code>-100 -15 0 13 55 124</code>
<code>IZLAZ:</code>	<code>IZLAZ:</code>	<code>258 315 516 7000</code>
<code>2</code>	<code>-1</code>	<code>IZLAZ:</code>
		<code>3</code>

[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 opadajući niz celih brojeva koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

<p><i>Test 1</i></p> <pre> ULAZ:  151 44 5 -12 -13 -15 IZLAZ:  3 </pre>	<p><i>Test 2</i></p> <pre> ULAZ:  100 55 15 0 -15 -124  -155 -258 -315 -516 IZLAZ:  4 </pre>	<p><i>Test 3</i></p> <pre> ULAZ:  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 pozitivan broj učitava sa standardnog ulaza, a logaritam ispisuje na standardnom izlazu.

<p><i>Test 1</i></p> <pre> ULAZ:  4 IZLAZ:  2 2 </pre>	<p><i>Test 2</i></p> <pre> ULAZ:  17 IZLAZ:  4 4 </pre>	<p><i>Test 3</i></p> <pre> ULAZ:  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  $[0, 90^\circ]$ .*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesi broj tacaka: 2
Unesi koordinate tacaka:
2 0 2 1
1 2 2 2
26.57
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesi broj tacaka: 2
Unesi koordinate tacaka:
1 0 1 1
0 1 1 1
45
```

### Primer 3

```
INTERAKCIJA SA PROGRAMOM:
Unesi broj tacaka: 3
Unesi koordinate tacaka:
1 0 1 1
2 0 2 1
1 2 2 2
26.57
```

## 3.2 Algoritmi sortiranja

**Zadatak 3.11** Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom (engl. **selection sort**), sortiranja spajanjem (engl. **merge sort**), brzog sortiranja (engl. **quick sort**), mehurastog sortiranja (engl. **bubble sort**), sortiranja direktnim umetanjem (engl. **insertion sort**) i sortiranja umetanjem sa inkrementom (engl. **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. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za sortiranje spajanjem, **-q** za brzo sortiranje, **-b** za mehurasto, **-i** za sortiranje direktnim umetanjem ili **-s** za sortiranje umetanjem sa inkrementom. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati algoritmom sortiranja izborom. Niz koji se sortira generisati neopadajuće ako je prisutna opcija **-r**, nerastuće ako je prisutna opcija **-o** ili potpuno slučajno ako nema nijedne opcije. Vreme meriti programom **time**. Analizirati porast vremena sa porastom dimenzije **n**.

### Test 1

```
Poziv: time ./a.out 200000
IZLAZ:
real 0m42.168s
user 0m42.100s
sys 0m0.000s
```

### Test 2

```
Poziv: time ./a.out 400000
IZLAZ:
real 2m48.395s
user 2m48.128s
sys 0m0.000s
```

### Test 3

```
Poziv: time ./a.out 800000
IZLAZ:
real 11m13.703s
user 11m12.636s
sys 0m0.000s
```

Test 4	Test 5	Test 6
<code>Poziv: time ./a.out 800000 -r</code>	<code>Poziv: time ./a.out 800000 -q</code>	<code>Poziv: time ./a.out 800000 -m</code>
IZLAZ:	IZLAZ:	IZLAZ:
real 11m21.533s	real 0m0.159s	real 0m0.137s
user 11m20.436s	user 0m0.156s	user 0m0.136s
sys 0m0.020s	sys 0m0.000s	sys 0m0.000s

[Rešenje 3.11]

**Zadatak 3.12** 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 da sortirane niske proveriti da li su identične.*

Primer 1	Primer 2	Primer 3
INTERAKCIJA SA PROGRAMOM:	INTERAKCIJA SA PROGRAMOM:	INTERAKCIJA SA PROGRAMOM:
Unesite prvu nisku <i>anagram</i>	Unesite prvu nisku <i>anagram</i>	Unesite prvu nisku <i>test</i>
Unesite drugu nisku <i>ramgana</i>	Unesite drugu nisku <i>anagram</i>	Unesite drugu nisku <i>tset</i>
jesu	nisu	jesu

[Rešenje 3.12]

**Zadatak 3.13** U datom nizu brojeva treba pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, ali neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati razliku pronađena dva broja. UPUTSTVO: *Prvo sortirati niz.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.11.*

Test 1	Test 2	Test 3
ULAZ:	ULAZ:	ULAZ:
23 64 123 76 22 7	21 654 65 123 65 12 61	34 30
IZLAZ:	IZLAZ:	IZLAZ:
1	0	4

[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. NAPOMENA: Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.11.

Test 1	Test 2	Test 3
<pre> ULAZ: 4 23 5 2 4 6 7 34 6 4 5  IZLAZ: 4 </pre>	<pre> ULAZ: 2 4 6 2 6 7 99 1  IZLAZ: 2 </pre>	<pre> ULAZ: 123  IZLAZ: 123 </pre>

[Rešenje 3.14]

**Zadatak 3.15** Napisati funkciju koja proverava da li u datom nizu postoje dva elementa čiji zbir je jednak zadatom celom broju. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz. Elementi niza se unose sve do kraja ulaza. Pretpostaviti da u niz neće biti uneto više od 256 brojeva. UPUTSTVO: *Prvo sortirati niz*. NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.11*.

Primer 1	Primer 2	Primer 3
<pre> INTERAKCIJA SA PROGRAMOM: Unesite trazeni zbir: 34 Unesite elemente niza: 134 4 1 6 30 23 da </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Unesite trazeni zbir: 12 Unesite elemente niza: 53 1 43 3 56 13 ne </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Unesite trazeni zbir: 52 Unesite elemente niza: 52 ne </pre>

[Rešenje 3.15]

**Zadatak 3.16** Napisati funkciju potpisa `int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3, int dim3)` koja prima dva sortirana niza, i na osnovu njih pravi novi sortirani niz koji koji sadrži elemente oba niza. Treća dimenzija predstavlja veličinu niza u koji se smešta rezultat. Ako je ona manja od potrebne dužine, funkcija vraća -1 kao indikator neuspeha, inače vraća 0. Napisati 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.

Primer 1	Primer 2
<pre> INTERAKCIJA SA PROGRAMOM: Unesite elemente prvog niza: 3 6 7 11 14 35 0 Unesite elemente drugog niza: 3 5 8 0 3 3 5 6 7 8 11 14 35 </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Unesite elemente prvog niza: 1 4 7 0 Unesite elemente drugog niza: 9 11 23 54 75 0 1 4 7 9 11 23 54 75 </pre>

[Rešenje 3.16]

**Zadatak 3.17** 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 1*

```
Poziv: ./a.out prvi-deo.txt drugi-deo.txt
```

## PRVI-DEO.TXT

```
Andrija Petrovic
Anja Ilic
Ivana Markovic
Lazar Micic
Nenad Brankovic
Sofija Filipovic
Vladimir Savic
Uros Milic
```

## DRUGI-DEO.TXT

```
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Marija Stankovic
Ognjen Peric
```

## CEO-TOK.TXT

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

[Rešenje 3.17]

**Zadatak 3.18** Napisati funkcije koje sortiraju niz struktura tačaka na osnovu sledećih kriterijuma: (i) njihovog rastojanja od koordinatnog početka, (ii)  $x$  koordinata tačaka, (iii)  $y$  koordinata tačaka. Napisati program koji učitava niz tačaka iz datoteke čije se ime zadaje kao drugi argument komandne linije, i u zavisnosti od prisutnih opcija (prvi argument) u komandnoj liniji (`-o`, `-x` ili `-y`) sortira tačke po jednom od prethodna tri kriterijuma i rezultat upisuje u datoteku čije se ime zadaje kao treći argument komandne linije. U ulaznoj datoteci nije zadato više od 128 tačaka.



*Test 1*

```

Poziv: ./a.out -x in.txt out.txt

IN.TXT
3 4
11 6
7 3
2 82
-1 6

OUT.TXT
-1 6
2 82
3 4
7 3
11 6

```

*Test 2*

```

Poziv: ./a.out -o in.txt out.txt

IN.TXT
3 4
11 6
7 3
2 82
-1 6

OUT.TXT
3 4
-1 6
7 3
11 6
2 82

```

[Rešenje 3.18]

**Zadatak 3.19** Napisati program koji učitava imena i prezimena građana (najviše njih 1000) iz datoteke `biracki-spisak.txt` i kreira biračke spiskove. Jedan birački spisak je sortiran po imenu građana, a drugi po prezimenu. Program treba da ispisuje koliko građana ima isti redni broj u oba biračka spiska. Pretpostaviti da je za ime, odnosno prezime građana dovoljno 15 karaktera.

*Test 1*

```

BIRACKI-SPISAK.TXT
Bojan Golubovic
Andrija Petrovic
Anja Ilic
Aleksandra Cvetic
Dragan Markovic
Ivana Markovic
Lazar Micic
Marija Stankovic
Filip Dukic

IZLAZ:
3

```

*Test 2*

```

BIRACKI-SPISAK.TXT
Milan Milicevic

IZLAZ:
1

```

*Test 3*

```

DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI

IZLAZ:
Problem pri otvaranju
datoteke.

```

[Rešenje 3.19]

**Zadatak 3.20** Definisati strukturu koja čuva imena, prezimena i godišta dece. Pretpostaviti da su imena i prezimena niske karaktera koje nisu duže od 30 karaktera. Napisati funkciju koja sortira niz dece po godištu, a decu istog godišta 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 dece.

#### Test 1

```

Poziv: ./a.out in.txt out.txt

IN.OUT                                OUT.TXT
Petar Petrovic 2007                   Marija Antic 2007
Milica Antonic 2008                   Ana Petrovic 2007
Ana Petrovic 2007                     Petar Petrovic 2007
Ivana Ivanovic 2009                   Milica Antonic 2008
Dragana Markovic 2010                 Ivana Ivanovic 2009
Marija Antic 2007                     Dragana Markovic 2010

```

#### Test 2

```

Poziv: ./a.out in.txt out.txt

IN.OUT                                OUT.TXT
Milijana Maric 2009                   Milijana Maric 2009

```

**Zadatak 3.21** Napisati funkciju koja sortira niz niski po broju suglasnika u niski. Ukoliko reči imaju isti broj suglasnika tada sortirati ih po dužini niske rastuće, a ukoliko su i dužine jednake onda leksikografski rastuće. Napisati program koji testira ovu funkciju za niske koje se zadaju u datoteci `niske.txt`. Pretpostaviti da u nizu nema više od 128 elemenata, kao i da svaka niska sadrži najviše 31 karakter.

#### Test 1

```

NISKE.TXT
ana petar andjela milos nikola aleksandar ljubica matej milica

IZLAZ:
ana matej milos petar milica nikola andjela ljubica aleksandar

```

[Rešenje 3.21]

**Zadatak 3.22** 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.

*Primer 1*

```

ARTIKLI.TXT
1001 Keks Jaffa 120
2530 Napolitanke Bambi 230
0023 MedenoSrce Pionir 150
2145 Pardon Marbo 70

INTERAKCIJA SA PROGRAMOM:
Asortiman:
KOD Naziv artikla Ime proizvođjaca Cena
23 MedenoSrce 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: MedenoSrce 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 trazanim 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 3.22]

**Zadatak 3.23** 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

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

### Test 1

AKTIVNOSTI.TXT

```
Aleksandra Cvetic 4 6
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Marija Stankovic 1 3
Ognjen Peric 1 2
Uros Milic 2 5
Andrija Petrovic 2 5
Anja Ilic 3 1
Lazar Micic 1 3
Nenad Brankovic 2 4
```

DAT1.TXT

```
Studenti sortirani po imenu
leksikografski rastuce:
Aleksandra Cvetic 4 6
Andrija Petrovic 2 5
Anja Ilic 3 1
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Lazar Micic 1 3
Marija Stankovic 1 3
Nenad Brankovic 2 4
Ognjen Peric 1 2
Uros Milic 2 5
```

DAT2.TXT

```
Studenti sortirani po broju zadataka
opadajuće, pa po dužini imena rastuce:
Aleksandra Cvetic 4 6
Uros Milic 2 5
Dragan Markovic 3 5
Andrija Petrovic 2 5
Nenad Brankovic 2 4
Lazar Micic 1 3
Bojan Golubovic 4 3
Marija Stankovic 1 3
Ognjen Peric 1 2
Anja Ilic 3 1
Ivana Stankovic 3 1
```

DAT3.TXT

```
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
```

[Rešenje 3.23]

**Zadatak 3.24** U datoteci `pesme.txt` nalaze se informacije o gledanosti pesama na Youtube-u. Format datoteke sa informacijama je sledeći:

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

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

- nema opcija, sortiranje se vrši po broju gledanja;

- prisutna je opcija `-i`, sortiranje se vrši po imenima izvođača;
- prisutna je opcija `-n`, sortiranje se vrši po naslovu pesama.

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Poziv: ./a.out</code>	<code>Poziv: ./a.out -i</code>	<code>Poziv: ./a.out -n</code>
<code>PESME.TXT</code>	<code>PESME.TXT</code>	<code>PESME.TXT</code>
<code>5</code>	<code>5</code>	<code>5</code>
<code>Ana - Nebo, 2342</code>	<code>Ana - Nebo, 2342</code>	<code>Ana - Nebo, 2342</code>
<code>Laza - Oblaci, 29</code>	<code>Laza - Oblaci, 29</code>	<code>Laza - Oblaci, 29</code>
<code>Pera - Ptice, 327</code>	<code>Pera - Ptice, 327</code>	<code>Pera - Ptice, 327</code>
<code>Jelena - Sunce, 92321</code>	<code>Jelena - Sunce, 92321</code>	<code>Jelena - Sunce, 92321</code>
<code>Mika - Kisa, 5341</code>	<code>Mika - Kisa, 5341</code>	<code>Mika - Kisa, 5341</code>
<code>IZLAZ:</code>	<code>IZLAZ:</code>	<code>IZLAZ:</code>
<code>Jelena - Sunce, 92321</code>	<code>Ana - Nebo, 2342</code>	<code>Mika - Kisa, 5341</code>
<code>Mika - Kisa, 5341</code>	<code>Jelena - Sunce, 92321</code>	<code>Ana - Nebo, 2342</code>
<code>Ana - Nebo, 2342</code>	<code>Laza - Oblaci, 29</code>	<code>Laza - Oblaci, 29</code>
<code>Pera - Ptice, 327</code>	<code>Mika - Kisa, 5341</code>	<code>Pera - Ptice, 327</code>
<code>Laza - Oblaci, 29</code>	<code>Pera - Ptice, 327</code>	<code>Jelena - Sunce, 92321</code>

[Rešenje 3.24]

**\*\* Zadatak 3.25** 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.

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesi niz operacija: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5
0 2 8 2 6
```

**\*\* Zadatak 3.26** 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.*

Test 1

```

|| ULAZ:
|| 23 64 123 76 22 7 34 123 54562 12 453 342 5342 42 542 1 3 432 1 32 43
||
|| IZLAZ:
|| 1 1 3 7 12 22 23 32 34 42 43 64 76 123 123 342 432 453 542 5342 54562

```

**Zadatak 3.27** 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. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

Test 1

```

|| INTERAKCIJA SA PROGRAMOM:
|| 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 2

```

|| INTERAKCIJA SA PROGRAMOM:
|| 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 3.27]

**Zadatak 3.28** Za zadatau kvadratnu matricu dimenzije  $n$  napisati funkciju koja sortira kolone matrice opadajuće na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 2
Unesite elemente matrice po vrstama:
6 -5
-4 3
Sortirana matrica je:
-5 6
3 -4
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
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
```

## 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.29** Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime, a zatim se učitava jedan karakter i pronalazi i štampa jedna struktura iz niza osoba čije prezime počinje tim karakterom. Ako takva osoba ne postoji, štampati  $-1$  na standardnom izlazu. Niz struktura ima manje od 100 elemenata i uređen je u rastućem leksikografskom poretku po prezimenima. Pretraživanje niza vršiti bibliotečkom funkcijom `bsearch`. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

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

#### Test 1

```
ULAZ:
R
IZLAZ:
Dusko Radovic
```

#### Test 2

```
ULAZ:
E
IZLAZ:
Dobrica Eric
```

#### Test 3

```
ULAZ:
X
IZLAZ:
-1
```

**Zadatak 3.30** 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.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
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
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
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 3.30]

**Zadatak 3.31** 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.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
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
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Uneti dimenziju niza: 1
Uneti elemente niza:
234
Sortirani niz u rastucem
poretku prema broju delilaca:
234
```

*Primer 3*

```
INTERAKCIJA SA PROGRAMOM:
Uneti dimenziju niza: 0
Uneti elemente niza:
Sortirani niz u rastucem
poretku prema broju
delilaca:
```

[Rešenje 3.31]

**Zadatak 3.32** Korišćenjem bibliotečke funkcije `qsort` napisati program koji sortira niz niski po sledećim kriterijumima:

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



Niske se učitavaju iz datoteke `niske.txt`. Pretpostaviti da datoteka ne sadrži više od 1000 niski kao i da je svaka niska dužine najviše 30 karaktera. Program prvo leksikografski sortira niz, primenjuje binarnu pretragu (`bsearch`) zarad traženja niske unete sa standardnog ulaza, a potom traži istu nisku koristeći funkciju `lfind` u nizu koji je neposredno pre toga sortiran po dužini. Rezultate svih sortiranja i pretraga ispisati na standardnom izlazu.

#### Primer 1

```
NISKE.TXT
ana petar andjela milos nikola aleksandar ljubica matej milica

INTERAKCIJA SA PROGRAMOM:
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
Niske sortirane po duzini:
ana matej milos petar milica nikola andjela ljubica aleksandar
Niska "matej"je pronadjena u nizu na poziciji 1
```

[Rešenje 3.32]

**Zadatak 3.33** Uraditi prethodni zadatak 3.32 sa dinamički alociranim niskama i sortiranjem niza pokazivača, umesto niza niski.

[Rešenje 3.33]

**Zadatak 3.34** Napisati program koji korišćenjem bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova ili poruka ukoliko nema takvog. Potom, sa 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 SA PROGRAMOM:
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 3.34]

**Zadatak 3.35** Uraditi zadatak 3.12, ali korišćenjem bibliotečke `qsort` funkcije.

[Rešenje 3.35]

**Zadatak 3.36** Napisati program koji sa standardnog ulaza učitava prvo ceo broj  $n$  ( $n \leq 10$ ), a zatim niz  $S$  od  $n$  niski. Maksimalna dužina svake niske je 31 karakter. Sortirati niz  $S$  bibliotečkom funkcijom `qsort` i proveriti da li u njemu ima identičnih niski.

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj niski: 4
Unesite niske:
prog search sort search
ima
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj niski: 3
Unesite niske:
test kol ispit
nema
```

#### Primer 3

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj niski: 5
Unesite niske:
a ab abc abcd abcde
nema
```

[Rešenje 3.36]

**Zadatak 3.37** Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr15125`, `mm14001`), ime, prezime i broj poena. Ni ime, ni prezime neće biti duže od 20 karaktera. Napisati

### 3.3 Bibliotečke funkcije pretrage i sortiranja

program koji korišćenjem funkcije `qsort` sortira 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 1

```
Poziv: ./a.out -n mm13321

STUDENTI.TXT
mr14123 Marko Antic 20
mm13321 Marija Radic 12
ml13011 Ivana Mitrovic 19
ml13066 Pera Simic 15
mv14003 Jovan Jovanovic 17

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 2

```
Poziv: ./a.out -p

STUDENTI.TXT
mr14123 Marko Antic 20
mm13321 Marija Radic 12
ml13011 Ivana Mitrovic 19
ml13066 Pera Simic 15
mv14003 Jovan Jovanovic 17

IZLAZ.TXT
mr14123 Marko Antic 20
ml13011 Ivana Mitrovic 19
mv14003 Jovan Jovanovic 17
ml13066 Pera Simic 15
mm13321 Marija Radic 12
```

[Rešenje 3.37]

**Zadatak 3.38** Definirati strukturu `Datum`. Napisati funkciju koja poredi dva datuma hronološki. Potom, napisati i program koji učitava datume iz datoteke koja se zadaje kao prvi argument komandne linije (ne više od 128 datuma), sortira ih pozivajući funkciju `qsort` iz standardne biblioteke i pozivanjem funkcije `bsearch` iz standardne biblioteke proverava da li datumi učitani sa standardnog ulaza postoje među prethodno unetim datumima. Datumi se učitavaju sve do kraja ulaza.

#### Primer 1

```
Poziv: ./a.out datoteka.txt

DATOTEKA.TXT
1.1.2013.
13.12.2016.
11.11.2011.
3.5.2015.
5.2.2009.
```

```
INTERAKCIJA SA PROGRAMOM:
Unesi sledeci datum: 13.12.2016.
postoji
Unesi sledeci datum: 10.5.2015.
ne postoji
Unesi sledeci datum: 5.2.2009.
postoji
```

## 3.4 Rešenja

### Rešenje 3.1

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <time.h>
   #define MAX 1000000

5
   /* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
7   -lrt zbog funkcije clock_gettime() */

9   /* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
   njemu element x. Pretraga se vrši prostom iteracijom kroz niz. Ako
11  se element pronadje funkcija vraca indeks pozicije na kojoj je
   pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
13  pronadjen u nizu, funkcija vraca -1, kao indikator neuspesne
   pretrage. */
15  int linearna_pretraga(int a[], int n, int x)
   {
17     int i;
       for (i = 0; i < n; i++)
19         if (a[i] == x)
           return i;
21     return -1;
   }

23
   /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
25  pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
   int binarna_pretraga(int a[], int n, int x)
27  {
       int levi = 0;
29     int desni = n - 1;
       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     }
48     /* Ako element x nije pronadjen, vraca se -1 */
49     return -1;
50 }
51
52 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
53    pozicije nadjenog elementa ili -1, ako element nije pronadjen */
54 int interpolaciona_pretraga(int a[], int n, int x)
55 {
56     int levi = 0;
57     int desni = n - 1;
58     int srednji;
59     /* Dokle god je indeks levi levo od indeksa desni... */
60     while (levi <= desni) {
61         /* Ako je trazeni element manji od pocetnog ili veci od
62            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
64            dogodilo da se prilikom izracunavanja indeksa srednji izadje
65            izvan opsega indeksa [levi,desni] */
66         if (x < a[levi] || x > a[desni])
67             return -1;
68         /* 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
70            jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
71            desni). Ova provera je neophodna, jer bi se u suprotnom
72            prilikom izracunavanja indeksa srednji pojavilo deljenje
73            nulom. */
74         else if (a[levi] == a[desni])
75             return levi;
76         /* Racunanje srednjeg indeksa */
77         srednji =
78             levi +
79             ((double) (x - a[levi]) / (a[desni] - a[levi])) *
80             (desni - levi);
81         /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
82            verovatno biti blize trazenoj vrednosti nego da je prosto uvek
83            uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
84            porediti sa pretragom recnika: ako neko trazi rec na slovo 'B
85            ',
86            sigurno nece da otvori recnik na polovini, vec verovatno negde
87            blize pocetku. */
88         /* Ako je element sa indeksom srednji veci od trazenog, tada se
89            trazeni element mora nalaziti u levoj polovini niza */
90         if (x < a[srednji])
91             desni = srednji - 1;
92         /* Ako je element sa indeksom srednji manji od trazenog, tada se
93            trazeni element mora nalaziti u desnoj polovini niza */
94         else if (x > a[srednji])
95             levi = srednji + 1;
96         else
97             /* Ako je element sa indeksom srednji jednak trazenom, onda se
98                pretraga zavrшава na poziciji srednji */

```

```

    return srednji;
99 }
    /* U slucaju neuspesne pretrage vraca se -1 */
101 return -1;
}

103
int main(int argc, char **argv)
105 {
    int a[MAX];
107 int n, i, x;
    struct timespec time1, time2, time3, time4, time5, time6;
109 FILE *f;

    /* Provera argumenata komandne linije */
111 if (argc != 3) {
113     fprintf(stderr,
        "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
        ;
115     exit(EXIT_FAILURE);
    }

117
    /* Dimenzija niza */
119 n = atoi(argv[1]);
    if (n > MAX || n <= 0) {
121     fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
        exit(EXIT_FAILURE);
123 }

125
    /* Broj koji se trazi */
    x = atoi(argv[2]);

127
    /* Elementi niza se generisu slucajno, tako da je svaki sledeci
129 veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
    pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
131 izgledao, seed se postavlja na tekuce vreme u sekundama od Nove
    godine 1970. random()%100 daje brojeve izmedju 0 i 99 */
133 srandom(time(NULL));
    for (i = 0; i < n; i++)
135     a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;

137
    /* Linearna pretraga */
    printf("Linearna pretraga:\n");
139
    /* Vreme proteklo od Nove godine 1970 */
    clock_gettime(CLOCK_REALTIME, &time1);
    i = linearna_pretraga(a, n, x);
141
    /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
    linearnu pretragu */
143 clock_gettime(CLOCK_REALTIME, &time2);
    if (i == -1)
145     printf("Element nije u nizu\n");
    else
147     printf("Element je u nizu na poziciji %d\n", i);

```

```

149  /* Binarna pretraga */
151  printf("Binarna pretraga:\n");
      clock_gettime(CLOCK_REALTIME, &time3);
153  i = binarna_pretraga(a, n, x);
      clock_gettime(CLOCK_REALTIME, &time4);
155  if (i == -1)
      printf("Element nije u nizu\n");
157  else
      printf("Element je u nizu na poziciji %d\n", i);
159
      /* Interpolaciona pretraga */
161  printf("Interpolaciona pretraga:\n");
      clock_gettime(CLOCK_REALTIME, &time5);
163  i = interpolaciona_pretraga(a, n, x);
      clock_gettime(CLOCK_REALTIME, &time6);
165  if (i == -1)
      printf("Element nije u nizu\n");
167  else
      printf("Element je u nizu na poziciji %d\n", i);
169
      /* Podaci o izvršavanju programa bivaju upisani u log fajl */
171  if ((f = fopen("vremena.txt", "a")) == NULL) {
      fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
173      exit(EXIT_FAILURE);
      }
175
      fprintf(f, "Dimenzija niza: %d\n", n);
177      fprintf(f, "\tLinearna: %10ld ns\n",
              (time2.tv_sec - time1.tv_sec) * 1000000000 +
              time2.tv_nsec - time1.tv_nsec);
179      fprintf(f, "\tBinarna: %19ld ns\n",
              (time4.tv_sec - time3.tv_sec) * 1000000000 +
              time4.tv_nsec - time3.tv_nsec);
181      fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
              (time6.tv_sec - time5.tv_sec) * 1000000000 +
              time6.tv_nsec - time5.tv_nsec);
183
185
187  /* Zatvaranje datoteke */
      fclose(f);
189
      return 0;
191 }

```

### Rešenje 3.2

```

#include <stdio.h>
2
#define MAX 1024
4
int lin_pretraga_rek_sufiks(int a[], int n, int x)

```

```

6 {
8     int tmp;
9     /* Izlaz iz rekurzije */
10    if (n <= 0)
11        return -1;
12    /* Ako je prvi element trazeni */
13    if (a[0] == x)
14        return 0;
15    /* Pretraga ostatka niza */
16    tmp = lin_pretraga_rek_sufiks(a + 1, n - 1, x);
17    return tmp < 0 ? tmp : tmp + 1;
18 }
19
20 int lin_pretraga_rek_prefiks(int a[], int n, int x)
21 {
22     /* Izlaz iz rekurzije */
23     if (n <= 0)
24         return -1;
25     /* Ako je poslednji element trazeni */
26     if (a[n - 1] == x)
27         return n - 1;
28     /* Pretraga ostatka niza */
29     return lin_pretraga_rek_prefiks(a, n - 1, x);
30 }
31
32 int bin_pretraga_rek(int a[], int l, int d, int x)
33 {
34     int srednji;
35     if (l > d)
36         return -1;
37     /* Sredisnja pozicija na kojoj se trazi vrednost x */
38     srednji = (l + d) / 2;
39     /* Ako je element na sredisnjoj poziciji trazeni */
40     if (a[srednji] == x)
41         return srednji;
42     /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
43        pretrazuje se desna polovina niza */
44     if (a[srednji] < x)
45         return bin_pretraga_rek(a, srednji + 1, d, x);
46     /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
47        pretrazuje se leva polovina niza */
48     else
49         return bin_pretraga_rek(a, l, srednji - 1, x);
50 }
51
52 int interp_pretraga_rek(int a[], int l, int d, int x)
53 {
54     int p;
55     if (x < a[l] || x > a[d])
56         return -1;
57     if (a[d] == a[l])

```



```

58     return l;
/* Pozicija na kojoj se trazi vrednost x */
60 p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
if (a[p] == x)
62     return p;
if (a[p] < x)
64     return interp_pretraga_rek(a, p + 1, d, x);
else
66     return interp_pretraga_rek(a, l, p - 1, x);
}

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

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

76     /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
77        korisnik pritisne CTRL+D za naznaku kraja */
78     i = 0;
79     printf("Unesite sortiran niz elemenata: ");
80     while (scanf("%d", &a[i]) == 1) {
81         i++;
82     }

83     /* Linearna pretraga */
84     printf("Linearna pretraga\n");
85     indeks = lin_pretraga_rek_sufiks(a, i, x);
86     if (indeks == -1)
87         printf("Element se ne nalazi u nizu.\n");
88     else
89         printf("Pozicija elementa je %d.\n", indeks);

90     /* Binarna pretraga */
91     printf("Binarna pretraga\n");
92     indeks = bin_pretraga_rek(a, 0, i - 1, x);
93     if (indeks == -1)
94         printf("Element se ne nalazi u nizu.\n");
95     else
96         printf("Pozicija elementa je %d.\n", indeks);

97     /* Interpolaciona pretraga */
98     printf("Interpolaciona pretraga\n");
99     indeks = interp_pretraga_rek(a, 0, i - 1, x);
100    if (indeks == -1)
101        printf("Element se ne nalazi u nizu.\n");
102    else
103        printf("Pozicija elementa je %d.\n", indeks);

```

```

110     return 0;
112 }

```

### 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 /* Ucitani niz studenata ce biti sortiran rastuce prema indeksu, jer
   su studenti u datoteci vec sortirani. Iz tog razloga pretraga po
19 indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
   jer nema garancije da postoji uredjenje po prezimenu. */
20
21
22 /* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
   sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
   ako element nije pronadjen. */
23
24 int binarna_pretraga(Student a[], int n, long x)
25 {
26     int levi = 0;
27     int desni = n - 1;
28     int srednji;
29     /* Dokle god je indeks levi levo od indeksa desni */
30     while (levi <= desni) {
31         /* Racuna se srednja pozicija */
32         srednji = (levi + desni) / 2;
33         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
34          se trazi indeks mora nalaziti u levoj polovini niza */
35         if (x < a[srednji].indeks)
36             desni = srednji - 1;
37         /* Ako je pak manji od trazenog, tada se on mora nalaziti u
38          desnoj polovini niza */
39         else if (x > a[srednji].indeks)
40             levi = srednji + 1;
41         else
42             /* Ako je jednak trazenom indeksu x, tada je pronadjen student
43              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 i opcija
63    prosledjuju kao argumenti 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 trazen_indeks = 0;
71     char trazeno_prezime[MAX_DUZINA];
72     int bin_pretraga;

73     /* Provera da li je korisnik prilikom poziva programa prosledio ime
74        datoteke sa informacijama o studentima i opciju pretrage */
75     if (argc != 3) {
76         fprintf(stderr,
77             "Greska: Program se poziva sa %s ime_datoteke opcija\n",
78             argv[0]);
79         exit(EXIT_FAILURE);
80     }

81     /* Provera prosledjene opcije */
82     if (strcmp(argv[2], "-indeks") == 0)
83         bin_pretraga = 1;
84     else if (strcmp(argv[2], "-prezime") == 0)
85         bin_pretraga = 0;
86     else {
87         fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
88         exit(EXIT_FAILURE);
89     }

90     /* Otvaranje datoteke */
91     fin = fopen(argv[1], "r");
92     if (fin == NULL) {
93         fprintf(stderr,
```

```
98         "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
99     exit(EXIT_FAILURE);
100 }
101
102 /* Citanje se vrši sve dok postoji red sa informacijama o studentu
103  */
104 i = 0;
105 while (1) {
106     if (i == MAX_STUDENATA)
107         break;
108     if (fscanf
109         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
110          dosije[i].prezime) != 3)
111         break;
112     i++;
113 }
114 br_studenata = i;
115
116 /* Nakon citanja, datoteka više nije neophodna i zatvara se. */
117 fclose(fin);
118
119 /* Pretraga po indeksu */
120 if (bin_pretraga) {
121     /* Unos indeksa koji se binarno traži u nizu */
122     printf("Unesite indeks studenta čije informacije želite: ");
123     scanf("%ld", &trazen_indeks);
124     i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
125     /* Rezultat binarne pretrage */
126     if (i == -1)
127         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
128     else
129         printf("Indeks: %ld, Ime i prezime: %s %s\n",
130               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
131 }
132
133 /* Pretraga po prezimenu */
134 else {
135     /* Unos prezimena koje se linearno traži u nizu */
136     printf("Unesite prezime studenta čije informacije želite: ");
137     scanf("%s", trazeno_prezime);
138     i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
139     /* Rezultat linearne pretrage */
140     if (i == -1)
141         printf("Ne postoji student sa prezimenom %s\n",
142               trazeno_prezime);
143     else
144         printf("Indeks: %ld, Ime i prezime: %s %s\n",
145               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
146 }
147 return 0;
148 }
```

## Rešenje 3.4

```

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 typedef struct {
9     long indeks;
10    char ime[MAX_DUZINA];
11    char prezime[MAX_DUZINA];
12 } Student;
13
14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
15                                long x)
16 {
17     /* Ako je pozicija elementa na levom kraju veca od pozicije
18        elementa na desnom kraju dela niza koji se pretrazuje, onda se
19        zapravo pretrazuje prazan deo niza. U praznom delu niza nema
20        trazenog elementa pa se vraća -1 */
21     if (levi > desni)
22         return -1;
23     /* Racunanje pozicije srednjeg elementa */
24     int srednji = (levi + desni) / 2;
25     /* Da li je srednji bas onaj trazeni */
26     if (a[srednji].indeks == x) {
27         return srednji;
28     }
29     /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
30        poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
31        je poznato da je niz sortiran po indeksu u rastucem poretku. */
32     if (x < a[srednji].indeks)
33         return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34     /* Inace ga treba traziti u desnoj polovini */
35     else
36         return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37 }
38
39 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
41     /* Ako je niz prazan, vraća se -1 */
42     if (n == 0)
43         return -1;
44     /* Kako se trazi prvi student sa trazenim prezimenom, pocinje se sa
45        prvim studentom u nizu. */
46     if (strcmp(a[0].prezime, x) == 0)
47         return 0;
48     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
49     return i >= 0 ? 1 + i : -1;
50 }

```

```
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 imati argumente jer se ime datoteke i opcija
65    prosledjuju kao argumenti 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     int bin_pretraga;

76     /* Provera da li je korisnik prilikom poziva programa prosledio ime
77        datoteke sa informacijama o studentima i opciju pretrage */
78     if (argc != 3) {
79         fprintf(stderr,
80             "Greska: Program se poziva sa %s ime_datoteke opcija\n",
81             argv[0]);
82         exit(EXIT_FAILURE);
83     }

84     /* Provera prosledjene opcije */
85     if (strcmp(argv[2], "-indeks") == 0)
86         bin_pretraga = 1;
87     else if (strcmp(argv[2], "-prezime") == 0)
88         bin_pretraga = 0;
89     else {
90         fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
91         exit(EXIT_FAILURE);
92     }

94     /* Otvaranje datoteke */
95     fin = fopen(argv[1], "r");
96     if (fin == NULL) {
97         fprintf(stderr,
98             "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
99         exit(EXIT_FAILURE);
100     }
102 }
```

```

/* Citanje se vrši sve dok postoji red sa informacijama o studentu
*/
104 i = 0;
while (1) {
106     if (i == MAX_STUDENATA)
        break;
108     if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
110         dosije[i].prezime) != 3)
        break;
112     i++;
}
114 br_studenata = i;

116 /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
fclose(fin);

118

/* Pretraga po indeksu */
120 if (bin_pretraga) {
    /* Unos indeksa koji se binarno trazi u nizu */
122     printf("Unesite indeks studenta cije informacije želite: ");
    scanf("%ld", &trazen_indeks);
124     i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata,
        trazen_indeks);

126     /* Rezultat binarne pretrage */
    if (i == -1)
128         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
130         printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132 }

/* Pretraga po prezimenu */
134 else {
    /* Unos prezimena koje se linearno trazi u nizu */
136     printf("Unesite prezime studenta cije informacije želite: ");
    scanf("%s", trazeno_prezime);
138     i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
        trazeno_prezime);

140     /* Rezultat linearne pretrage */
    if (i == -1)
142         printf("Ne postoji student sa prezimenom %s\n",
            trazeno_prezime);
    else
144         printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
146 }
148 return 0;
}

```

### Rešenje 3.5

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 /* Struktura koja opisuje tacku u ravni */
7 typedef struct Tacka {
8     float x;
9     float y;
10 } Tacka;
11
12 /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13    pocetka (0,0) */
14 float rastojanje(Tacka A)
15 {
16     return sqrt(A.x * A.x + A.y * A.y);
17 }
18
19 /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u nizu
20    zadatih tacaka t dimenzije n */
21 Tacka najbliza_koordinatnom(Tacka t[], int n)
22 {
23     Tacka najbliza;
24     int i;
25     najbliza = t[0];
26     for (i = 1; i < n; i++) {
27         if (rastojanje(t[i]) < rastojanje(najbliza)) {
28             najbliza = t[i];
29         }
30     }
31     return najbliza;
32 }
33
34 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih tacaka
35    t dimenzije n */
36 Tacka najbliza_x_osi(Tacka t[], int n)
37 {
38
39     Tacka najbliza;
40     int i;
41     najbliza = t[0];
42     for (i = 1; i < n; i++) {
43         if (fabs(t[i].x) < fabs(najbliza.x)) {
44             najbliza = t[i];
45         }
46     }
47     return najbliza;
48 }
49
50 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
51    t dimenzije n */
```



```
Tacka najbliza_y_osi(Tacka t[], int n)
53 {
    Tacka najbliza;
55     int i;
    najbliza = t[0];
57     for (i = 1; i < n; i++) {
        if (fabs(t[i].y) < fabs(najbliza.y)) {
59             najbliza = t[i];
        }
61     }
    return najbliza;
63 }

65 #define MAX 1024

67 int main(int argc, char *argv[])
{
69     FILE *ulaz;
    Tacka tacke[MAX];
71     Tacka najbliza;
    int i, n;
73
    /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75     ime datoteke sa tackama */
    if (argc < 2) {
77         fprintf(stderr,
            "koriscenje programa: %s ime_datoteke\n", argv[0]);
79         return EXIT_FAILURE;
    }
81
    /* Otvaranje datoteke za citanje */
83     ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
85         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[1]);
87         return EXIT_FAILURE;
    }
89
    /* Sve dok ima tacka u datoteci, one bivaju smestane u niz sa
91     tackama; i predstavlja indeks tekuce tacke */
    i = 0;
93     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
        i++;
95     }
    n = i;
97
    /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
99     dodatnih argumenata */
    if (argc == 2)
101        /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
        najbliza = najbliza_koordinatnom(tacke, n);
103    /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
```

```

105     pitanju opcija -x */
106     else if (strcmp(argv[2], "-x") == 0)
107         /* Racuna se rastojanje u odnosu na x osu */
108         najbliza = najbliza_x_osi(tacke, n);
109     /* Ako je u pitanju opcija -y */
110     else if (strcmp(argv[2], "-y") == 0)
111         /* Racuna se rastojanje u odnosu na y osu */
112         najbliza = najbliza_y_osi(tacke, n);
113     else {
114         /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
115            korisnika i prekida se izvršavanje programa */
116         fprintf(stderr, "Pogresna opcija\n");
117         return EXIT_FAILURE;
118     }

119     /* Stampanje 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     }
26     /* Ako je nula u levom delu intervala, nastavlja se pretraga na
27        */

```

```

    [l, s] */
26     if (cos(l) * cos(s) < 0)
        d = s;
28     else
        /* Inace, na intervalu [s, d] */
30         l = s;
    }
32
    /* Stapanje vrednost trazene tacke */
34     printf("%g\n", s);
36     return 0;
}

```

### Rešenje 3.7

```

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

4  #define MAX 256

6  int prvi_veci_od_nule(int niz[], int n)
{
8     /* Granice pretrage */
    int l = 0, d = n - 1;
    int s;
    /* Sve dok je leva manja od desne granice */
12    while (l <= d) {
        /* Racuna se sredisnja pozicija */
14        s = (l + d) / 2;
        /* Ako je broj na toj poziciji veci od nule, a eventualni njegov
16        prethodnik manji ili jednak nuli, pretraga je završena */
        if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
18            return s;
        /* U slucaju broja manjeg ili jednakog nuli, pretražuje se desna
20        polovina niza */
        if (niz[s] <= 0)
22            l = s + 1;
        /* A inace, leva polovina */
24        else
            d = s - 1;
26    }
    return -1;
28 }

30 int main()
{
32     int niz[MAX];
    int n = 0;
34
    /* Unos niza */

```

```

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

```

### Rešenje 3.8

```

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

     /* Unos niza */
     while (scanf("%d", &niz[n]) == 1)
37       n++;

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

```

```
41     return 0;
43 }
```

### Rešenje 3.9

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  unsigned int logaritam_a(unsigned int x)
5  {
6      /* Izlaz iz rekurzije */
7      if (x == 1)
8          return 0;
9      /* Rekurzivni korak */
10     return 1 + logaritam_a(x >> 1);
11 }
12
13 unsigned int logaritam_b(unsigned int x)
14 {
15     /* Binarnom pretragom 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     scanf("%u", &x);
43
44     /* Provera da li je uneti broj pozitivan */
45     if (x == 0) {
```

```

47     fprintf(stderr, "Logaritam od nule nije definisan\n");
    exit(EXIT_FAILURE);
}

49 /* Ispis povratnih vrednosti funkcija */
51 printf("%u %u\n", logaritam_a(x), logaritam_b(x));

53 return 0;
}

```

### Rešenje 3.11

Datoteka 3.1: *sort.h*

```

1  #ifndef _SORT_H_
2  #define _SORT_H_ 1

4  /* Selection sort: Funkcija sortira niz celih brojeva metodom
   sortiranja izborom. Ideja algoritma je sledeca: U svakoj
6  iteraciji pronalazi se najmanji element i premesta se na pocetak
   niza. Dakle, u prvoj iteraciji, pronalazi se najmanji element, i
8  dovodi na nulto mesto u nizu. U i-toj iteraciji najmanjih i
   elemenata su vec na svojim pozicijama, pa se od i+1 do n-1
10  elementa trazi najmanji, koji se dovodi na i+1 poziciju. */
   void selection_sort(int a[], int n);

12

14  /* Insertion sort: Funkcija sortira niz celih brojeva metodom
   sortiranja umetanjem. Ideja algoritma je sledeca: neka je na
   pocetku i-te iteracije niz prvih i elemenata
16  (a[0],a[1],...,a[i-1]) sortirano. U i-toj iteraciji treba element
   a[i] umetnuti na pravu poziciju medju prvih i elemenata tako da se
18  dobije niz duzine i+1 koji je sortiran. Ovo se radi tako sto se
   i-ti element najpre uporedi sa njegovim prvim levim susedom
20  (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 se moze preci na sledecu
22  iteraciju. Ako je a[i-1] vece, tada se zamenjuju a[i] i a[i-1], a
   zatim se proverava da li je potrebno dalje potiskivanje elementa u
24  levo, poredeci ga sa njegovim novim levim susedom. Ovim uzastopnim
   premestanjem se a[i] umece na pravo mesto u nizu. */
26  void insertion_sort(int a[], int n);

28  /* Bubble sort: Funkcija sortira niz celih brojeva metodom mehurica.
   Ideja algoritma je sledeca: prolazi se kroz niz redom poredeci
30  susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
   poretku. Ovim se najveći element poput mehurica istiskuje na
32  "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je potrebno
   ovaj postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih
34  n-1 elemenata niza bez poslednjeg koji je postavljen na pravu
   poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
36  kracim prefiksima niza, cime se jedan po jedan istiskuju

```

```

    elementi na svoje prave pozicije. */
38 void bubble_sort(int a[], int n);

40 /* Selsort: Ovaj algoritam je jednostavno prosirenje sortiranja
   umetanjem koje dopusta direktnu razmenu udaljenih elemenata.
   Prosirenje se sastoji u tome da se kroz algoritam umetanja prolazi
42 vise puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
   koji je manji od n (sto omogucuje razmenu udaljenih elemenata) i
44 tako se dobija h-sortiran niz, tj. niz u kome su elementi na
   rastojanju h sortirani, mada susedni elementi to ne moraju biti. U
46 drugom prolazu kroz isti algoritam sprovodi se isti postupak ali
   za manji korak h. Sa prolazima se nastavlja sve do koraka h = 1, u
48 kome se dobija potpuno sortirani niz. Izbor pocetne vrednosti za
   h, i nacina njegovog smanjivanja menja u nekim slucajevima brzinu
50 algoritma, ali bilo koja vrednost ce rezultovati ispravnim
   sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
52 vrednost 1. */
54 void shell_sort(int a[], int n);

56 /* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
   Sortiranje se vrsi od elementa na poziciji l do onog na poziciji
58 d. Na pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a
   d je jednako poslednjem validnom indeksu u nizu. Funkcija niz
   podeli na dve polovine, levu i desnu, koje zatim rekurzivno
60 sortira. Od ova dva sortirana podniza, sortiran niz se dobija
   ucesljavanjem, tj. istovremenim prolaskom kroz oba niza i izborom
62 trenutnog manjeg elementa koji se smesta u pomocni niz. Na kraju
   algoritma, sortirani elementi su u pomocnom nizu, koji se kopira u
64 originalni niz. */
66 void merge_sort(int a[], int l, int r);

68 /* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
   i r. Njena ideja sortiranja je izbor jednog elementa niza, koji se
70 naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
   svi elementi levo od njega bice manji, a svi desno bice veci od
72 njega. Kako je pivot doveden na svoje mesto, da bi niz bio
   kompletno sortiran, potrebno je sortirati elemente levo (manje) od
74 njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
   manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
76 deo moze se uraditi rekurzivno. */
   void quick_sort(int a[], int l, int r);
78
   #endif

```

Datoteka 3.2: *sort.c*

```

1 #include "sort.h"
3 #define MAX 1000000
5 void selection_sort(int a[], int n)

```

```
{
7   int i, j;
   int min;
9   int pom;

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

23   /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
   su (i) i min razliciti, inace je nepotrebno. */
25   if (min != i) {
       pom = a[i];
27       a[i] = a[min];
       a[min] = pom;
29   }
   }
31 }

33 void insertion_sort(int a[], int n)
{
35   int i, j;

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

41       /* 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]
43       bude sortiran. Indeks j je trenutna pozicija na kojoj se
   element koji se umece nalazi. Petlja se zavrшава ili kada
45       element dodje do levog kraja (j==0) ili kada se naidje na
   element a[j-1] koji je manji od a[j]. */
47       for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
           int temp = a[j];
49           a[j] = a[j - 1];
           a[j - 1] = temp;
51       }
   }
53 }

55 void bubble_sort(int a[], int n)
{
```



```

57  int i, j;
58  int ind;
59
60  for (i = n, ind = 1; i > 1 && ind; i--)
61
62      /* Poput "mehurica" potiskuje se najveći element među elementima
63       od a[0] do a[i-1] na poziciju i-1 upoređujući susedne
64       elemente niza i potiskujući veći u desno */
65      for (j = 0, ind = 0; j < i - 1; j++)
66          if (a[j] > a[j + 1]) {
67              int temp = a[j];
68              a[j] = a[j + 1];
69              a[j + 1] = temp;
70
71              /* Promenljiva ind registruje da je bilo premestanja. Samo u
72              tom slučaju ima smisla ici na sledeću iteraciju, jer ako
73              nije bilo premestanja, znaci da su svi elementi već u
74              dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
75              niza. Algoritam može biti i bez ovoga, sortiranje bi bilo
76              ispravno, ali manje efikasno, jer bi se često nepotrebno
77              vrsila mnoga upoređivanja, kada je već jasno da je
78              sortiranje završeno. */
79              ind = 1;
80          }
81  }
82
83  void shell_sort(int a[], int n)
84  {
85      int h = n / 2, i, j;
86      while (h > 0) {
87          /* Insertion sort sa korakom h */
88          for (i = h; i < n; i++) {
89              int temp = a[i];
90              j = i;
91              while (j >= h && a[j - h] > temp) {
92                  a[j] = a[j - h];
93                  j -= h;
94              }
95              a[j] = temp;
96          }
97          h = h / 2;
98      }
99  }
100
101  void merge_sort(int a[], int l, int d)
102  {
103      int s;
104      static int b[MAX];          /* Pomocni niz */
105      int i, j, k;
106
107      /* Izlaz iz rekurzije */
108      if (l >= d)

```

```
109     return;

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

113

115     /* Rekurzivni pozivi */
    merge_sort(a, l, s);
    merge_sort(a, s + 1, d);

117

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

125     /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
    while (i <= s && j <= d) {
127         if (a[i] < a[j])
            b[k++] = a[i++];
129         else
            b[k++] = a[j++];
131     }

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

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

145     /* Prepisuje se "ucesljani" niz u originalni niz */
    for (k = 0, i = l; i <= d; i++, k++)
147         a[i] = b[k];
    }

149

151     /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
    void swap(int a[], int i, int j)
    {
153         int tmp = a[i];
        a[i] = a[j];
155         a[j] = tmp;
    }

157

159     void quick_sort(int a[], int l, int r)
    {
        int i, pivot_position;
```

```

161  /* Izlaz iz rekurzije -- prazan niz */
163  if (l >= r)
165      return;

167  /* Particionisanje niza. Svi elementi na pozicijama levo od
169  pivot_position (izuzev same pozicije l) su strogo manji od
171  pivota. Kada se pronadje neki element manji od pivota, uvecava
173  se promenljiva pivot_position i na tu poziciju se premesta
175  nadjeni element. Na kraju ce pivot_position zaista biti pozicija
177  na koju treba smestiti pivot, jer ce svi elementi levo od te
179  pozicije biti manji a desno biti veci ili jednaki od pivota. */
181  pivot_position = l;
183  for (i = l + 1; i <= r; i++)
185      if (a[i] < a[l])
187          swap(a, ++pivot_position, i);

189  /* Postavljanje pivota na svoje mesto */
191  swap(a, l, pivot_position);

193  /* Rekurzivno sortiranje elementa manjih od pivota */
195  quick_sort(a, l, pivot_position - 1);
197  /* Rekurzivno sortiranje elementa vecih od pivota */
199  quick_sort(a, pivot_position + 1, r);
201  }

```

Datoteka 3.3: *main.c*

```

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

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

8

9  int main(int argc, char *argv[])
10 {
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     tip_niza == 1 => rastuce sortirani nizovi,    -r opcija

```

```
24     tip_niza == 2 => opadajuće sortirani nizovi, -o opcija
*****
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 /* Učitavanje 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 dužina niza koji treba
72        da se sortira */
73     else {
74         dimenzija = atoi(argv[i]);
```

```

76     if (dimenzija <= 0 || dimenzija > MAX) {
77         fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
78         exit(EXIT_FAILURE);
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     selection_sort(niz, dimenzija);
113 else if (tip_sortiranja == 1)
114     insertion_sort(niz, dimenzija);
115 else if (tip_sortiranja == 2)
116     bubble_sort(niz, dimenzija);
117 else if (tip_sortiranja == 3)
118     shell_sort(niz, dimenzija);
119 else if (tip_sortiranja == 4)
120     merge_sort(niz, 0, dimenzija - 1);
121 else
122     quick_sort(niz, 0, dimenzija - 1);
123
124 /* Ispis elemenata niza */
125 /******

```

```

128     Ovaj deo je iskomentarisan jer vreme potrebno za njegovo
130     izvršavanje ne bi trebalo da bude ukljuceno u vreme izmereno
132     programom time. Takodje, kako je svrha ovog programa da prikaze
134     vremena razlicitih algoritama sortiranja, dimenzije nizova ce
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.

    printf("Sortiran niz je:\n");
    for (i = 0; i < dimenzija; i++)
        printf("%d\n", niz[i]);
    *****/

    return 0;
}

```

### Rešenje 3.12

```

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

#define MAX_DIM 128

/* Funkcija za sortiranje niza karaktera */
void selectionSort(char s[])
{
    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;
        }
    }
}

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

    /* Ako dve niske imaju razlicit broj karaktera onda one nisu
       anagrami */
    if (strlen(s) != strlen(t))
        return 0;
}

```

```

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

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

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

48
    /* Ucitavanje niski sa ulaza */
50  printf("Unesite prvu nisku: ");
    scanf("%s", s);
52  printf("Unesite drugu nisku: ");
    scanf("%s", t);

54
    /* Poziv funkcije */
56  if (anagrami(s, t))
        printf("jesu\n");
58  else
        printf("nisu\n");
60
    return 0;
62 }

```

### Rešenje 3.13

```

1  #include <stdio.h>
    #include "sort.h"
3  #define MAX 256

5  /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
    sortiranom nizu celih brojeva */
7  int najmanje_rastojanje(int a[], int n)
    {
9      int i, min;
        min = a[1] - a[0];
11     for (i = 2; i < n; i++)
            if (a[i] - a[i - 1] < min)
13         min = a[i] - a[i - 1];
        return min;
15 }

17
19  int main()
    {

```

```

21     int i, a[MAX];

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

    /* Za sortiranje niza moze se koristiti bilo koja od funkcija
       sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
27       se selection sort. */
    selection_sort(a, i);

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

35     return 0;
}

```

### Rešenje 3.14

```

1  #include <stdio.h>
   #include "sort.h"
3  #define MAX_DIM 256

   /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
      najvise puta pojavio u tom nizu */
7  int najvise_puta(int a[], int n)
   {
9      int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
      /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
11     for (i = 0; i < n; i = j) {
         br_pojava = 1;
13         for (j = i + 1; j < n && a[i] == a[j]; j++)
             br_pojava++;
15         /* Ispitivanje da li se do tog trenutka i-ti element pojavio
            najvise puta u nizu */
17         if (br_pojava > max_br_pojava) {
             max_br_pojava = br_pojava;
19             i_max_pojava = i;
         }
21     }

    /* Vraca se element koji se najvise puta pojavio u nizu */
23     return a[i_max_pojava];
}

25
int main()
27 {
    int a[MAX_DIM], i;

29     /* Ucitavanje elemenata niza sve do kraja ulaza */
31     i = 0;

```



```

33     while (scanf("%d", &a[i]) != EOF)
        i++;

35     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
        sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
37     se merge sort. */
    merge_sort(a, 0, i - 1);

39

    /* Odredjuje se broj koji se najvise puta pojavio u nizu */
41    printf("%d\n", najvise_puta(a, i));

43    return 0;
}

```

### Rešenje 3.15

```

1  #include <stdio.h>
   #include "sort.h"
3  #define MAX_DIM 256

5  /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
   u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
7  poretku */
   int binarna_pretraga(int a[], int n, int x)
9  {
   int levi = 0, desni = n - 1, srednji;

11     while (levi <= desni) {
13         srednji = (levi + desni) / 2;
         if (a[srednji] == x)
15             return 1;
         else if (a[srednji] > x)
17             desni = srednji - 1;
         else if (a[srednji] < x)
19             levi = srednji + 1;
   }
21     return 0;
}

23
   int main()
25 {
   int a[MAX_DIM], n = 0, zbir, i;

27

   /* Ucitava se trazeni zbir */
29     printf("Unesite trazeni zbir: ");
     scanf("%d", &zbir);

31

   /* Ucitavaju se elementi niza sve do kraja ulaza */
33     i = 0;
     printf("Unesite elemente niza: ");
     while (scanf("%d", &a[i]) != EOF)
35

```

```

37     i++;
    n = i;

39     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
        sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
41     se quick sort. */
    quick_sort(a, 0, n - 1);

43
    for (i = 0; i < n; i++)
45         /* Za i-ti element niza binarno se pretrazuje da li se u ostatku
            niza nalazi element koji sabran sa njim ima ucitanu vrednost
47         zbira */
        if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
49             printf("da\n");
            return 0;
51         }
        printf("ne\n");
53
    return 0;
55 }

```

### Rešenje 3.16

```

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

4 int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
        int dim3)
6 {
    int i = 0, j = 0, k = 0;
8     /* U slucaju da je dimenzija treceg niza manja od neophodne,
        funkcija vraca -1 */
10    if (dim3 < dim1 + dim2)
        return -1;

12
    /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog
        od njih */
14    while (i < dim1 && j < dim2) {
16        if (niz1[i] < niz2[j])
            niz3[k++] = niz1[i++];
18        else
            niz3[k++] = niz2[j++];
20    }
    /* Ostatak prvog niza prepisujemo u treci */
22    while (i < dim1)
        niz3[k++] = niz1[i++];
24
    /* Ostatak drugog niza prepisujemo u treci */
26    while (j < dim2)
        niz3[k++] = niz2[j++];
28    return dim1 + dim2;

```

```

30 }
31
32 int main()
33 {
34     int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
35     int i = 0, j = 0, k, dim3;
36
37     /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
38        Pretpostavka je da na ulazu nece biti vise od MAX_DIM elemenata
39        */
40     printf("Unesite elemente prvog niza: ");
41     while (1) {
42         scanf("%d", &niz1[i]);
43         if (niz1[i] == 0)
44             break;
45         i++;
46     }
47     printf("Unesite elemente drugog niza: ");
48     while (1) {
49         scanf("%d", &niz2[j]);
50         if (niz2[j] == 0)
51             break;
52         j++;
53     }
54
55     /* Poziv trazene funkcije */
56     dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);
57
58     /* Ispis niza */
59     for (k = 0; k < dim3; k++)
60         printf("%d ", niz3[k]);
61     printf("\n");
62
63     return 0;
64 }

```

### Rešenje 3.17

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[])
6 {
7     FILE *fin1 = NULL, *fin2 = NULL;
8     FILE *fout = NULL;
9     char ime1[11], ime2[11];
10    char prezime1[16], prezime2[16];
11    int kraj1 = 0, kraj2 = 0;
12
13    /* Ako nema dovoljno argumenata komandne linije */

```

```
14  if (argc < 3) {
    fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0])
    ;
16  exit(EXIT_FAILURE);
    }

18  /* Otvaranje datoteke zadate prvim argumentom komandne linije */
20  fin1 = fopen(argv[1], "r");
    if (fin1 == NULL) {
22      fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
        exit(EXIT_FAILURE);
24  }

26  /* Otvaranje datoteke zadate drugim argumentom komandne linije */
    fin2 = fopen(argv[2], "r");
28  if (fin2 == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
30      exit(EXIT_FAILURE);
    }

32  /* Otvaranje datoteke za upis rezultata */
34  fout = fopen("ceo-tok.txt", "w");
    if (fout == NULL) {
36      fprintf(stderr,
        "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
38      exit(EXIT_FAILURE);
    }

40  /* Citanje narednog studenta iz prve datoteke */
42  if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
        kraj1 = 1;

44  /* Citanje narednog studenta iz druge datoteke */
46  if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
        kraj2 = 1;

48  /* Sve dok nije dostignut kraj neke datoteke */
50  while (!kraj1 && !kraj2) {
    if (strcmp(ime1, ime2) < 0) {
52      /* Ime i prezime iz prve datoteke je leksikografski ranije, i
        biva upisano u izlaznu datoteku */
54      fprintf(fout, "%s %s\n", ime1, prezime1);
        /* Citanje narednog studenta iz prve datoteke */
56      if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
            kraj1 = 1;
58    } else {
        /* Ime i prezime iz druge datoteke je leksikografski ranije, i
        biva upisano u izlaznu datoteku */
60      fprintf(fout, "%s %s\n", ime2, prezime2);
        /* Citanje narednog studenta iz druge datoteke */
62      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
            kraj2 = 1;
64    }
```

```

    }
66 }

68 /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
    druge datoteke, onda ima jos studenata u prvoj datoteci, koje
70 treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
    */
72 while (!kraj1) {
    fprintf(fout, "%s %s\n", ime1, prezime1);
74     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)
84                 kraj2 = 1;
    }
86
    /* Zatvaranje datoteka */
88     fclose(fin1);
    fclose(fin2);
90     fclose(fout);

92     return 0;
}

```

### Rešenje 3.18

```

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

```

```
21 /* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
    pocetka */
23 void sortiraj_po_rastojanju(Tacka t[], int n)
    {
25     int min, i, j;
        Tacka tmp;
27
        for (i = 0; i < n - 1; i++) {
29             min = i;
                for (j = i + 1; j < n; j++) {
31                 if (rastojanje(t[j]) < rastojanje(t[min])) {
                    min = j;
33                 }
                }
35             if (min != i) {
                tmp = t[i];
37                 t[i] = t[min];
                    t[min] = tmp;
39             }
        }
41 }

43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
void sortiraj_po_x(Tacka t[], int n)
45 {
47     int min, i, j;
        Tacka tmp;
49
        for (i = 0; i < n - 1; i++) {
            min = i;
51             for (j = i + 1; j < n; j++) {
                if (abs(t[j].x) < abs(t[min].x)) {
53                     min = j;
                }
55             }
            if (min != i) {
57                 tmp = t[i];
                    t[i] = t[min];
59                 t[min] = tmp;
            }
61 }
    }

63 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
void sortiraj_po_y(Tacka t[], int n)
65 {
67     int min, i, j;
        Tacka tmp;
69
        for (i = 0; i < n - 1; i++) {
71             min = i;
```

```
73     for (j = i + 1; j < n; j++) {
74         if (abs(t[j].y) < abs(t[min].y)) {
75             min = j;
76         }
77     }
78     if (min != i) {
79         tmp = t[i];
80         t[i] = t[min];
81         t[min] = tmp;
82     }
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     /* Proveravanje broja argumenata komandne linije: ocekuje se ime
92        izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
93        datoteke, tj. 4 argumenta */
94     if (argc != 4) {
95         fprintf(stderr,
96             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
97         return 0;
98     }

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    }

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    }

117    /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
118       koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
119       brojacem i. */
120    i = 0;
121    while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
122        i++;
123    }
```

```

125  /* Ukupan broj procitanih tacaka */
    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
    sortiranja */
131  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 koorinatnog 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 3.19

```

#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;
    Gradjanin pom;

    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 }

/* Funkcija sortira niz gradjana rastuce po prezimenima */
void sort_prezime(Gradjanin a[], int n)
40 {
    int i, j;
42     int min;
    Gradjanin pom;

    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 }

/* 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

74 int main()
{
76     Gradjanin spisak1[MAX], spisak2[MAX];
    int isti_rbr = 0;
78     int i, n;
    FILE *fp = NULL;

80     /* Otvaranje datoteke */
82     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
        fprintf(stderr,
84             "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
        exit(EXIT_FAILURE);
86     }

88     /* Citanje sadrzaja */
    for (i = 0;
90         fscanf(fp, "%s %s", spisak1[i].ime,
                spisak1[i].prezime) != EOF; i++)
92         spisak2[i] = spisak1[i];
    n = i;

94     /* Zatvaranje datoteke */
96     fclose(fp);

98     sort_ime(spisak1, n);

100    /******
    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
102    sortiranih nizova. Koriscen je samo u fazi testiranja programa.

    printf("Biracki spisak [uredjen prema imenima]:\n");
    for(i=0; i<n; i++)
104         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
    *****/

108    sort_prezime(spisak2, n);

110    /******
    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
112    sortiranih nizova. Koriscen je samo u fazi testiranja programa.

    printf("Biracki spisak [uredjen prema prezimenima]:\n");
    for(i=0; i<n; i++)
114         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
    *****/
118

```

```

120  /* Linearno pretrazivanje nizova */
    for (i = 0; i < n; i++)
122      if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
          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 3.21

```

1  #include <stdio.h>
   #include <string.h>
3  #include <ctype.h>

5  #define MAX_BR_RECII 128
   #define MAX_DUZINA_RECII 32

7
   /* Funkcija koja izracunava broj suglasnika u reci */
9  int broj_suglasnika(char s[])
   {
11     char c;
       int i;
13     int suglasnici = 0;
       /* Prolaz karakter po karakter kroz zadatu nisku */
15     for (i = 0; s[i]; i++) {
         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
17         pokriven slucaj i malih i velikih suglasnika. */
         if (isalpha(s[i])) {
19             c = toupper(s[i]);
             /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika.
21             */
             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
                 suglasnici++;
23         }
       }
25     /* Vraca se izracunata vrednost */
       return suglasnici;
27 }

```

```

29 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
    duzini reci se mora proslediti zbog pravilnog upravljanja
31 memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_RECI], int n)
33 {
    int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35     duzina_j, duzina_min;
    char tmp[MAX_DUZINA_RECI];
37     for (i = 0; i < n - 1; i++) {
        min = i;
39         for (j = i; j < n; j++) {
            /* Prvo se uporedjuje broj suglasnika */
41             broj_suglasnika_j = broj_suglasnika(reci[j]);
            broj_suglasnika_min = broj_suglasnika(reci[min]);
43             if (broj_suglasnika_j < broj_suglasnika_min)
                min = j;
45             else if (broj_suglasnika_j == broj_suglasnika_min) {
                /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
47                 se duzine */
                duzina_j = strlen(reci[j]);
49                 duzina_min = strlen(reci[min]);

                if (duzina_j < duzina_min)
                    min = j;
51                 else
                    /* Ako reci imaju i isti broj suglasnika i iste duzine,
53                     uporedjuju se leksikografski */
                    if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
55                         min = j;
57             }
        }
59     }
    if (min != i) {
61         strcpy(tmp, reci[min]);
        strcpy(reci[min], reci[i]);
63         strcpy(reci[i], tmp);
    }
65 }

67 int main()
69 {
    FILE *ulaz;
71     int i = 0, n;

73     /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
        a drugi maksimalnu duzinu pojedinačne reci */
75     char reci[MAX_BR_RECI][MAX_DUZINA_RECI];

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

```

```

81         "Greska prilikom otvaranja datoteke niske.txt!\n");
82     return 0;
83 }

84 /* Sve dok se moze procitati sledeca rec */
85 while (fscanf(ulaz, "%s", reci[i]) != EOF) {
86     /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
87        prekida se učitavanje */
88     if (i == MAX_BR_RECII)
89         break;
90     /* Priprema brojac za narednu iteraciju */
91     i++;
92 }

93

94 /* n je duzina niza reci i predstavlja poslednju vrednost
95    koriscenog brojac */
96 n = i;
97 /* Poziv funkcije za sortiranje reci */
98 sortiraj_reci(reci, n);

99

100 /* Ispis sortiranog niza reci */
101 for (i = 0; i < n; i++) {
102     printf("%s ", reci[i]);
103 }
104 printf("\n");

105

106 /* Zatvaranje datoteke */
107 fclose(ulaz);

108

109 return 0;
110 }

```

### Rešenje 3.22

```

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

#define MAX_ARTIKALA 100000

/* Struktura koja predstavlja jedan artikal */
typedef struct art {
    long kod;
    char naziv[20];
    char proizvođjac[20];
    float cena;
} Artikal;

/* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
   traženim bar kodom */
int binarna_pretraga(Artikal a[], int n, long x)

```

```
18 {
19     int levi = 0;
20     int desni = n - 1;

22     /* Dokle god je indeks levi levo od indeksa desni */
23     while (levi <= desni) {
24         /* Racuna se sredisnji indeks */
25         int srednji = (levi + desni) / 2;
26         /* Ako je sredisnji element veci od trazenog, tada se trazeni
27            mora nalaziti u levoj polovini niza */
28         if (x < a[srednji].kod)
29             desni = srednji - 1;
30         /* Ako je sredisnji element manji od trazenog, tada se trazeni
31            mora nalaziti u desnoj polovini niza */
32         else if (x > a[srednji].kod)
33             levi = srednji + 1;
34         else
35             /* Ako je sredisnji element jednak trazenom, tada je artikal sa
36                bar kodom x pronadjen na poziciji srednji */
37             return srednji;
38     }
39     /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
40     return -1;
41 }

42
43 /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44 void selection_sort(Artikal a[], int n)
45 {
46     int i, j;
47     int min;
48     Artikal pom;

50     for (i = 0; i < n - 1; i++) {
51         min = i;
52         for (j = i + 1; j < n; j++)
53             if (a[j].kod < a[min].kod)
54                 min = j;
55         if (min != i) {
56             pom = a[i];
57             a[i] = a[min];
58             a[min] = pom;
59         }
60     }
61 }

62
63 int main()
64 {
65     Artikal asortiman[MAX_ARTIKALA];
66     long kod;
67     int i, n;
68     float racun;
```

```

70 FILE *fp = NULL;

72 /* Otvaranje datoteke */
73 if ((fp = fopen("artikli.txt", "r")) == NULL) {
74     fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
75     exit(EXIT_FAILURE);
76 }

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

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

86 n = i;

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

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

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

```

```

122     break;
123     /* Trenutni racun novog kupca */
124     racun = 0;
125     /* Za sve artikle trenutnog kupca */
126     while (1) {
127         /* Vrsi se njihov pronalazak u nizu */
128         if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
129             printf("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
130         } else {
131             printf("\tTrazili ste:\t%s %s %12.2f\n",
132                 asortiman[i].naziv, asortiman[i].proizvodjac,
133                 asortiman[i].cena);
134             /* I dodavanje na ukupan racun */
135             racun += asortiman[i].cena;
136         }
137         /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
138            nema vise artikla */
139         printf("Unesite kod artikla [ili 0 za prekid]: \t");
140         scanf("%ld", &kod);
141         if (kod == 0)
142             break;
143     }
144     /* Stampanje ukupnog racuna trenutnog kupca */
145     printf("\n\tUKUPNO: %.21f dinara.\n\n", racun);
146 }
147
148 printf("Kraj rada kase!\n");
149
150 exit(EXIT_SUCCESS);
151 }

```

### Rešenje 3.23

```

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

```



```
19  int i, j;
20  int min;
21  Student pom;

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

29      if (min != i) {
30          pom = niz[min];
31          niz[min] = niz[i];
32          niz[i] = pom;
33      }
34  }
35 }

37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
38    zadataka opadajuće, a ukoliko neki studenti imaju isti broj
39    uradjenih zadataka sortiraju se po dužini imena rastuće. */
40 void sort_zadatke_pa_imena(Student niz[], int n)
41 {
42     int i, j;
43     int max;
44     Student pom;
45     for (i = 0; i < n - 1; i++) {
46         max = i;
47         for (j = i + 1; j < n; j++)
48             if (niz[j].zadaci > niz[max].zadaci)
49                 max = j;
50             else if (niz[j].zadaci == niz[max].zadaci
51                     && strlen(niz[j].ime) < strlen(niz[max].ime))
52                 max = j;
53         if (max != i) {
54             pom = niz[max];
55             niz[max] = niz[i];
56             niz[i] = pom;
57         }
58     }
59 }

61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
62    su bili opadajuće. Ukoliko neki studenti imaju isti broj casova,
63    sortiraju se opadajuće po broju uradjenih zadataka, a ukoliko se
64    i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65    prezimenu opadajuće. */
66 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
67 {
68     int i, j;
69     int max;
70     Student pom;
```

```

71     for (i = 0; i < n - 1; i++) {
72         max = i;
73         for (j = i + 1; j < n; j++)
74             if (niz[j].prisustvo > niz[max].prisustvo)
75                 max = j;
76             else if (niz[j].prisustvo == niz[max].prisustvo
77                     && niz[j].zadaci > niz[max].zadaci)
78                 max = j;
79             else if (niz[j].prisustvo == niz[max].prisustvo
80                     && niz[j].zadaci == niz[max].zadaci
81                     && strcmp(niz[j].prezime, niz[max].prezime) > 0)
82                 max = j;
83         if (max != i) {
84             pom = niz[max];
85             niz[max] = niz[i];
86             niz[i] = pom;
87         }
88     }
89 }

91 int main(int argc, char *argv[])
92 {
93     Student praktikum[MAX];
94     int i, br_studenata = 0;
95
96     FILE *fp = NULL;
97
98     /* Otvaranje datoteke za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
100         fprintf(stderr, "Neuspesno otvaranje datoteke aktivnost.txt.\n");
101         exit(EXIT_FAILURE);
102     }
103
104     /* Ucitavanje sadrzaja */
105     for (i = 0;
106          fscanf(fp, "%s%d", praktikum[i].ime,
107                praktikum[i].prezime, &praktikum[i].prisustvo,
108                &praktikum[i].zadaci) != EOF; i++);
109
110     /* Zatvaranje datoteke */
111     fclose(fp);
112     br_studenata = i;
113
114     /* Kreiranje prvog spiska studenata po prvom kriterijumu */
115     sort_ime_leksikografski(praktikum, br_studenata);
116     /* Otvaranje datoteke za pisanje */
117     if ((fp = fopen("dat1.txt", "w")) == NULL) {
118         fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
119         exit(EXIT_FAILURE);
120     }
121     /* Upis niza u datoteku */
122     fprintf
123         (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");

```

```

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

```

### Rešenje 3.24

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4

```

```

6      #define KORAK 10

8      /* Struktura koja opisuje jednu pesmu */
9      typedef struct {
10         char *izvodjac;
11         char *naslov;
12         int broj_gledanja;
13     } Pesma;

14     /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
15        rad qsort funkcije) */
16     int uporedi_gledanost(const void *pp1, const void *pp2)
17     {
18         Pesma *p1 = (Pesma *) pp1;
19         Pesma *p2 = (Pesma *) pp2;
20
21         return p2->broj_gledanja - p1->broj_gledanja;
22     }

23     /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
24        funkcije) */
25     int uporedi_naslove(const void *pp1, const void *pp2)
26     {
27         Pesma *p1 = (Pesma *) pp1;
28         Pesma *p2 = (Pesma *) pp2;
29
30         return strcmp(p1->naslov, p2->naslov);
31     }

32     /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
33        qsort funkcije) */
34     int uporedi_izvodjace(const void *pp1, const void *pp2)
35     {
36         Pesma *p1 = (Pesma *) pp1;
37         Pesma *p2 = (Pesma *) pp2;
38
39         return strcmp(p1->izvodjac, p2->izvodjac);
40     }

41     int main(int argc, char *argv[])
42     {
43         FILE *ulaz;
44         Pesma *pesme;
45
46         int alocirano_za_pesme;
47         int i;
48
49         int n;
50         int j, k;
51         char c;
52         int alocirano;
53
54         /* Pokazivac na deo memorije za
55            cuvanje pesama */
56         /* Broj mesta alociranih za pesme */
57         /* Redni broj pesme cije se
58            informacije citaju */
59         /* Ukupan broj pesama */
60         /* Broj mesta alociranih za propratne
61            informacije o pesmama */

```

```
int broj_gledanja;

58
/* Priprema datoteke za citanje */
60 ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
if (ulaz == NULL) {
62     printf("Greska pri otvaranju ulazne datoteke!\n");
    return 0;
64 }

/* Citanje informacija o pesmama */
66 pesme = NULL;
68 alocirano_za_pesme = 0;
i = 0;
70
while (1) {
72
    /* Proverava da li je dostignut kraj datoteke */
74     c = fgetc(ulaz);
    if (c == EOF) {
76         /* Nema vise sadrzaja za citanje */
            break;
78     } else {
        /* Inace, vracamo procitani karakter nazad */
80         ungetc(c, ulaz);
    }

82
    /* Provera da li postoji dovoljno memorije za citanje nove pesme
    */
84     if (alocirano_za_pesme == i) {

86         /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
            novih KORAK mesta */
88         alocirano_za_pesme += KORAK;
        pesme =
90             (Pesma *) realloc(pesme,
                                alocirano_za_pesme * sizeof(Pesma));
92
        /* Proverava da li je nova memorija uspesno realocirana */
94         if (pesme == NULL) {
            /* Ako nije ispisuje se obavestenje */
96             printf("Problem sa alokacijom memorije!\n");
            /* I oslobadja sva memorija zauzeta do ovog koraka */
98             for (k = 0; k < i; k++) {
                free(pesme[k].izvodjac);
100                free(pesme[k].naslov);
            }
102            free(pesme);
            return 0;
104        }
    }

106
    /* Ako jeste, nastavlja se sa citanjem pesama ... */
```

```
108      /* Cita se ime izvodjaca */
109      j = 0;                                /* Pozicija na koju treba smestiti
110                                           procitani karakter */
111      alocirano = 0;                        /* Broj alociranih mesta */
112      pesme[i].izvodjac = NULL;           /* Memorija za smestanje procitanih
113                                           karaktera */
114
115      /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
116       izvodjaca) citaju se karakteri iz datoteke */
117      while ((c = fgetc(ulaz)) != ' ') {
118          /* Proverav da li postoji dovoljno memorije za smestanje
119             procitanog karaktera */
120          if (j == alocirano) {
121
122              /* Ako ne, ako je potrosena sva alocirana memorija, alocira
123                 se novih KORAK mesta */
124              alocirano += KORAK;
125              pesme[i].izvodjac =
126                  (char *) realloc(pesme[i].izvodjac,
127                                  alocirano * sizeof(char));
128
129              /* Provera da li je nova alokacija uspesna */
130              if (pesme[i].izvodjac == NULL) {
131                  /* Ako nije oslobadja se sva memorija zauzeta do ovog
132                     koraka */
133                  for (k = 0; k < i; k++) {
134                      free(pesme[k].izvodjac);
135                      free(pesme[k].naslov);
136                  }
137                  free(pesme);
138                  /* I prekida sa izvršavanjem programa */
139                  return 0;
140              }
141
142          }
143          /* Ako postoji dovoljno memorije, smestamo procitani karakter
144          */
145          pesme[i].izvodjac[j] = c;
146          j++;
147          /* I nastavlja se sa citanjem */
148      }
149
150      /* Upis terminirajuće nule na kraj reci */
151      pesme[i].izvodjac[j] = '\\0';
152
153      /* Preskace se karakter - */
154      fgetc(ulaz);
155
156      /* Preskace se razmak */
157      fgetc(ulaz);
158
159      /* Cita se naslov pesme */
```

```

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

```

```

210     broj_gledanja = broj_gledanja * 10 + (c - '0');
211 }
212 pesme[i].broj_gledanja = broj_gledanja;
213
214 /* Prelazi se na citanje sledece pesme */
215 i++;
216 }
217
218 /* Informacija o broju procitanih pesama */
219 n = i;
220 /* Zatvaranje nepotrebne datoteke */
221 fclose(ulaz);
222
223 /* Analiza argumenta komandne linije */
224 if (argc == 1) {
225     /* Nema dodatnih opcija => sortiranje po broju gledanja */
226     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
227 } else {
228     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
229         /* Sortiranje po naslovu */
230         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
231     } else {
232         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
233             /* Sortiranje po izvodjacu */
234             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
235         } else {
236             printf("Nedozvoljeni argumenti!\n");
237             free(pesme);
238             return 0;
239         }
240     }
241 }
242
243 /* Ispis rezultata */
244 for (i = 0; i < n; i++) {
245     printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
246           pesme[i].broj_gledanja);
247 }
248
249 /* Oslobadjanje memorije */
250 for (i = 0; i < n; i++) {
251     free(pesme[i].izvodjac);
252     free(pesme[i].naslov);
253 }
254 free(pesme);
255
256 return 0;
257 }

```

## Rešenje 3.27



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
6    kolona */
7 int zbir_vrste(int **a, int v, int m)
8 {
9     int i, zbir = 0;
10
11     for (i = 0; i < m; i++) {
12         zbir += a[v][i];
13     }
14     return zbir;
15 }
16
17 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
18    osnovu zbira koriscenjem selection sort algoritma */
19 void sortiraj_vrste(int **a, int n, int m)
20 {
21     int i, j, min;
22
23     for (i = 0; i < n - 1; i++) {
24         min = i;
25         for (j = i + 1; j < n; j++) {
26             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
27                 min = j;
28             }
29         }
30         if (min != i) {
31             int *tmp;
32             tmp = a[i];
33             a[i] = a[min];
34             a[min] = tmp;
35         }
36     }
37 }
38
39 int main(int argc, char *argv[])
40 {
41     int **a;
42     int n, m;
43
44     /* Unos dimenzija matrice */
45     printf("Unesite dimenzije matrice: ");
46     scanf("%d %d", &n, &m);
47
48     /* Alokacija memorije */
49     a = alociraj_matricu(n, m);
50
51     /* Ucitavanje elementa matrice */
```

```
52 printf("Unesite elemente matrice po vrstama:\n");
   ucitaj_matricu(a, n, m);
54
   /* Poziv funkcije koja sortira vrste matrice prema zbiru */
56 sortiraj_vrste(a, n, m);
   /* Ispis rezultujuće matrice */
58 printf("Sortirana matrica je:\n");
   ispisi_matricu(a, n, m);
60
   /* Oslobadjanje memorije */
62 a = dealociraj_matricu(a, n);
64
   return 0;
66 }
```

### Rešenje 3.30

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
   #define MAX 100
7
   /* Funkcija poredjenja dva cela broja */
9  int compare_int(const void *a, const void *b)
   {
11     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
       se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
13
       /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
15         oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */

       int b1 = *((int *) a);
       int b2 = *((int *) b);
17
19         if (b1 > b2)
21             return 1;
       else if (b1 < b2)
23             return -1;
       else
25             return 0;
   }
27
   int compare_int_desc(const void *a, const void *b)
29 {
   /* Za obrnuti poredak treba samo oduzimati a od b */
31 /* return *((int *)b) - *((int *)a); */

   /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
33
```

```
    funkcija */
35     return -compare_int(a, b);
    }
37
int main()
39 {
    size_t n;
41     int i, x;
    int a[MAX], *p = NULL;
43
    /* Unos dimenzije */
45     printf("Uneti dimenziju niza: ");
    scanf("%ld", &n);
47     if (n > MAX)
        n = MAX;
49
    /* Unos elementa niza */
51     printf("Uneti elemente niza:\n");
    for (i = 0; i < n; i++)
53         scanf("%d", &a[i]);

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

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

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

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

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

85     return 0;
```

```
}

```

### Rešenje 3.31

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <search.h>
5
6  #define MAX 100
7
8  /* Funkcija racuna broj delilaca broja x */
9  int no_of_deviders(int x)
10 {
11     int i;
12     int br;
13
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 izaslo 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
33     return br;
34 }
35
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);
43
44     if (n_d_a > n_d_b)
45         return 1;
46     else if (n_d_a < n_d_b)
47         return -1;

```

```

49     else
        return 0;
51 }
53
55 int main()
56 {
57     size_t n;
58     int i;
59     int a[MAX];
60
61     /* Unos dimenzije */
62     printf("Uneti dimenziju niza: ");
63     scanf("%ld", &n);
64     if (n > MAX)
65         n = MAX;
66
67     /* Unos elementa niza */
68     printf("Uneti elemente niza:\n");
69     for (i = 0; i < n; i++)
70         scanf("%d", &a[i]);
71
72     /* Sortiranje niza celih brojeva prema broju delilaca */
73     qsort(a, n, sizeof(int), &compare_no_deviders);
74
75     /* Prikaz sortiranog niza */
76     printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
77     for (i = 0; i < n; i++)
78         printf("%d ", a[i]);
79     putchar('\n');
80
81     return 0;
82 }

```

### Rešenje 3.32

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <search.h>
5
6  #define MAX_NISKI 1000
7  #define MAX_DUZINA 30
8
9  /*****
10   Niz nizova karaktera ovog potpisa
11   char niske[3][4];
12   se moze graficki predstaviti ovako:
13   -----
14   | a | b | c | \0 | | d | e | \0 |   | f | g | h | \0 |
15   -----
16   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za

```

```

17  svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
19  nalazi na adresi koja je za 4 veca od prve reci, a za 4 manja od
    adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
    i ona je tipa char*.

21
    Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23  koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
    reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
    nad njima.
27  *****/
    int poredi_leksikografski(const void *a, const void *b)
29  {
    return strcmp((char *) a, (char *) b);
31  }

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

39
    int main()
41  {
    int i;
43  size_t n;
    FILE *fp = NULL;
45  char niske[MAX_NISKI][MAX_DUZINA];
    char *p = NULL;
47  char x[MAX_DUZINA];

49  /* Otvaranje datoteke */
    if ((fp = fopen("niske.txt", "r")) == NULL) {
51  fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
    exit(EXIT_FAILURE);
53  }

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

57
    /* Zatvaranje datoteke */
59  fclose(fp);
    n = i;

61
    /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
63  prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
    niske po duzini */
65  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);

67  printf("Leksikografski sortirane niske:\n");
    for (i = 0; i < n; i++)

```

```

69     printf("%s ", niske[i]);
    printf("\n");
71
    /* Unos trazene niske */
73     printf("Uneti trazenu nisku: ");
    scanf("%s", x);
75
    /* Binarna pretraga */
77     /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
       je niz vec sortiran leksikografski. */
79     p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
                  &poredi_leksikografski);
81
    if (p != NULL)
83         printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
                  p, (p - (char *) niske) / MAX_DUZINA);
85     else
        printf("Niska nije pronadjena u nizu\n");
87
    /* Sortiranje po duzini */
89     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

    printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
93         printf("%s ", niske[i]);
    printf("\n");
95
    /* Linearna pretraga */
97     p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
                &poredi_leksikografski);
99
    if (p != NULL)
101        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
                 p, (p - (char *) niske) / MAX_DUZINA);
103    else
        printf("Niska nije pronadjena u nizu\n");
105
    exit(EXIT_SUCCESS);
107 }

```

### Rešenje 3.33

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <search.h>
5
   #define MAX_NISKI 1000
7  #define MAX_DUZINA 30
9
   /*****

```

```

11  Niz pokazivaca na karaktere ovog potpisa
    char *niske[3];
    posle alokacije u main-u se moze graficki predstaviti ovako:
13  ----
    | X | -----> | a | b | c | \0|
15  ----
    | Y | -----> | d | e | \0|
17  ----
    | Z | -----> | f | g | h | \0|
19  ----

    Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
21  njegov element pokazivac koji pokazuje na alocirane karaktere i-te
    reci. Njegov tip je char*.

23
    Kako pokazivaci a i b u sledecoj funkciji sadrže adrese elemenata
25  koji trebaju biti upoređeni (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
27  moraju i kastovati. Da bi se leksikografski uporedili elementi X i
    Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
29  u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
    kastovani na odgovarajuci tip.

31  *****/
    int poredi_leksikografski(const void *a, const void *b)
33  {
        return strcmp(*(char **) a, *(char **) b);
35  }

37  /* Funkcija slicna prethodnoj, osim sto elemente ne upoređuje
    leksikografski, vec po duzini */
    int poredi_duzine(const void *a, const void *b)
39  {
        return strlen(*(char **) a) - strlen(*(char **) b);
41  }

43
    /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
    element u nizu sa kojim se poredi, pa njega treba kastovati na
45  char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
    ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
47  main funkciji je to x, koji je tipa char*, tako da pokazivac a
    ovde samo treba kastovati i ne dereferencirati. */
49  int poredi_leksikografski_b(const void *a, const void *b)
51  {
        return strcmp((char *) a, *(char **) b);
53  }

55  int main()
    {
57      int i;
        size_t n;
        FILE *fp = NULL;
        char *niske[MAX_NISKI];
61      char **p = NULL;

```



```
char x[MAX_DUZINA];

63
/* Otvaranje datoteke */
65 if ((fp = fopen("niske.txt", "r")) == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
67     exit(EXIT_FAILURE);
}

69
/* Citanje sadrzaja datoteke */
71 i = 0;
while (fscanf(fp, "%s", x) != EOF) {
73     /* Alociranje dovoljne memorije za i-tu nisku */
    if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
75         fprintf(stderr, "Greska pri alociranju niske\n");
        exit(EXIT_FAILURE);
77     }
    /* Kopiranje procitane niske na svoje mesto */
79     strcpy(niske[i], x);
    i++;
81 }

83
/* Zatvaranje datoteke */
fclose(fp);
85 n = i;

87
/* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
   prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
   niske po duzini */
89 qsort(niske, n, sizeof(char *), &poredi_leksikografski);

91
printf("Leksikografski sortirane niske:\n");
93 for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
95 printf("\n");

97
/* Unos trazene niske */
printf("Uneti trazenu nisku: ");
99 scanf("%s", x);

101
/* Binarna pretraga */
p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
103 if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
105         *p, p - niske);
else
107     printf("Niska nije pronadjena u nizu\n");

109
/* Linearna pretraga */
p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
111 if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
113         *p, p - niske);
```

```
115     else
116         printf("Niska nije pronadjena u nizu\n");
117
118     /* Sortiramo po duzini */
119     qsort(niske, n, sizeof(char *), &poredi_duzine);
120
121     printf("Niske sortirane po duzini:\n");
122     for (i = 0; i < n; i++)
123         printf("%s ", niske[i]);
124     printf("\n");
125
126     /* Oslobadjanje zauzete memorije */
127     for (i = 0; i < n; i++)
128         free(niske[i]);
129
130     exit(EXIT_SUCCESS);
131 }
```

### Rešenje 3.34

```
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 poredi1(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     {
29         /* Ako su jednaki po broju bodova, treba ih uporediti po
30            prezimenu */
31         return strcmp(prvi->prezime, drugi->prezime);
32     }
33 }
```

```
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 poredi2(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.
44    Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
45    parametar je element niza cije se prezime poredi. */
46 int poredi3(const void *a, const void *b)
47 {
48     char *prezime = (char *) a;
49     Student *s = (Student *) b;
50     return strcmp(prezime, s->prezime);
51 }
52
53 int main(int argc, char *argv[])
54 {
55     Student kolokvijum[MAX];
56     int i;
57     size_t br_studenata = 0;
58     Student *nadjen = NULL;
59     FILE *fp = NULL;
60     int bodovi;
61     char prezime[21];
62
63     /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64        informacija korisniku kako se program koristi i prekida se
65        izvršavanje. */
66     if (argc < 2) {
67         fprintf(stderr,
68             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
69             argv[0]);
70         exit(EXIT_FAILURE);
71     }
72
73     /* Otvaranje datoteke */
74     if ((fp = fopen(argv[1], "r")) == NULL) {
75         fprintf(stderr, "Neuspješno otvaranje datoteke %s\n", argv[1]);
76         exit(EXIT_FAILURE);
77     }
78
79     /* Učitavanje sadržaja */
80     for (i = 0;
81          fscanf(fp, "%s%s%d", kolokvijum[i].ime,
82                kolokvijum[i].prezime,
83                &kolokvijum[i].bodovi) != EOF; i++);
```

```
84  /* Zatvaranje datoteke */
86  fclose(fp);
    br_studenata = i;
88
90  /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
    studenata sa istim brojem bodova sortiranje vrši po prezimenu */
    qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
92
94  printf("Studenti sortirani po broju poena opadajuće, ");
    printf("pa po prezimenu rastuće:\n");
    for (i = 0; i < br_studenata; i++)
96        printf("%s %s %d\n", kolokvijum[i].ime,
            kolokvijum[i].prezime, kolokvijum[i].bodovi);
98
100  /* Pretraživanje studenata po broju bodova se vrši binarnom
    pretragom jer je niz sortiran po broju bodova. */
    printf("Unesite broj bodova: ");
102    scanf("%d", &bodovi);
104
    nadjen =
        bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
106            &poredi2);
108
    if (nadjen != NULL)
        printf
110            ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
            nadjen->ime, nadjen->prezime, nadjen->bodovi);
112    else
        printf("Nema studenta sa unetim brojem bodova\n");
114
116  /* Pretraga po prezimenu se mora vršiti linearno jer je niz
    sortiran po bodovima. */
    printf("Unesite prezime: ");
118    scanf("%s", prezime);
120
    nadjen =
        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
122            &poredi3);
124
    if (nadjen != NULL)
        printf
126            ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
            nadjen->ime, nadjen->prezime, nadjen->bodovi);
128    else
        printf("Nema studenta sa unetim prezimenom\n");
130
    return 0;
132 }
```

## Rešenje 3.35

```

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

4 #define MAX 128

6 /* Funkcija poredi dva karaktera */
8 int uporedi_char(const void *pa, const void *pb)
{
10     return *(char *) pa - *(char *) pb;
}

12 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14 int anagrami(char s[], char t[])
{
16     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
    if (strlen(s) != strlen(t))
18         return 0;

20     /* Sortiranje niski */
    qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);

24     /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
        suprotnom, nisu */
26     return !strcmp(s, t);
}

28
int main()
30 {
    char s[MAX], t[MAX];

32     /* Unos niski */
    printf("Unesite prvu nisku: ");
34     scanf("%s", s);
    printf("Unesite drugu nisku: ");
36     scanf("%s", t);

38     /* Ispituje se da li su niske anagrami */
40     if (anagrami(s, t))
        printf("jesu\n");
42     else
        printf("nisu\n");

44     return 0;
46 }

```

### Rešenje 3.36

```

1 #include <stdio.h>

```

```

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

5  #define MAX 10
   #define MAX_DUZINA 32

7  /* Funkcija porenjenja */
9  int uporedi_niske(const void *pa, const void *pb)
   {
11     return strcmp((char *) pa, (char *) pb);
   }

13
15 int main()
   {
17     int i, n;
       char S[MAX][MAX_DUZINA];

19     /* Unos broja niski */
       printf("Unesite broj niski:");
21     scanf("%d", &n);

23     /* Unos niza niski */
       printf("Unesite niske:\n");
25     for (i = 0; i < n; i++)
         scanf("%s", S[i]);

27     /* Sortiranje niza niski */
29     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

31     /******
       Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
       sortiranih niski. Koriscen je samo u fazi testiranja programa.
       *****/

35     printf("Sortirane niske su:\n");
       for(i = 0; i < n; i++)
37         printf("%s ", S[i]);
       /******

39     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
       niza biti jedna do druge */
41     for (i = 0; i < n - 1; i++)
43         if (strcmp(S[i], S[i + 1]) == 0) {
           printf("ima\n");
           return 0;
         }

47     printf("nema\n");
49     return 0;
   }

```

Rešenje 3.37

```
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    char prezime[MAX];
12    int poeni;
13 } Student;
14
15 /* Funkcija poredi studente prema broju poena, rastuce */
16 int uporedi_poeni(const void *a, const void *b)
17 {
18     Student s = *(Student *) a;
19     Student t = *(Student *) b;
20     return s.poeni - t.poeni;
21 }
22
23 /* Funkcija poredi studente prvo prema godini, zatim prema smeru i na
24    kraju prema indeksu */
25 int uporedi_nalog(const void *a, const void *b)
26 {
27     Student s = *(Student *) a;
28     Student t = *(Student *) b;
29     /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
30        broj indeksa */
31     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
32     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
33     char smer1 = s.nalog[1];
34     char smer2 = t.nalog[1];
35     int indeks1 =
36         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
37         s.nalog[6] - '0';
38     int indeks2 =
39         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
40         t.nalog[6] - '0';
41     if (godina1 != godina2)
42         return godina1 - godina2;
43     else if (smer1 != smer2)
44         return smer1 - smer2;
45     else
46         return indeks1 - indeks2;
47 }
48
49 int uporedi_bsearch(const void *a, const void *b)
50 {
51     /* Nalog studenta koji se trazi */
```

```

char *nalog = (char *) a;
53 /* Ključ pretrage */
Student s = *(Student *) b;
55
int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
57 int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
char smer1 = nalog[1];
59 char smer2 = s.nalog[1];
int indeks1 =
61     (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
    ;
int indeks2 =
63     (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
    s.nalog[6] - '0';
65 if (godina1 != godina2)
    return godina1 - godina2;
67 else if (smer1 != smer2)
    return smer1 - smer2;
69 else
    return indeks1 - indeks2;
71 }

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

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

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

97     /* Otvaranje datoteke za pisanje */
    out = fopen("izlaz.txt", "w");
99     if (out == NULL) {
        fprintf(stderr,

```



```

101         "Greska prilikom otvaranja datoteke izlaz.txt!\n");
        exit(EXIT_FAILURE);
103     }

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

111     br_studenata = i;

113     /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
    if (strcmp(argv[1], "-p") == 0)
115         qsort(niz_studenata, br_studenata, sizeof(Student),
            &uporedi_poeni);
117     /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
    else if (strcmp(argv[1], "-n") == 0)
119         qsort(niz_studenata, br_studenata, sizeof(Student),
            &uporedi_nalog);
121

123     /* Sortirani studenti se ispisuju u izlaznu datoteku */
    for (i = 0; i < br_studenata; i++)
125         fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
            niz_studenata[i].ime, niz_studenata[i].prezime,
127             niz_studenata[i].poeni);

129     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
        studenta... */
131     if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
        strcpy(nalog_trazeni, argv[2]);
133

        /* ... pronalazi se student sa tim nalogom... */
135        nadjen =
            (Student *) bsearch(nalog_trazeni, niz_studenata,
137                                br_studenata, sizeof(Student),
                                    &uporedi_bsearch);

139

        if (nadjen == NULL)
141            printf("Nije nadjen!\n");
        else
143            printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
                nadjen->prezime, nadjen->poeni);
145    }

147    /* Zatvaranje datoteka */
    fclose(in);
149    fclose(out);

151    return 0;
}

```

## 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. Čvor 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 mu polja 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 `int dodaj_iza(Cvor * tekuci, int broj)` koja iza čvora `tekuci` dodaje novi čvor sa vrednošću `broj`.
- (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 zagradaama `[, ]` i međusobno razdvojene zapetama.
- (j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.
- (k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.
- (l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.
- (n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. 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*

```

INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj 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]
17
Lista: [17, 3, 5, 14, 3, 2]

Unosite broj koji se trazi: 5
Trazeni broj 5 je u listi!

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [35, 14, 23]

Unosite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!

```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 14, 3]
3
Lista: [2, 3, 14, 3, 3]
17
Lista: [2, 3, 14, 3, 3, 17]
3
Lista: [2, 3, 14, 3, 3, 17, 3]

Unosite broj koji se brise: 3
Lista nakon brisanja: [2, 14, 17]

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [23, 14]
35
Lista: [23, 14, 35]

Unosite broj koji se brise: 3
Lista nakon brisanja: [23, 14, 35]

```

- (3) U glavnom programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se 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 čvora liste koristiti činjenicu da je lista uređena.*

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 3, 14]
3
Lista: [2, 3, 3, 3, 14]
5
Lista: [2, 3, 3, 3, 5, 14]

Unesite broj koji se trazi: 14
Trazeni broj 14 je u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 5, 14]

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [14, 23, 35]

Unesite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!

Unesite broj koji se brise: 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 test primere iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etiketke smeštati u listu, a za formiranje liste koristiti strukturu:

```

typedef struct _Element
{
    unsigned broj_pojavljivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;

```

*Test 1*

```
POZIV: ./a.out datoteka.html

DATOTEKA.HTML
<html>
<head><title>Primer</title></head>
<body>
  <h1>Naslov</h1>
  Danas je lep i suncan dan. <br>
  A sutra ce biti jos lepsi.
  <a link='http://www.google.com'> Link 1</a>
  <a link='http://www.math.rs'> Link 2</a>
</body>
</html>

IZLAZ:
a - 4
br - 1
h1 - 2
body - 2
title - 2
head - 2
html - 2
```

*Test 2*

```
POZIV: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja
datoteke datoteka.html.
```

[Rešenje 4.3]

**Zadatak 4.4** 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 podataka o studentima, 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). Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *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 SA PROGRAMOM:
3/2014 da: Ana Sokic
235/2008 ne
41/2009 da: Marija Zaric

```

*Primer 2*

```

Poziv: ./a.out studenti.txt

DATOTEKA STUDENTI.TXT JE PRAZNA

INTERAKCIJA SA PROGRAMOM:
3/2014 ne
235/2008 ne
41/2009 ne

```

[Rešenje 4.4]

**Zadatak 4.5** 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*

```

BROJEVI.TXT
43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
12 15 16

```

*Test 2*

```

DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAZ:
REZULTAT.TXT
Greska prilikom otvaranja
datoteke brojevi.txt.

```

*Test 3*

```

DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ:
REZULTAT.TXT
Rezultat.txt ce biti prazna.

```

**Zadatak 4.6** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime 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

DATOTEKA DAT2.TXT NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja datoteke
dat2.txt.

```

*Test 3*

```

Poziv: ./a.out dat1.txt dat2.txt

DATOTEKA DAT1.TXT JE PRAZNA

DAT2.TXT
  5 6 11 12 14 16

IZLAZ:
[5, 6, 11, 12, 14, 16]

```

*Test 4*

```

Poziv: ./a.out dat1.txt

IZLAZ:
Program se poziva sa:
./a.out dat1.txt dat2.txt!

```

[Rešenje 4.6]

**Zadatak 4.7** Date su dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od ovih listi formira novu listu L koja sadrži naizmenično raspoređene čvorove listi 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 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 - 6 za zadatak 4.6.*

*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 4.8** 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.

<p><i>Test 1</i></p> <pre> IZRAZ.TXT {[23 + 5344] * (24 - 234)} - 23  IZLAZ: Zagrade su ispravno uparene. </pre>	<p><i>Test 2</i></p> <pre> IZRAZ.TXT {[23 + 5] * (9 * 2)} - {23}  IZLAZ: Zagrade su ispravno uparene. </pre>
<p><i>Test 3</i></p> <pre> IZRAZ.TXT {[2 + 54] / (24 * 87)} + (234 + 23)  IZLAZ: Zagrade nisu ispravno uparene. </pre>	<p><i>Test 4</i></p> <pre> IZRAZ.TXT {(2 - 14) / (23 + 11)} * (2 + 13)  IZLAZ: Zagrade nisu ispravno uparene. </pre>
<p><i>Test 5</i></p> <pre> DATOTEKA IZRAZ.TXT JE PRAZNA  IZLAZ: Zagrade su ispravno uparene. </pre>	<p><i>Test 6</i></p> <pre> DATOTEKA IZRAZ.TXT NE POSTOJI.  IZLAZ: Greska prilikom otvaranja datoteke izraz.txt! </pre>

[Rešenje 4.8]

**Zadatak 4.9** 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 čvorovi sadrže HTML etikete.*

<p><i>Test 1</i></p> <pre> Poziv: ./a.out datoteka.html  DATOTEKA.HTML &lt;html&gt; &lt;head&gt;   &lt;title&gt;Primer&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;/body&gt;  IZLAZ: Etikete nisu pravilno uparene (etiketa &lt;html&gt; nije zatvorena) </pre>	<p><i>Test 2</i></p> <pre> Poziv: ./a.out datoteka.html  DATOTEKA.HTML &lt;head&gt;   &lt;title&gt;Primer&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;/body&gt; &lt;/html&gt;  IZLAZ: Etikete nisu pravilno uparene (nadjena je etiketa &lt;/html&gt; koja nije otvorena) </pre>
--	---

## 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>
    Sutra ce biti jos lepsi.
    <a link='http://www.math.rs'>Link</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 je etiketa </html>, a poslednja
  otvorena je <body>)

```

## Test 5

```

Poziv: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ:
  Greska prilikom otvaranja
  datoteke datoteka.html.

```

## Test 6

```

Poziv: ./a.out datoteka.html

DATOTEKA.HTML JE PRAZNA

IZLAZ:
  Etikete su pravilno uparene!

```

[Rešenje 4.9]

**Zadatak 4.10** Napisati program koji pomaže službeniku u radu na šalteru. Službenik najpre evidentira sve 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. Program mu 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*, 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 *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nezvano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Sluzbenik evidentira korisnicke zahteve:
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:

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 2345678901234
i zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

IZVESTAJ.TXT
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 3456789012345 Zahtev: Reklamacija
JMBG: 1234567890123 Zahtev: Otvaranje racuna

```

[Rešenje 4.10]

**Zadatak 4.11** 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 **`

`adresa_kraja`, `Cvor * tekuci`) koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave` i poslednji čvor liste na adresi `adresa_kraja`.

- (b) Napisati funkciju `void ispisi_listu_unazad(Cvor * kraj)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. Zbog efikasnog izvršavanja operacija dodavanja na kraj liste i ispisivanja liste unazad treba, pored pokazivača na glavu liste, čuvati i pokazivač na poslednji čvor liste. NAPOMENA: *Funkcije testirati koristeći test primere iz zadatka 4.1*

[Rešenje 4.11]

**Zadatak 4.12** Grupa od  $n$  plesača na kostimima ima 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 sada počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>    ULAZ:    5 3       IZLAZ:    3 1 5 2 4 </pre>	<pre>    ULAZ:    8 4       IZLAZ:    4 8 5 2 1 3 7 6 </pre>	<pre>    ULAZ:    3 8       IZLAZ:    n mora biti uvek vece    od k, a 3 &lt; 8! </pre>

**Zadatak 4.13** Grupa od  $n$  plesača na kostimima ima 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 sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojivati 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.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>    ULAZ:    5 3       IZLAZ:    3 5 4 2 1 </pre>	<pre>    ULAZ:    8 4       IZLAZ:    4 8 5 7 6 3 2 1 </pre>	<pre>    ULAZ:    5 8       IZLAZ:    n mora biti uvek vece    od k, a 5 &lt; 8! </pre>

## 4.2 Stabla

**Zadatak 4.14** Napisati biblioteku za rad sa binarnim pretraživačkim stablima.

- Definisati strukturu `Cvor` kojom se opisuje čvor stabla, a koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.
- Napisati funkciju `Cvor *napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.
- Napisati funkciju `int dodaj_u_stablo(Cvor ** adresa_korena, int broj)` koja u stablo na koje pokazuje argument `adresa_korena` dodaje ceo broj `broj`. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.
- 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.
- Napisati funkciju `Cvor *pronadji_najmanji(Cvor * koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- Napisati funkciju `Cvor *pronadji_najveci(Cvor * koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- Napisati funkciju `void obrisi_element(Cvor ** adresa_korena, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `adresa_korena`.

- (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 ** adresa_korena)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `adresa_korena`.

Korišćenjem kreirane biblioteke, 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.

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
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
Rezultuje stablo: 1 2 9 18 32
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
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
Rezultuje stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

**Zadatak 4.15** Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog 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 1*

```

Poziv: ./a.out test.txt

TEST.TXT
  Sunce utorak raCunar SUNCE programiranje
  jabuka PROGramiranje sunCE JABUKa

IZLAZ:
  jabuka: 2
  programiranje: 2
  racunar: 1
  sunce: 3
  utorak: 1

  Najcesca rec: sunce (pojavljuje se 3 puta)

```

*Test 2*

```

Poziv: ./a.out suma.txt

SUMA.TXT
  lipa zova hrast ZOVA breza LIPA

IZLAZ:
  breza: 1
  hrast: 1
  lipa: 2
  zova: 2

  Najcesca rec: lipa
  (pojavljuje se 2 puta)

```

*Test 3*

```

Poziv: ./a.out ulaz.txt

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
  Greska: Neuspesno otvaranje datoteke ulaz.txt.

```

*Test 4*

```

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.

*Primer 1*

```

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

INTERAKCIJA SA PROGRAMOM:
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

```

*Primer 2*

```

DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik1.txt
Greska: Neuspesno otvaranje datoteke
imenik1.txt.

```

[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 (niska od najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

*Test 1*

```

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

```

*Test 2*

```

DATOTEKA PRIJEMNI.TXT NE POSTOJI

IZLAZ:
Greska: Neuspesno otvaranje datoteke
prijemni.txt.

```



[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. Svaka linija datoteke je formata **Ime Prezime DD.MM.** i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadanog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovak postupak treba ponavljati dokle god korisnik ne unese komandu za kraj unosa. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu DD.MM..Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

*Primer 1*

```
Poziv: ./a.out rodjendani.txt

RODJENDANI.TXT
Marko Markovic 12.12.
Milan Jevremovic 04.06.
Maja Agic 23.04.
Nadica Zec 01.01.
Jovana Milic 05.05.

INTERAKCIJA SA PROGRAMOM:
Unesite datum: 23.04.
Slavljenik: Maja Agic
Unesite datum: 01.01.
Slavljenik: Nadica Zec
Unesite datum: 01.05.
Slavljenik: Jovana Milic 05.05.
Unesite datum: 20.12.
Slavljenik: Nadica Zec 01.01.
Unesite datum:
```

*Primer 2*

```
Poziv: ./a.out rodjendani.txt

DATOTEKA RODJENDANI.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje datoteke
rodjendani.txt.
```

[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: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Primer 1*

```

|| INTERAKCIJA SA PROGRAMOM:
|| 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
|| Stabla jesu identicna.

```

*Primer 2*

```

|| INTERAKCIJA SA PROGRAMOM:
|| 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
|| Stabla nisu identicna.

```

[Rešenje 4.19]

**\* Zadatak 4.20** Napisati program za rad sa skupovima u kojem se skupovi predstavljaju pomoću binarnih pretraživačkih stabala. Program za dva skupa čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku skupova. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Primer 1*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Prvi skup: 1 7 8 9 2 2
|| Drugi skup: 3 9 6 11 1
|| Unija: 1 1 2 2 3 6 7 8 9 9 11
|| Presek: 1 9
|| Razlika: 2 2 7 8

```

*Primer 2*

```

|| INTERAKCIJA SA PROGRAMOM:
|| Prvi skup: 11 2 7 5
|| Drugi skup: 4 3 3 7
|| 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 standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Primer 1*

```

|| INTERAKCIJA SA PROGRAMOM:
|| n: 7
|| a: 1 11 8 6 37 25 30
|| 1 6 8 11 25 30 37

```

*Primer 2*

```

|| INTERAKCIJA SA PROGRAMOM:
|| n: 55
|| 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. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

<i>Test 1</i>	<i>Test 2</i>
<pre> 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 element na 2. nivou: 30   Zbir elemenata na 2. nivou: 45   Zbir elemenata manjih ili jednakih od 15:   78           </pre>	<pre> Poziv: ./a.out 3 31 Ulaz:   24 53 61 9 7 55 20 16 Izlaz:   Broj cvorova: 8   Broj listova: 3   Pozitivni listovi: 7 16 55   Zbir cvorova: 245   Najveci element: 61   Dubina stabla: 4   Broj cvorova na 3. nivou: 2   Elementi na 3. nivou: 16 55   Maksimalni element na 3. nivou: 55   Zbir elemenata na 3. nivou: 71   Zbir elemenata manjih ili jednakih od 31:   76           </pre>

[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. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> 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           </pre>	<pre> ULAZ:  6 11 8 3 -2  IZLAZ: 0.nivo: 6 1.nivo: 3 11 2.nivo: -2 8           </pre>	<pre> ULAZ:  24 53 61 9 7 55 20 16  IZLAZ: 0.nivo: 24 1.nivo: 9 53 2.nivo: 7 20 61 3.nivo: 16 55           </pre>

[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.

<i>Primer 1</i>	<i>Primer 2</i>
<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 14 30 1 0 Stabla su slicna kao u ogledalu.           </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 20 15 0 Stabla nisu slicna kao u ogledalu.           </pre>

**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. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

## Test 1

```

|| ULAZ:
|| 10 5 15 2 11 16 1 13
||
|| IZLAZ:
|| 7

```

## Test 2

```

|| ULAZ:
|| 16 30 40 24 10 18 45 22
||
|| IZLAZ:
|| 6

```

[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *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 hip. Napisati zatim i glavni program koji kreira stablo zadato slikom 4.1, poziva funkciju `heap` i ispisuje rezultat na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

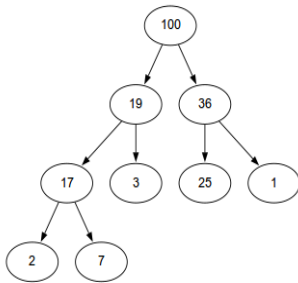
## Test 1

```

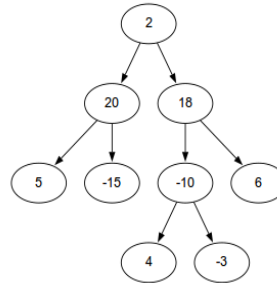
|| IZLAZ:
|| Zadato stablo je hip!

```

[Rešenje 4.26]



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

#### Test 1

```

IZLAZ:
Vrednost u cvoru sa maksimalnim desnim zbirom: 18
Vrednost u cvoru sa minimalnim levim zbirom: 18
2 18 -10 4
2 20 -15

```

## 4.3 Rešenja

### Rešenje 4.1

Datoteka 4.1: *lista.h*

```

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;
   struct cvor *sledeci;
8  } Cvor;

10 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12 dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
14 Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);

20 /* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo

```

```

22     greske pri alokaciji memorije, inace vraca 0. */
23 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
24
25 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
26    NULL ukoliko je lista prazna. */
27 Cvor *pronadji_poslednji(Cvor * glava);
28
29 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
30    pri alokaciji memorije, inace vraca 0. */
31 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
32
33 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
34    nov cvor sa vrednoscu broj. */
35 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
36
37 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
38    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
39 int dodaj_iza(Cvor * tekuci, int broj);
40
41 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
42    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
43    inace vraca 0. */
44 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
45
46 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
47    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
48    NULL u slučaju da takav cvor ne postoji u listi. */
49 Cvor *pretrazi_listu(Cvor * glava, int broj);
50
51 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
52    U pretrazi oslanja se na cinjenicu da je lista koja se pretražuje
53    neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
54    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
55    */
56 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
57
58 /* Funkcija briše iz liste sve cvorove koji sadrže dati broj. Azurira
59    pokazivac na glavu liste, koji može biti promenjen u slučaju da se
60    obrise stara glava. */
61 void obrisi_cvor(Cvor ** adresa_glave, int broj);
62
63 /* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
64    oslanjajući se na cinjenicu da je prosledjena lista sortirana
65    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
66    promenjen ukoliko se obrise stara glava liste. */
67 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
68
69 /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
70    liste, razdvojene zarezima i uokvirene zagradama. */
71 void ispisi_listu(Cvor * glava);
72
73 #endif

```

Datoteka 4.2: *lista.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 Cvor *napravi_cvor(int broj)
6 {
7     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8        alokacije */
9     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10    if (novi == NULL)
11        return NULL;
12
13    /* Inicijalizacija polja strukture */
14    novi->vrednost = broj;
15    novi->sledeci = NULL;
16
17    /* Vraca se adresa novog cvora */
18    return novi;
19 }
20
21 void oslobodi_listu(Cvor ** adresa_glave)
22 {
23     Cvor *pomocni = NULL;
24
25     /* Ako lista nije prazna, onda treba osloboditi memoriju */
26     while (*adresa_glave != NULL) {
27         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
28            osloboditi cvor koji predstavlja glavu liste */
29         pomocni = (*adresa_glave)->sledeci;
30         free(*adresa_glave);
31
32         /* Sledeci cvor je nova glava liste */
33         *adresa_glave = pomocni;
34     }
35 }
36
37 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
38 {
39     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
40     Cvor *novi = napravi_cvor(broj);
41     if (novi == NULL)
42         return 1;
43
44     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
45     novi->sledeci = *adresa_glave;
46     *adresa_glave = novi;
47
48     /* Vraca se indikator uspesnog dodavanja */
49     return 0;
```



```

}
51 Cvor *pronadji_poslednji(Cvor * glava)
53 {
    /* U praznoj listi nema cvorova pa se vraća NULL */
55     if (glava == NULL)
        return NULL;
57
    /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
59     glava se pomera na sledeći cvor. Nakon izlaska iz petlje, glava
61     će pokazivati na cvor liste koji nema sledbenika, tj. na
        poslednji cvor liste pa se vraća vrednost pokazivaca glava.

63     Pokazivac glava je argument funkcije i njegove promene neće se
        odraziti na vrednost pokazivaca glava u pozivajućoj funkciji. */
65     while (glava->sledeci != NULL)
        glava = glava->sledeci;
67
69     return glava;
71
72 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
73 {
    /* Kreira se novi cvor i proverava se uspešnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
75     if (novi == NULL)
        return 1;
77
    /* Ako je lista prazna */
79     if (*adresa_glave == NULL) {
        /* Glava nove liste je upravo novi cvor */
81         *adresa_glave = novi;
    } else {
83         /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
            se dodaje na kraj liste kao sledbenik poslednjeg */
85         Cvor *poslednji = pronadji_poslednji(*adresa_glave);
            poslednji->sledeci = novi;
87     }

89     /* Vraća se indikator uspešnog dodavanja */
    return 0;
91 }

93 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
94 {
    /* U praznoj listi nema takvog mesta i vraća se NULL */
95     if (glava == NULL)
96         return NULL;
97
98     /* Pokazivac glava se pomera na sledeći cvor sve dok ne bude
99     pokazivao na cvor čiji sledeći ili ne postoji ili ima vrednost
101     veću ili jednaku vrednosti novog cvora.

```

```
103     Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
105     proveriti da li se doslo do poslednjeg cvora liste pre nego sto se
107     proveru vrednost u sledecem cvoru, jer u slucaju poslednjeg,
109     sledeci ne postoji, pa ni njegova vrednost. */
111     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
113         glava = glava->sledeci;

115     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
117     poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
119     vrednost vecu od broj. */
121     return glava;
123 }

125 int dodaj_iza(Cvor * tekuci, int broj)
127 {
129     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
131     Cvor *novi = napravi_cvor(broj);
133     if (novi == NULL)
135         return 1;

137     /* Novi cvor se dodaje iza tekuceg cvora. */
139     novi->sledeci = tekuci->sledeci;
141     tekuci->sledeci = novi;

143     /* Vraca se indikator uspesnog dodavanja */
145     return 0;
147 }

149 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
151 {
153     /* Ako je lista prazna */
155     if (*adresa_glave == NULL) {
157         /* Glava nove liste je novi cvor */

159         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
161         Cvor *novi = napravi_cvor(broj);
163         if (novi == NULL)
165             return 1;

167         *adresa_glave = novi;

169         /* Vraca se indikator uspesnog dodavanja */
171         return 0;
173     }

175     /* Inace... */

177     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
179     treba dodati na pocetak liste */
181     if ((*adresa_glave)->vrednost >= broj) {
183         return dodaj_na_pocetak_liste(adresa_glave, broj);
185     }
```

```

155     }
156     /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
157        tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
158     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
159     return dodaj_iza(pomocni, broj);
160 }
161
162 Cvor *pretrazi_listu(Cvor * glava, int broj)
163 {
164     /* Obilaze se cvorovi liste */
165     for (; glava != NULL; glava = glava->sledeci)
166         /* Ako je vrednost tekućeg cvora jednaka zadatom broju, pretraga
167            se obustavlja */
168         if (glava->vrednost == broj)
169             return glava;
170
171     /* Nema traženog broja u listi i vraća se NULL */
172     return NULL;
173 }
174
175 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
176 {
177     /* Obilaze se cvorovi liste */
178     /* U uslovu ostanka u petlji, bitan je redosled proveru u
179        konjunktiji */
180     for (; glava != NULL && glava->vrednost <= broj;
181           glava = glava->sledeci)
182         /* Ako je vrednost tekućeg cvora jednaka zadatom broju, pretraga
183            se obustavlja */
184         if (glava->vrednost == broj)
185             return glava;
186
187     /* Nema traženog broja u listi i vraća se NULL */
188     return NULL;
189 }
190
191 void obrisi_cvor(Cvor ** adresa_glave, int broj)
192 {
193     Cvor *tekuci = NULL;
194     Cvor *pomocni = NULL;
195
196     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
197        broju i azurira se pokazivac na glavu liste */
198     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
199     {
200         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
201            adresi adresa_glave */
202         pomocni = (*adresa_glave)->sledeci;
203         free(*adresa_glave);
204         *adresa_glave = pomocni;
205     }

```

```

205  /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
207  funkcije */
209  if (*adresa_glave == NULL)
211      return;

211  /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
213  pokazuje na cvor cija je vrednost razlicita od trazenog broja.
215  Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
217  sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
219  ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
221  Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
223  */
225  tekuci = *adresa_glave;
227  while (tekuci->sledeci != NULL)
229  {
231      if (tekuci->sledeci->vrednost == broj) {
233          /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
235          prvo cuva u promenljivoj pomocni. */
237          pomocni = tekuci->sledeci;
239          /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
241          njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
243          sledeci od cvora koji se brise. */
245          tekuci->sledeci = pomocni->sledeci;
247          /* Sada treba osloboditi cvor sa vrednoscu broj. */
249          free(pomocni);
251      } else {
253          /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
255          pomera na sledeci. */
257          tekuci = tekuci->sledeci;
259      }
261      return;
263  }

265  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
267  {
269      Cvor *tekuci = NULL;
271      Cvor *pomocni = NULL;

273      /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
275      broju i azurira se pokazivac na glavu liste. */
277      while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
279      {
281          /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
283          adresi adresa_glave. */
285          pomocni = (*adresa_glave)->sledeci;
287          free(*adresa_glave);
289          *adresa_glave = pomocni;
291      }

293      /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
295      prekida. Isto se radi i ukoliko glava liste sadrzi vrednost
297      koja je veca od broja, jer kako je lista sortirana rastuce nema

```

```

255     potrebe broj traziti u repu liste. */
256     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
257         return;

259     /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
260        na cvor cija vrednost je manja od trazenog broja, kao i svim
261        cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
262        je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
263        ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
264        cija vrednost je veca od trazenog broja. */
265     tekuci = *adresa_glave;
266     while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
267            )
268     {
269         if (tekuci->sledeci->vrednost == broj) {
270             pomocni = tekuci->sledeci;
271             tekuci->sledeci = tekuci->sledeci->sledeci;
272             free(pomocni);
273         } else {
274             /* Ne treba brisati sledeceg od tekuceg jer je manji od
275                trazenog i tekuci se pomera na sledeci cvor. */
276             tekuci = tekuci->sledeci;
277         }
278     }
279     return;
280 }

281 void ispisi_listu(Cvor * glava)
282 {
283     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
284        jer se lista nece menjati */
285     putchar('[');
286     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
287        pocetka prema kraju liste. */
288     for (; glava != NULL; glava = glava->sledeci) {
289         printf("%d", glava->vrednost);
290         if (glava->sledeci != NULL)
291             printf(", ");
292     }
293     printf("]\n");
294 }

```

Datoteka 4.3: *main\_a.c*

```

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

4

/* 1) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku */
    Cvor *glava = NULL;

```

```

10  Cvor *trazeni = NULL;
    int broj;

12

14  /* Testiranje dodavanja novog broja na pocetak liste */
    printf("Unesite brojeve: (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
16      /* 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. */
18      if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
20          fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
                oslobodi_listu(&glava);
22          exit(EXIT_FAILURE);
            }
24      printf("\tLista: ");
            ispisi_listu(glava);
26  }

28  /* Testiranje funkcije za pretragu liste */
    printf("\nUnesite broj koji se trazi: ");
30    scanf("%d", &broj);

32    trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
34        printf("Broj %d se ne nalazi u listi!\n", broj);
    else
36        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

38    /* Oslobadja se memorija zauzeta listom */
    oslobodi_listu(&glava);

40    return 0;
42 }

```

Datoteka 4.4: *main\_b.c*

```

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

4

/* 2) Glavni program */
6  int main()
    {
8      /* Lista je prazna na pocetku */
        Cvor *glava = NULL;
        int broj;

12     /* Testira se funkcija za dodavanja novog broja na kraj liste */
        printf("Unesite brojeve: (za kraj CTRL+D)\n");
14     while (scanf("%d", &broj) > 0) {
            /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji

```

```

16     memorije za nov cvor. Memoriju alociranu za cvorove liste
      treba osloboditi pre napustanja programa. */
18     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
  /* Testira se funkcije kojom se brise cvor liste */
28  printf("\nUnesite broj koji se brise: ");
  scanf("%d", &broj);

30
  /* Brisu se cvorovi iz liste cije je polje vrednost jednako broju
   procitanom sa ulaza */
32  obrisi_cvor(&glava, broj);

34
  printf("Lista nakon brisanja: ");
36  ispisi_listu(glava);

38
  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);

40
  return 0;
42 }

```

Datoteka 4.5: *main\_c.c*

```

#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

4
/* 3) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku */
    Cvor *glava = NULL;
10    Cvor *trazeni = NULL;
    int broj;

12
    /* Testira se funkcija za dodavanje vrednosti u listu tako da bude
     uredjena neopadajuće */
14    printf("Unosite brojeve (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
16        /* 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. */
18        if (dodaj_sortirano(&glava, broj) == 1) {
20            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);

```

```

22     oslobodi_listu(&glava);
23     exit(EXIT_FAILURE);
24 }
25 printf("\tLista: ");
26 ispisi_listu(glava);
27 }
28
29 /* Testira se funkcija za pretragu liste */
30 printf("\nUnesite broj koji se trazi: ");
31 scanf("%d", &broj);
32
33 trazeni = pretrazi_listu(glava, broj);
34 if (trazeni == NULL)
35     printf("Broj %d se ne nalazi u listi!\n", broj);
36 else
37     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
38
39 /* Testira se funkcija kojom se brise cvor liste */
40 printf("\nUnesite broj koji se brise: ");
41 scanf("%d", &broj);
42
43 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
44    procitanom sa ulaza */
45 obrisi_cvor_sortirane_liste(&glava, broj);
46
47 printf("Lista nakon brisanja: ");
48 ispisi_listu(glava);
49
50 /* Oslobadja se memorija zauzeta listom */
51 oslobodi_listu(&glava);
52
53 return 0;
54 }

```

## Rešenje 4.2

Datoteka 4.6: *lista.h*

```

1  #ifndef _LISTA_H_
2  #define _LISTA_H_
3
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 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac

```



```
14     na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
15 Cvor *napravi_cvor(int broj);
16
17 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
18    ciji se pokazivac glava nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave);
20
21 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
22    bilo greske pri alokaciji memorije, inace vraca 0. */
23 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
24
25 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
26    pri alokaciji memorije, inace vraca 0. */
27 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
28
29 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
30    ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
31    memorije, inace vraca 0. */
32 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
33
34 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
35    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
36    NULL u slučaju da takav cvor ne postoji u listi. */
37 Cvor *pretrazi_listu(Cvor * glava, int broj);
38
39 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
40    U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
41    neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
42    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
43    */
44 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
45
46 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
47    pokazivac na glavu liste, koji može biti promenjen u slučaju da se
48    obrise stara glava liste. */
49 void obrisi_cvor(Cvor ** adresa_glave, int broj);
50
51 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
52    oslanjajući se na cinjenicu da je prosledjena lista sortirana
53    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
54    promenjen ukoliko se obrise stara glava liste. */
55 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
56
57 /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
58    zaptama. */
59 void ispisi_vrednosti(Cvor * glava);
60
61 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
62    liste, razdvojene zaptama i uokvirene zagradama. */
63 void ispisi_listu(Cvor * glava);
64
65 #endif
```

Datoteka 4.7: *lista.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 Cvor *napravi_cvor(int broj)
6 {
7     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8        alokacije */
9     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10    if (novi == NULL)
11        return NULL;
12
13    /* Inicijalizacija polja strukture */
14    novi->vrednost = broj;
15    novi->sledeci = NULL;
16
17    /* Vraca se adresa novog cvora */
18    return novi;
19 }
20
21 void oslobodi_listu(Cvor ** adresa_glave)
22 {
23     /* Ako je lista vec prazna */
24     if (*adresa_glave == NULL)
25         return;
26
27     /* Ako lista nije prazna, treba osloboditi memoriju. Pre
28        oslobadjanja memorije za glavu liste, treba osloboditi rep liste
29        */
30     oslobodi_listu(&(*adresa_glave)->sledeci);
31     /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
32        glava u pozivajucoj funkciji tako da odgovara praznoj listi */
33     free(*adresa_glave);
34     *adresa_glave = NULL;
35 }
36
37 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
38 {
39     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
40     Cvor *novi = napravi_cvor(broj);
41     if (novi == NULL)
42         return 1;
43
44     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
45     novi->sledeci = *adresa_glave;
46     *adresa_glave = novi;
47
48     /* Vraca se indikator uspesnog dodavanja */
49     return 0;
```

```
51 }
52
53 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
54 {
55     /* Ako je lista prazna */
56     if (*adresa_glave == NULL) {
57
58         /* Novi cvor postaje glava liste */
59         Cvor *novi = napravi_cvor(broj);
60         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */
61         if (novi == NULL)
62             return 1;
63
64         /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
65            azurira i pokazivacka promenljiva u pozivajucoj funkciji */
66         *adresa_glave = novi;
67
68         /* Vraca se indikator uspesnog dodavanja */
69         return 0;
70     }
71
72     /* Ako lista nije prazna, broj se dodaje u rep liste. */
73     /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
74        novi broj se dodaje u rep liste u rekurzivnom pozivu.
75        Informaciju o uspesnosti alokacije u rekurzivnom pozivu
76        funkcija prosledjuje visem rekurzivnom pozivu koji tu
77        informaciju vraca u rekurzivni poziv iznad, sve dok se ne vrati
78        u main. Tek je iz main funkcije moguće pristupiti pravom pocetku
79        liste i osloboditi je celu, ako ima potrebe. Ako je funkcija
80        vratila 0, onda nije bilo greske. */
81     return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
82 }
83
84 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
85 {
86     /* Ako je lista prazna */
87     if (*adresa_glave == NULL) {
88
89         /* Novi cvor postaje glava liste */
90         Cvor *novi = napravi_cvor(broj);
91
92         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */
93         if (novi == NULL)
94             return 1;
95
96         /* Azurira se glava liste */
97         *adresa_glave = novi;
98
99         /* Vraca se indikator uspesnog dodavanja */
100        return 0;
101    }
```

```

101     }
102
103     /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
104        glavi liste, onda se dodaje na pocetak liste */
105     if ((*adresa_glave)->vrednost >= broj)
106         return dodaj_na_pocetak_liste(adresa_glave, broj);
107
108     /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
109        bude sortirana lista. */
110     return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
111 }
112
113 Cvor *pretrazi_listu(Cvor * glava, int broj)
114 {
115     /* U praznoj listi nema vrednosti */
116     if (glava == NULL)
117         return NULL;
118
119     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
120     if (glava->vrednost == broj)
121         return glava;
122
123     /* Inace, pretraga se nastavlja u repu liste */
124     return pretrazi_listu(glava->sledeci, broj);
125 }
126
127 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
128 {
129     /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
130        vrednosti u glavi liste, jer je lista neopadajuce sortirana */
131     if (glava == NULL || glava->vrednost > broj)
132         return NULL;
133
134     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
135     if (glava->vrednost == broj)
136         return glava;
137
138     /* Inace, pretraga se nastavlja u repu liste */
139     return pretrazi_listu(glava->sledeci, broj);
140 }
141
142 void obrisi_cvor(Cvor ** adresa_glave, int broj)
143 {
144     /* U praznoj listi nema cvorova za brisanje. */
145     if (*adresa_glave == NULL)
146         return;
147
148     /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
149     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
150
151     /* Preostaje provera da li glavu liste treba obrisati */
152     if ((*adresa_glave)->vrednost == broj) {

```

```

153     /* Pomocni pokazuje na cvor koji treba da se obrise */
154     Cvor *pomocni = *adresa_glave;
155     /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
156        brise se cvor koji je bio glava liste. */
157     *adresa_glave = (*adresa_glave)->sledeci;
158     free(pomocni);
159 }
160 }
161
162 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
163 {
164     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
165        od broja, kako je lista sortirana rastuce nema potrebe broj
166        traziti u repu liste i zato se funkcija prekida */
167     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
168         return;
169
170     /* Brisu se cvorovi iz repa koji imaju vrednost broj */
171     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
172
173     /* Preostaje provera da li glavu liste treba obrisati */
174     if ((*adresa_glave)->vrednost == broj) {
175         /* Pomocni pokazuje na cvor koji treba da se obrise */
176         Cvor *pomocni = *adresa_glave;
177         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
178            brise se cvor koji je bio glava liste */
179         *adresa_glave = (*adresa_glave)->sledeci;
180         free(pomocni);
181     }
182 }
183
184 void ispisi_vrednosti(Cvor * glava)
185 {
186     /* Prazna lista */
187     if (glava == NULL)
188         return;
189
190     /* Ispisuje se vrednost u glavi liste */
191     printf("%d", glava->vrednost);
192
193     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
194        se poziva ista funkcija za ispis ostalih. */
195     if (glava->sledeci != NULL) {
196         printf(", ");
197         ispisi_vrednosti(glava->sledeci);
198     }
199 }
200
201 void ispisi_listu(Cvor * glava)
202 {
203     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,

```

```

205     jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
na glavu liste iz pozivajuće funkcije. Ona ispisuje samo
207     zgrade, a rekurzivno ispisivanje vrednosti u listi prepusta
rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
elemente razdvojene zapetom i razmakom. */
209     putchar('[');
ispisi_vrednosti(glava);
211     printf("]\n");
}

```

### Rešenje 4.3

```

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

5  #define MAX_DUZINA 20

7  /* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
   broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
9  typedef struct _Cvor {
   char etiketa[20];
11     unsigned broj_pojavljivanja;
   struct _Cvor *sledeci;
13 } Cvor;

15 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
   ili NULL ako alokacija nije uspesno izvršena */
17 Cvor *napravi_cvor(unsigned br, char *etiketa)
{
19     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
       alokacije */
21     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   if (novi == NULL)
23         return NULL;

25     /* Inicijalizacija polja strukture */
   novi->broj_pojavljivanja = br;
27     strcpy(novi->etiketa, etiketa);
   novi->sledeci = NULL;

29     /* Vraca se adresa novog cvora */
31     return novi;
}

33 /* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
35 void oslobodi_listu(Cvor ** adresa_glave)
{
37     Cvor *pomocni = NULL;

39     /* Sve dok lista ni bude prazna, brise se tekuca glava liste i

```

```

    azurira se vrednost glave liste */
41 while (*adresa_glave != NULL) {
    pomocni = (*adresa_glave)->sledeci;
43 free(*adresa_glave);
    *adresa_glave = pomocni;
45 }
    /* Nakon izlaska iz petlje pokazivac glava u main funkciji koji se
47 nalazi na adresi adresa_glave bice postavljen na NULL vrednost.
    */
}

49
/* Funkcija dodaje novi cvor na pocetak liste. Povratna vrednost je 1
51 ako je doslo do greske pri alokaciji memorije za nov cvor, odnosno
0 */
53 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
    char *etiketa)
55 {
    /* Kreira se novi cvor i proverava se uspesnost alokacije */
57 Cvor *novi = napravi_cvor(br, etiketa);
    if (novi == NULL)
59     return 1;

61 /* Dodaje se novi cvor na pocetak liste */
    novi->sledeci = *adresa_glave;
63 *adresa_glave = novi;

65 /* Vraca se indikator uspesnog dodavanja */
    return 0;
67 }

69 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
    NULL ako takav cvor ne postoji u listi */
71 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
    {
73     Cvor *tekuci;

75     /* Obilazi se cvor po cvor liste */
    for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
77         /* Ako tekuci cvor sadrzi trazenu etiketu, vracamo njegovu
            vrednost */
79         if (strcmp(tekuci->etiketa, etiketa) == 0)
            return tekuci;

81     /* Cvor nije pronadjen */
83     return NULL;
    }

85
/* Funkcija ispisuje sadrzaj liste */
87 void ispisi_listu(Cvor * glava)
    {
89     /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
        broj njihovog pojavljivanja u HTML datoteci. */

```

```

91     for (; glava != NULL; glava = glava->sledeci)
        printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
93 }

95 int main(int argc, char **argv)
{
97     /* Provera se da li je program pozvan sa ispravnim brojem
        argumenata. */
99     if (argc != 2) {
        fprintf(stderr,
101             "Greska! Program se poziva sa: ./a.out datoteka.html!\n");
        ;
        exit(EXIT_FAILURE);
103     }

105     /* Priprema datoteke za citanje */
    FILE *in = NULL;
107     in = fopen(argv[1], "r");
    if (in == NULL) {
109         fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
111         exit(EXIT_FAILURE);
    }

113     char c;
115     int i = 0;
    char procitana[MAX_DUZINA];
117     Cvor *glava = NULL;
    Cvor *trazeni = NULL;

119     /* Cita se karakter po karakter datoteke sve dok se ne procita cela
        datoteka */
121     while ((c = fgetc(in)) != EOF) {
123
        /* Proverava se da li se pocinje sa citanjem nove etikete */
125         if (c == '<') {
            /* Proverava se da li se cita zatvarajuca etiketa */
127             if ((c = fgetc(in)) == '/') {
                i = 0;
129                 while ((c = fgetc(in)) != '>')
                    procitana[i++] = c;
131             }
            /* Cita se otvarajuca etiketa */
133             else {
                i = 0;
135                 procitana[i++] = c;
                while ((c = fgetc(in)) != ' ' && c != '>')
137                     procitana[i++] = c;
            }
139             procitana[i] = '\0';

141             /* Trazi se procitana etiketa medju postojećim cvorovima liste.

```



```

    Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
sa
143     brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
    etikete */
145     trazeni = pretrazi_listu(glava, procitana);
    if (trazeni == NULL) {
147         if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
            fprintf(stderr, "Neuspela alokacija za nov cvor\n");
149             oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
151         }
    } else
153     trazeni->broj_pojavljivanja++;
}
155 }

157 /* Zatvaranje datoteke */
fclose(in);

159 /* Ispisuje se sadrzaj cvorova liste */
ispisi_listu(glava);

161 /* Oslobadja se memorija zauzeta listom */
oslobodi_listu(&glava);

163
165     return 0;
167 }

```

#### Rešenje 4.4

```

#include <stdio.h>
2  #include <stdlib.h>
  #include <string.h>
4
#define MAX_INDEKS 11
6  #define MAX_IME_PREZIME 21

8  /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
    studentu */
10 typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
12     char ime[MAX_IME_PREZIME];
    char prezime[MAX_IME_PREZIME];
14     struct _Cvor *sledeci;
} Cvor;
16

18 /* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
    novi cvor ili NULL ukoliko je doslo do greske */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
    /* Alocira se memorija za novi cvor liste i proverava se uspesnost

```

```

22     alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
        return NULL;

26
    /* Inicijalizacija polja strukture */
28     strcpy(novi->broj_indeksa, broj_indeksa);
    strcpy(novi->ime, ime);
30     strcpy(novi->prezime, prezime);
    novi->sledeci = NULL;

32
    /* Vraca se adresa novog cvora */
34     return novi;
}

36
/* Funkcija oslobadja memoriju zauzetu cvorovima liste */
38 void oslobodi_listu(Cvor ** adresa_glave)
{
40     /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
    if (*adresa_glave == NULL)
42         return;

44     /* Rekurzivnim pozivom se oslobadja rep liste */
    oslobodi_listu(&(*adresa_glave)->sledeci);

46
    /* Potom se oslobadja i glava liste */
48     free(*adresa_glave);

50     /* Proglasava se lista praznom */
    *adresa_glave = NULL;
52 }

54
/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
   do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
56 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
                             char *ime, char *prezime)
58 {
    /* Kreira se novi cvor i proverava se uspesnost alokacije */
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
    if (novi == NULL)
62         return 1;

64     /* Dodaje se novi cvor na pocetak liste */
    novi->sledeci = *adresa_glave;
66     *adresa_glave = novi;

68     /* Vraca se indikator uspesnog dodavanja */
    return 0;
70 }

72
/* Funkcija ispisuje sadrzaj cvorova liste. */
void ispisi_listu(Cvor * glava)

```

```

74 {
75     /* Pocevsi od glave liste */
76     for (; glava != NULL; glava = glava->sledeci)
77         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
78             glava->prezime);
79 }
80
81 /* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj indeksa.
82    U suprotnom funkcija vraca NULL */
83 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
84 {
85     /* Ako je lista prazna, ne postoji trazeni cvor */
86     if (glava == NULL)
87         return NULL;
88
89     /* Poredi se trazeni broj indeksa sa brojem indeksa u glavi liste
90        */
91     if (!strcmp(glava->broj_indeksa, broj_indeksa))
92         return glava;
93
94     /* Ukoliko u glavi liste nije trazeni indeks, pretraga se nastavlja
95        u repu liste */
96     return pretrazi_listu(glava->sledeci, broj_indeksa);
97 }
98
99 int main(int argc, char **argv)
100 {
101     /* Argumenti komandne linije su neophodni jer se iz komandne linije
102        dobija ime datoteke sa informacijama o studentima */
103     if (argc != 2) {
104         fprintf(stderr,
105             "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
106         exit(EXIT_FAILURE);
107     }
108
109     /* Priprema datoteke za citanje */
110     FILE *in = NULL;
111     in = fopen(argv[1], "r");
112     if (in == NULL) {
113         fprintf(stderr,
114             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
115         exit(EXIT_FAILURE);
116     }
117
118     /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
119        listu */
120     char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
121     char broj_indeksa[MAX_INDEKS];
122     Cvor *glava = NULL;
123     Cvor *trazeni = NULL;
124
125     /* Ucitavanje vrednosti u listu */

```

```

126 while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
    if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
128         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_listu(&glava);
        exit(EXIT_FAILURE);
130     }

132 /* Datoteka vise nije potrebna i zatvara se. */
fclose(in);

134 /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
136 while (scanf("%s", broj_indeksa) != EOF) {
    trazeni = pretrazi_listu(glava, broj_indeksa);
138     if (trazeni == NULL)
        printf("ne\n");
140     else
        printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
142 }

144 /* Oslobadja se memorija zauzeta za cvorove liste. */
oslobodi_listu(&glava);

146 return 0;
148 }

```

### Rešenje 4.6

```

1 #include <stdio.h>
#include <stdlib.h>
3 #include "lista.h"

5 /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
   na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7 pokazivaca postojećih cvorova listi */
Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9 {
    /* Pokazivaci na pocetne cvorove listi koje se prevezuju */
11    Cvor *lista1 = *adresa_glave_1;
    Cvor *lista2 = *adresa_glave_2;

13    /* Pokazivac na pocetni cvor rezultujuće liste */
    Cvor *rezultujuca = NULL;
    Cvor *tekuci = NULL;

15    /* Ako su obe liste prazne i rezultat je prazna lista */
    if (lista1 == NULL && lista2 == NULL)
17        return NULL;

    /* Ako je prva lista prazna, rezultat je druga lista */
19    if (lista1 == NULL)
        return lista2;
21
23

```

```
25  /* Ako je druga lista prazna, rezultat je prva lista */
27  if (lista2 == NULL)
    return lista1;
29
31  /* Odredjuje se prvi cvor rezultujuce liste - to je ili prvi cvor
    liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
    koji sadrzi manju vrednost */
33  if (lista1->vrednost < lista2->vrednost) {
    rezultujuca = lista1;
35    lista1 = lista1->sledeci;
  } else {
37    rezultujuca = lista2;
    lista2 = lista2->sledeci;
39  }
41  /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste, ne
    sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
    koji sadrzi adresu trenutnog cvora rezultujuce liste */
43  tekuci = rezultujuca;

45  /* U svakoj iteraciji petlje rezultujucoj listi se dodaje novi cvor
    tako da bude uredjena neopadajuce. Pokazivac na listu iz koje se
    uzima cvor se azurira tako da pokazuje na sledeci cvor */
47  while (lista1 != NULL && lista2 != NULL) {
49    if (lista1->vrednost < lista2->vrednost) {
    tekuci->sledeci = lista1;
51    lista1 = lista1->sledeci;
    } else {
53    tekuci->sledeci = lista2;
    lista2 = lista2->sledeci;
55    }
    tekuci = tekuci->sledeci;
57  }

59  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
    rezultujucu listu treba nadovezati ostatak druge liste */
61  if (lista1 == NULL)
    tekuci->sledeci = lista2;
63  else
    /* U suprotnom treba nadovezati ostatak prve liste */
65    tekuci->sledeci = lista1;

67  /* Preko adresa glava polaznih listi vrednosti pokazivaca u
    pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
    prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
    se moze doci prateci pokazivace iz glave rezultujuce liste, tako
    da stare pokazivace treba postaviti na NULL. */
71  *adresa_glave_1 = NULL;
73  *adresa_glave_2 = NULL;

75  return rezultujuca;
}
```

```
77 int main(int argc, char **argv)
79 {
    /* Argumenti komandne linije su neophodni */
81     if (argc != 3) {
        fprintf(stderr,
83             "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
        exit(EXIT_FAILURE);
85     }

87     /* Otvaramo datoteke u kojima se nalaze elementi listi */
    FILE *in1 = NULL;
89     in1 = fopen(argv[1], "r");
    if (in1 == NULL) {
91         fprintf(stderr,
93             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
    }

95     FILE *in2 = NULL;
97     in2 = fopen(argv[2], "r");
    if (in2 == NULL) {
99         fprintf(stderr,
101             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
        exit(EXIT_FAILURE);
    }

103     /* Liste su na pocetku prazne */
105     int broj;
    Cvor *lista1 = NULL;
107     Cvor *lista2 = NULL;

109     /* Ucitavanje listi */
    while (fscanf(in1, "%d", &broj) != EOF)
111         dodaj_na_kraj_liste(&lista1, broj);

113     while (fscanf(in2, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&lista2, broj);

115     /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
117     fclose(in1);
    fclose(in2);

119     /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
121        objedinjavanjem listi */
    Cvor *rezultat = objedini(&lista1, &lista2);

123     /* Ispis rezultujuce liste. */
125     ispisi_listu(rezultat);

127     /* Lista rezultat dobijena je prevezivanjem cvorova polaznih listi.
        Njenim oslobadjanjem bice oslobodjena sva zauzeta memorija. */
```

```
129     oslobodi_listu(&rezultat);
131     return 0;
}
```

### Rešenje 4.8

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
   5     predstavlja zagradu koja se koristi i pokazivac na sledeci cvor
       liste */
7  typedef struct cvor {
   char zagrada;
   struct cvor *sledeci;
9  } Cvor;
11
   /* Funkcija koja oslobadja memoriju zauzetu stekom */
13 void oslobodi_stek(Cvor ** stek)
   {
15     Cvor *tekuci;
   Cvor *pomocni;
17
   /* Oslobadja se cvor po cvor steka */
19     tekuci = *stek;
   while (tekuci != NULL) {
21         pomocni = tekuci->sledeci;
         free(tekuci);
23         tekuci = pomocni;
   }
25
   /* Stek se proglašava praznim */
27     *stek = NULL;
   }
29
   int main()
31 {
   /* Stek je na pocetku prazan */
33     Cvor *stek = NULL;
   FILE *ulaz = NULL;
35     char c;
   Cvor *pomocni = NULL;
37
   /* Otvaranje datotoke za citanje izraza */
39     ulaz = fopen("izraz.txt", "r");
   if (ulaz == NULL) {
41         fprintf(stderr,
           "Greska prilikom otvaranja datoteke izraz.txt!\n");
43         exit(EXIT_FAILURE);
   }
}
```

```

45  /* Cita se karakter po karakter iz datoteke dok se ne dodje do
46  kraja */
47  while ((c = fgetc(ulaz)) != EOF) {
48      /* Ako je učitana otvorena zagrada, stavlja se na stek */
49      if (c == '(' || c == '{' || c == '[') {
50          /* Alocira se memorija za novi cvor liste i proverava se
51             uspesnost alokacije */
52          pomocni = (Cvor *) malloc(sizeof(Cvor));
53          if (pomocni == NULL) {
54              fprintf(stderr, "Greska prilikom alokacije memorije!\n");
55              /* Oslobadja se memorija zauzeta stekom */
56              oslobodi_stek(&stek);
57              /* I prekida se sa izvršavanjem programa */
58              exit(EXIT_FAILURE);
59          }
60
61          /* Inicijalizacija polja strukture */
62          pomocni->zagrada = c;
63
64          /* Promena vrha steka */
65          pomocni->sledeci = stek;
66          stek = pomocni;
67      }
68      /* Ako je učitana zatvorena zagrada, proverava se da li je stek
69         prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
70         otvorena zagrada */
71      else {
72          if (c == ')' || c == '}' || c == ']') {
73              if (stek != NULL && ((stek->zagrada == '(' && c == ')')
74                                  || (stek->zagrada == '{' && c == '}')
75                                  || (stek->zagrada == '[' && c == ']')))
76              {
77                  /* Sa vrha steka se uklanja otvorena zagrada */
78                  pomocni = stek->sledeci;
79                  free(stek);
80                  stek = pomocni;
81              } else {
82                  /* Inace, zakljucujemo da zagrade u izrazu nisu ispravno
83                     uparene */
84                  break;
85              }
86          }
87      }
88  }
89
90  /* Pročitana je cela datoteka i treba je zatvoriti */
91  fclose(ulaz);
92
93  /* Ako je stek prazan i pročitana je cela datoteka, zagrade su
94     ispravno uparene */
95  if (stek == NULL && c == EOF)

```



```

    printf("Zagrade su ispravno uparene.\n");
97  else {
    /* U suprotnom se zaključuje da zagrade nisu ispravno uparene */
99  printf("Zagrade nisu ispravno uparene.\n");
    /* Oslobadja se memorija koja je ostala zauzeta stekom */
101  oslobodi_stek(&stek);
    }
103
    return 0;
105 }

```

## Rešenje 4.9

Datoteka 4.8: *stek.h*

```

1  #ifndef _STEK_H_
2  #define _STEK_H_

4  #include <stdio.h>
   #include <stdlib.h>
6  #include <string.h>
   #include <ctype.h>
8
   #define MAX 100
10
   #define OTVORENA 1
12  #define ZATVORENA 2

14  #define VAN_ETIKETE 0
   #define PROCITANO_MANJE 1
16  #define U_ETIKETI 2

18  /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
    pokazivac na sledeci cvor */
20  typedef struct cvor {
    char etiketa[MAX];
22  struct cvor *sledeci;
    } Cvor;
24

   /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
26  adresu ili NULL ako alokacija nije bila uspesna */
   Cvor *napravi_cvor(char *etiketa);
28

   /* Funkcija oslobadja memoriju zauzetu stekom */
30  void oslobodi_stek(Cvor ** adresa_vrha);

32  /* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske pri
    alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0 */
34  int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);

```

```

36 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
37    pokazivac razlicit od NULL, tada u niz karaktera na koji on
38    pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
39    suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
40    samim tim nije bilo moguće skinuti vrednost sa steka) ili 1 u
41    suprotnom */
42 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);

44 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
45    steka. Ukoliko je stek prazan, vraca NULL */
46 char *vrh_steka(Cvor * vrh);

48 /* Funkcija prikazuje stek od vrha prema dnu */
49 void prikazi_stek(Cvor * vrh);

50
51 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
52    etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
53    Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
54    procita etiketa. Vraca OTVORENA, ako je procitana otvorena
55    etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
56 int uzmi_etiketu(FILE * f, char *etiketa);

57 #endif

```

Datoteka 4.9: stek.c

```

1 #include "stek.h"
2
3 Cvor *napravi_cvor(char *etiketa)
4 {
5     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
6        alokacije */
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8     if (novi == NULL)
9         return NULL;
10
11     /* Inicijalizacija polja u novom cvoru */
12     if (strlen(etiketa) >= MAX) {
13         fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
14         etiketa[MAX - 1] = '\0';
15     }
16     strcpy(novi->etiketa, etiketa);
17     novi->sledeci = NULL;
18
19     /* Vraca se adresa novog cvora */
20     return novi;
21 }
22
23 void oslobodi_stek(Cvor ** adresa_vrha)
24 {
25     Cvor *pomocni;

```

```
26      /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
27      while (*adresa_vrha != NULL) {
28          /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
29             osloboditi cvor koji predstavlja vrh steka */
30          pomocni = *adresa_vrha;
31          /* Sledeci cvor je novi vrh steka */
32          *adresa_vrha = (*adresa_vrha)->sledeci;
33          free(pomocni);
34      }
35
36      /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
37         adresa_vrha ce pokazivati na NULL. */
38  }
39
40  int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
41  {
42      /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43      Cvor *novi = napravi_cvor(etiketa);
44      if (novi == NULL)
45          return 1;
46
47      /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
48      novi->sledeci = *adresa_vrha;
49      *adresa_vrha = novi;
50      return 0;
51  }
52
53  int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
54  {
55      Cvor *pomocni;
56
57      /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom i
58         vraća se 0 */
59      if (*adresa_vrha == NULL)
60          return 0;
61
62      /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
63         adresu kopira etiketa sa vrha steka */
64      if (etiketa != NULL)
65          strcpy(etiketa, (*adresa_vrha)->etiketa);
66
67      /* Element sa vrha steka se uklanja */
68      pomocni = *adresa_vrha;
69      *adresa_vrha = (*adresa_vrha)->sledeci;
70      free(pomocni);
71
72      /* Vraća se indikator uspesno izvršene radnje */
73      return 1;
74  }
75
76  char *vrh_steka(Cvor * vrh)
```

```

78 {
80     /* Prazan stek nema cvor koji je vrh i vraća se NULL */
82     if (vrh == NULL)
84         return NULL;
86 }

88 void prikazi_stek(Cvor * vrh)
89 {
90     /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
92     for (; vrh != NULL; vrh = vrh->sledeci)
94         printf("<%=s>\n", vrh->etiketa);
96 }

98 int uzmi_etiketu(FILE * f, char *etiketa)
99 {
100     int c;
102     int i = 0;
104     /* Stanje predstavlja informaciju dokle se stalo sa citanjem
106     etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
108     nije zapoceto citanje. */
110     /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
112     OTVORENA ili ZATVORENA. */
114     int stanje = VAN_ETIKETE;
116     int tip;

118     /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
120     to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
122     samo malim slovima. */
124     while ((c = fgetc(f)) != EOF) {
126         switch (stanje) {
128             case VAN_ETIKETE:
130                 if (c == '<')
132                     stanje = PROCITANO_MANJE;
134                 break;
136             case PROCITANO_MANJE:
138                 if (c == '/') {
140                     /* Cita se zatvorena etiketa */
142                     tip = ZATVORENA;
144                 } else {
146                     if (isalpha(c)) {
148                         /* Cita se otvorena etiketa */
150                         tip = OTVORENA;
152                         etiketa[i++] = tolower(c);
154                     }
156                 }
158             }
160         }
162     }
164     /* Od sada se cita etiketa i zato se menja stanje */
166     stanje = U_ETIKETI;
168     break;

```

```

130     case U_ETIKETI:
131         if (isalpha(c) && i < MAX - 1) {
132             /* Ako je procitani karakter slovo i nije prekoracena
133                dozvoljena duzina etikete, procitani karakter se smanjuje
134                i smesta u etiketu */
135             etiketa[i++] = tolower(c);
136         } else {
137             /* Inace, staje se sa citanjem etikete. Korektno se zavrшава
138                niska koja sadrzi procitanu etiketu i vraca se njen tip */
139             etiketa[i] = '\0';
140             return tip;
141         }
142         break;
143     }
144 }
145 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
146    etiketa i vraca se EOF. */
147 return EOF;
148 }

```

Datoteka 4.10: *main.c*

```

#include "stek.h"
2
int main(int argc, char **argv)
4 {
    /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
6       nije procitana. */
    Cvor *vrh = NULL;
    char etiketa[MAX];
    int tip;
    int uparene = 1;
    FILE *f = NULL;
12
    /* Ime datoteke se preuzima iz komandne linije. */
14    if (argc < 2) {
        fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
        exit(0);
    }
18
    /* Datoteka se otvara za citanje */
20    if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
22                argv[1]);
        exit(1);
    }
24
    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
    while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
26        /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
28           etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije

```

```

30     potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
31     koje nemaju sadrzaj (npr link). Zbog jednostavnosti
32     pretpostavlja se da njih nema u HTML dokumentu. */
33     if (tip == OTVORENA) {
34         if (strcmp(etiketa, "br") != 0
35             && strcmp(etiketa, "hr") != 0
36             && strcmp(etiketa, "meta") != 0)
37             if (potisni_na_stek(&vrh, etiketa) == 1) {
38                 fprintf(stderr, "Neuspela alokacija za nov cvor\n");
39                 oslobodi_stek(&vrh);
40                 exit(EXIT_FAILURE);
41             }
42     }
43     /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
44     u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
45     uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
46     je taj uslov ispunjen, skida se sa steka, jer je upravo
47     zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
48     nisu pravilno uparene. */
49     else if (tip == ZATVORENA) {
50         if (vrh_steka(vrh) != NULL
51             && strcmp(vrh_steka(vrh), etiketa) == 0)
52             skini_sa_steka(&vrh, NULL);
53         else {
54             printf("Etikete nisu pravilno uparene\n");
55             printf("(nadjena je etiketa <%s>", etiketa);
56             if (vrh_steka(vrh) != NULL)
57                 printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh));
58             else
59                 printf(" koja nije otvorena)\n");
60             uparene = 0;
61             break;
62         }
63     }
64 }
65 /* Zavrшено je citanje i datoteka se zatvara */
66 fclose(f);
67
68 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
69 da bude prazan. Ukoliko nije, tada postoje etikete koje su
70 ostale otvorene */
71 if (uparene) {
72     if (vrh_steka(vrh) == NULL)
73         printf("Etikete su pravilno uparene!\n");
74     else {
75         printf("Etikete nisu pravilno uparene\n");
76         printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
77         /* Oslobadja se memorija zauzeta stekom */
78         oslobodi_stek(&vrh);
79     }
80 }

```

```

82     return 0;
    }

```

## Rešenje 4.10

Datoteka 4.11: *red.h*

```

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 korisnika i
11    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
18    korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
20     Zahtev nalog;
21     struct cvor *sledeci;
22 } Cvor;
23
24 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
25    poslate adrese i vraća adresu novog cvora ili NULL ako je doslo do
26    greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
27    treba smestiti u novi cvor zbog smestanja manjeg podatka na
28    sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
29    bajtovima(B) u odnosu na strukturu Zahtev. */
30 Cvor *napravi_cvor(Zahtev * zahtev);
31
32 /* Funkcija prazni red oslobadjajuci memoriju koji je red zauzimao */
33 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
34
35 /* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
36    greske pri alokaciji memorije za novi cvor, inace vraća 0. */
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
38                 Zahtev * zahtev);
39
40 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
41    pokazivac razlicit od NULL, tada se u strukturu na koju on
    pokazuje upisuje zahtev koji je upravo skinut sa reda dok u

```

```

43     suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
        suprotnom. */
45 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
        Zahtev * zahtev);
47
48 /* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev korisnika
49    na pocetku reda. Ukoliko je red prazan funkcija vraca NULL. */
        Zahtev *pocetak_reda(Cvor * pocetak);
51
52 /* Funkcija prikazuje sadrzaj reda. */
53 void prikazi_red(Cvor * pocetak);
55 #endif

```

Datoteka 4.12: *red.c*

```

1  #include "red.h"
3  Cvor *napravi_cvor(Zahtev * zahtev)
4  {
5      /* Alocira se memorija za novi cvor liste i proverava uspesnost
        alokacije */
7      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
        if (novi == NULL)
9          return NULL;
11
12     /* Inicijalizacija polja strukture */
        novi->naalog = *zahtev;
13     novi->sledeci = NULL;
15
16     /* Vraca se adresa novog cvora */
        return novi;
17 }
19 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
20 {
21     Cvor *pomocni = NULL;
23
24     /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
        while (*pocetak != NULL) {
25         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
            osloboditi cvor sa pocetka reda */
27         pomocni = *pocetak;
            *pocetak = (*pocetak)->sledeci;
29         free(pomocni);
        }
31     /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
        treba postaviti na NULL. */
33     *kraj = NULL;
35 }

```



```
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,  
38                Zahtev * zahtev)  
39 {  
40     /* Kreira se novi cvor i proverava se uspesnost kreiranja */  
41     Cvor *novi = napravi_cvor(zahtev);  
42     if (novi == NULL)  
43         return 1;  
44  
45     /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na  
46     kraj, to je podjednako efikasno kao dodavanje na pocetak liste  
47     */  
48     if (*adresa_kraja != NULL) {  
49         (*adresa_kraja)->sledeci = novi;  
50         *adresa_kraja = novi;  
51     } else {  
52         /* Ako je red bio ranije prazan */  
53         *adresa_pocetka = novi;  
54         *adresa_kraja = novi;  
55     }  
56  
57     /* Vraca se indikator uspesnog dodavanja */  
58     return 0;  
59 }  
60  
61 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,  
62                  Zahtev * zahtev)  
63 {  
64     Cvor *pomocni = NULL;  
65  
66     /* Ako je red prazan */  
67     if (*adresa_pocetka == NULL)  
68         return 0;  
69  
70     /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje  
71     zahtev koji je na pocetku reda. */  
72     if (zahtev != NULL)  
73         *zahtev = (*adresa_pocetka)->nalog;  
74  
75     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac  
76     na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor  
77     u redu. */  
78     pomocni = *adresa_pocetka;  
79     *adresa_pocetka = (*adresa_pocetka)->sledeci;  
80     free(pomocni);  
81  
82     /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,  
83     potrebno je azurirati i vrednost pokazivaca na adresi  
84     adresa_kraja na NULL */  
85     if (*adresa_pocetka == NULL)  
86         *adresa_kraja = NULL;
```

```

87     return 1;
88 }
89
90 Zahtev *pocetak_reda(Cvor * pocetak)
91 {
92     /* U praznom redu nema zahteva */
93     if (pocetak == NULL)
94         return NULL;
95
96     /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
97     return &(amp;pocetak->nalog);
98 }
99
100 void prikazi_red(Cvor * pocetak)
101 {
102     /* Prikazuje se sadrzaj reda od pocetka prema kraju */
103     for (; pocetak != NULL; pocetak = pocetak->sledeci)
104         printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
105
106     printf("\n");
107 }

```

Datoteka 4.13: *main.c*

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "red.h"
5
6  #define VREME_ZA_PAUZU 5
7
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];
15     int broj_usluzenih = 0;
16
17     /* Sluzbenik evidentira korisnicke zahteve unosanjem njihovog JMBG
18        broja i opisa potrebne usluge. */
19     printf("Sluzbenik evidentira korisnicke zahteve:\n");
20     while (1) {
21
22         /* Ucitava se JMBG */
23         printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
24         if (scanf("%s", nov_zahtev.jmbg) == EOF)
25             break;
26
27         /* Neophodan je poziv funkcije getchar da bi se i nov red nakon

```

```

29         JMBG broja procitao i da bi fgets nakon toga procitala
        ispravan red sa opisom zahteva */
31     getchar();

33     /* Ucitava se opis problema */
    printf("\tOpis problema: ");
    fgets(nov_zahtev.opis, MAX - 1, stdin);
35     /* Ako je poslednji karakter nov red, eliminise se */
    if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
37         nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';

39     /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
    if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
41         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_red(&pocetak, &kraj);
43         exit(EXIT_FAILURE);
    }
45 }

47 /* Otvaranje datoteke za dopisivanje izvestaja */
FILE *izlaz = fopen("izvestaj.txt", "a");
49 if (izlaz == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
51     exit(EXIT_FAILURE);
}

53
55 /* Dokle god ima korisnika u redu, treba ih usluziti */
while (1) {
    sledeci = pocetak_reda(pocetak);
57     /* Ako nema nikog vise u redu, prekida se petlja */
    if (sledeci == NULL)
59         break;

61     printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
    printf("i zahtevom: %s\n", sledeci->opis);

63
65     skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

    broj_usluzenih++;

67     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
69     scanf("%s", odgovor);

71     if (strcmp(odgovor, "Da") == 0)
        dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
73     else
        fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
75                 nov_zahtev.opis);

77     if (broj_usluzenih == VREME_ZA_PAUZU) {
        printf("\nDa li je kraj smene? [Da/Ne] ");
79         scanf("%s", odgovor);
    }

```

```

81     if (strcmp(odgovor, "Da") == 0)
82         break;
83     else
84         broj_usluzenih = 0;
85 }
86 }
87
88 /*****
89  Usluzivanje korisnika moze da se izvrši i na sledeci način:
90
91  while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
92      printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
93          nov_zahtev.jmbg);
94      printf("sa zahtevom: %s\n", nov_zahtev.opis);
95      broj_usluzenih++;
96
97      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
98      scanf("%s", odgovor);
99      if (strcmp(odgovor, "Da") == 0)
100          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
101      else
102          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
103              nov_zahtev.jmbg, nov_zahtev.opis);
104
105      if (broj_usluzenih == VREME_ZA_PAUZU) {
106          printf("\nDa li je kraj smene? [Da/Ne] ");
107          scanf("%s", odgovor);
108          if (strcmp(odgovor, "Da") == 0)
109              break;
110          else
111              broj_usluzenih = 0;
112      }
113  }
114  *****/
115  /* Datoteka vise nije potrebna i treba je zatvoriti. */
116  fclose(izlaz);
117
118  /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
119     neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
120     koju zauzima red sa neobradjenim zahtevima korisnika. */
121  oslobodi_red(&pocetak, &kraj);
122
123  return 0;
124 }

```

### Rešenje 4.11

Datoteka 4.14: *dvostruko\_povezana\_lista.h*

```

2  #ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
3  #define _DVOSTRUKO_POVEZANA_LISTA_H_
4
5  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
6     vrednost i pokazivace na sledeci i prethodni cvor liste. */
7  typedef struct cvor {
8      int vrednost;
9      struct cvor *sledeci;
10     struct cvor *prethodni;
11 } Cvor;
12
13 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
14    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
15    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
16 Cvor *napravi_cvor(int broj);
17
18 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
19    ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na
20    adresi adresa_kraja. */
21 void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);
22
23 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
24    bilo greske pri alokaciji memorije, inace vraca 0. */
25 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
26                             adresa_kraja, int broj);
27
28 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
29    pri alokaciji memorije, inace vraca 0. */
30 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
31                          int broj);
32
33 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
34    novi cvor sa vrednoscu broj. */
35 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
36
37 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
38    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
39 int dodaj_iza(Cvor * tekuci, int broj);
40
41 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
42    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
43    inace vraca 0. */
44 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
45                     broj);
46
47 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
48    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
49    NULL u slucaju da takav cvor ne postoji u listi. */
50 Cvor *pretrazi_listu(Cvor * glava, int broj);
51
52 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
53    U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje

```

```

54     neopadajuće sortirana. Vraća pokazivač na cvor liste u kome je
    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
    */
    Cvor *pretraži_sortiranu_listu(Cvor * glava, int broj);

56
    /* Funkcija briše cvor na koji pokazuje pokazivač tekuci u listi čiji
    pokazivač glava se nalazi na adresi adresa_glave. */
58     void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
    tekuci);
60
    /* Funkcija briše iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivač na glavu liste, koji može biti promenjen u slučaju da se
62     obrise stara glava. */
64     void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
    broj);
66
    /* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na činjenicu da je prosledjena lista neopadajuće
70     sortirana. Azurira pokazivač na glavu liste, koji može biti
    promenjen ukoliko se obrise stara glava liste. */
72     void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
    adresa_kraja, int broj);
74
    /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
    liste, razdvojene zapetama i uokvirene zagradama. */
76     void ispisi_listu(Cvor * glava);
78
    /* Funkcija prikazuje cvorove liste počevši od kraja ka glavi liste.
    */
80     void ispisi_listu_unazad(Cvor * kraj);
82 #endif

```

Datoteka 4.15: *dvostruko\_povezana\_lista.c*

```

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

4
    Cvor *napravi_cvor(int broj)
    {
6        /* Alocira se memorija za novi cvor liste i proverava se uspesnost
        alokacije */
8        Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
        if (novi == NULL)
10            return NULL;

12        /* Inicijalizacija polja strukture */
        novi->vrednost = broj;
        novi->sledeci = NULL;
14
16

```

```
18  /* Vraca se adresa novog cvora */
    return novi;
}

20 void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
22 {
    Cvor *pomocni = NULL;

24     /* Ako lista nije prazna, onda treba osloboditi memoriju */
    while (*adresa_glave != NULL) {
26         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
           osloboditi memoriju cvora koji predstavlja glavu liste */
28         pomocni = (*adresa_glave)->sledeci;
30         free(*adresa_glave);
           /* Sledeci cvor je nova glava liste */
32         *adresa_glave = pomocni;
    }
34     /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
       treba postaviti na NULL */
36     *adresa_kraja = NULL;
}

38 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
40                             adresa_kraja, int broj)
{
42     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
44     if (novi == NULL)
        return 1;

46     /* Sledbenik novog cvora je glava stare liste */
48     novi->sledeci = *adresa_glave;

50     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
       treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
52     krajnji */
    if (*adresa_glave != NULL)
54         (*adresa_glave)->prethodni = novi;
    else
56         *adresa_kraja = novi;

58     /* Novi cvor je nova glava liste */
    *adresa_glave = novi;

60     /* Vraca se indikator uspesnog dodavanja */
62     return 0;
}

64 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
66                         int broj)
{
68     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
```

```

70 Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
        return 1;
72
74 /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
    ujedno i cela lista. Azurira se vrednost na koju pokazuju
    adresa_glave i adresa_kraja */
76 if (*adresa_glave == NULL) {
    *adresa_glave = novi;
    *adresa_kraja = novi;
78 } else {
80 /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
    sledbenik poslednjeg cvora i azurira se samo pokazivac na kraj
82 liste */
    (*adresa_kraja)->sledeci = novi;
    novi->prethodni = (*adresa_kraja);
    *adresa_kraja = novi;
86 }

88 /* Vraca se indikator uspesnog dodavanja */
    return 0;
90 }

92 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
94 /* U praznoj listi nema takvog mesta i vraca se NULL */
    if (glava == NULL)
        return NULL;
96
98 /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
    pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
100 vrednost vecu ili jednaku od vrednosti novog cvora.

102 Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
    provera da li se doslo do poslednjeg cvora liste pre nego sto se
104 proverci vrednost u sledecem cvoru jer u slucaju poslednjeg,
    sledeci ne postoji pa ni njegova vrednost. */
106 while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
    glava = glava->sledeci;
108
110 /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
    poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
    vrednost vecu od broj */
112 return glava;
}

114 int dodaj_iza(Cvor * tekuci, int broj)
116 {
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
        return 1;
120

```



```
122 novi->sledeci = tekuci->sledeci;
123 novi->prethodni = tekuci;
124
125 /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
126    a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
127    na ispravne adrese */
128 if (tekuci->sledeci != NULL)
129     tekuci->sledeci->prethodni = novi;
130 tekuci->sledeci = novi;
131
132 /* Vraca se indikator uspesnog dodavanja */
133 return 0;
134 }
135
136 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
137                     broj)
138 {
139     /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste
140        */
141     if (*adresa_glave == NULL) {
142         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
143         Cvor *novi = napravi_cvor(broj);
144         if (novi == NULL)
145             return 1;
146
147         /* Azuriraju se vrednosti pocetka i kraja liste */
148         *adresa_glave = novi;
149         *adresa_kraja = novi;
150
151         /* Vraca se indikator uspesnog dodavanja */
152         return 0;
153     }
154
155     /* Ukoliko je vrednost glave liste veca ili jednaka od nove
156        vrednosti onda novi cvor treba staviti na pocetak liste */
157     if ((*adresa_glave)->vrednost >= broj) {
158         return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
159     }
160
161     /* Pronazi se cvor iza koga treba uvezati novi cvor */
162     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
163     /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
164     if (dodaj_iza(pomocni, broj) == 1)
165         return 1;
166     /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
167        novi cvor poslednji u listi. */
168     if (pomocni == *adresa_kraja)
169         *adresa_kraja = novi;
170
171     return 0;
172 }
```

```

172 Cvor *pretrazi_listu(Cvor * glava, int broj)
173 {
174     /* Obilaze se cvorovi liste */
175     for (; glava != NULL; glava = glava->sledeci)
176         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
177            se obustavlja */
178         if (glava->vrednost == broj)
179             return glava;
180
181     /* Nema trazenog broja u listi i vraca se NULL */
182     return NULL;
183 }
184
185 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
186 {
187     /* Obilaze se cvorovi liste */
188     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
189     for (; glava != NULL && glava->vrednost <= broj;
190           glava = glava->sledeci)
191         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
192            se obustavlja */
193         if (glava->vrednost == broj)
194             return glava;
195
196     /* Nema trazenog broja u listi i bice vrateno NULL */
197     return NULL;
198 }
199
200 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
201    lako realizovati jer on sadrzi pokazivace na svog sledbenika i
202    prethodnika u listi. U funkciji se bise cvor zadat argumentom
203    tekuci */
204 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
205                   tekuci)
206 {
207     /* Ako je tekuci NULL pokazivac, nema sta da se brise */
208     if (tekuci == NULL)
209         return;
210
211     /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da
212        njegov sledbenik bude sledbenik tekuceg cvora */
213     if (tekuci->prethodni != NULL)
214         tekuci->prethodni->sledeci = tekuci->sledeci;
215
216     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
217        treba da bude prethodnik tekuceg cvora */
218     if (tekuci->sledeci != NULL)
219         tekuci->sledeci->prethodni = tekuci->prethodni;
220
221     /* Ako je glava cvor koji se brise, nova glava liste ce biti
222        sledbenik stare glave */

```

```

224     if (tekuci == *adresa_glave)
        *adresa_glave = tekuci->sledeci;
226
        /* Ako je cvor koji se brise poslednji u listi, azurira se i
228         pokazivac na kraj liste */
        if (tekuci == *adresa_kraja)
230             *adresa_kraja = tekuci->prethodni;

232     /* Oslobadja se dinamički alociran prostor za cvor tekuci */
    free(tekuci);
234 }

236 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
    )
    {
238     Cvor *tekuci = *adresa_glave;

240     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju, takvi
        cvorovi se brisu iz liste. */
242     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
244 }

246 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
        adresa_kraja, int broj)
248 {
    Cvor *tekuci = *adresa_glave;
250
    /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
252     takvi cvorovi se brisu iz liste. */
    while ((tekuci =
254         pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
256 }

258 void ispisi_listu(Cvor * glava)
    {
260     putchar('[');
        /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
262         pocetka prema kraju liste */
        for (; glava != NULL; glava = glava->sledeci) {
264             printf("%d", glava->vrednost);
                if (glava->sledeci != NULL)
266                 printf(", ");
        }

268     printf("]\n");
270 }

272 void ispisi_listu_unazad(Cvor * kraj)
    {
274     putchar('[');

```

```

276  /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
    prema pocetku liste */
278  for (; kraj != NULL; kraj = kraj->prethodni) {
    printf("%d", kraj->vrednost);
    if (kraj->prethodni != NULL)
280     printf(", ");
    }
282  printf("]\n");
}

```

Datoteka 4.16: *main\_a.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"

5  /* 1) Glavni program */
   int main()
7  {
    /* Lista je prazna na pocetku */
    /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
    da bi operacije poput dodavanja na kraj liste i ispisivanja
11   liste unazad bile efikasne poput dodavanja na pocetak liste i
    ispisivanja liste od pocetnog do poslednjeg cvora. */
13   Cvor *glava = NULL;
    Cvor *kraj = NULL;
15   Cvor *trazeni = NULL;
    int broj;

17   /* Testira se funkcija za dodavanja novog broja na pocetak liste */
    printf("Unesite brojeve: (za kraj CTRL+D)\n");
19   while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
    memorije za novi cvor. Memoriju alociranu za cvorove liste
23   treba osloboditi pre napustanja programa */
    if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
25     fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
        oslobodi_listu(&glava, &kraj);
27     exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
29     ispisi_listu(glava);
31 }

33 /* Testira se funkcija za pretragu liste */
    printf("\nUnesite broj koji se trazi u listi: ");
35     scanf("%d", &broj);

37 /* Pokazivac trazeni dobija vrednost rezultata pretrage */
    trazeni = pretrazi_listu(glava, broj);
39     if (trazeni == NULL)

```

```

    printf("Broj %d se ne nalazi u listi!\n", broj);
41 else
    printf("Trazeni broj %d je u listi!\n", trazen->vrednost);
43
/* Ispisuje se lista unazad */
45 printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);
47
/* Oslobadja se memorija zauzeta za cvorove liste */
49 oslobodi_listu(&glava, &kraj);
51
return 0;
}

```

Datoteka 4.17: *main\_b.c*

```

#include <stdio.h>
2 #include <stdlib.h>
#include "dvostruko_povezana_lista.h"
4
/* 2) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku. */
    Cvor *glava = NULL;
    Cvor *kraj = NULL;
10     int broj;
12
    /* Testira se funkcija za dodavanja novog broja na kraj liste */
14     printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
            memorije za novi cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa */
18         if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
20             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
                oslobodi_listu(&glava, &kraj);
22             exit(EXIT_FAILURE);
        }
24         printf("\tLista: ");
        ispisi_listu(glava);
26     }

28     /* Testira se funkcija za brisanje elemenata iz liste */
    printf("\nUnesite broj koji se brise iz liste: ");
30     scanf("%d", &broj);

32     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
        procitanom sa ulaza */
34     obrisi_cvor(&glava, &kraj, broj);

```

```

36     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);
38
    /* Ispisuje se lista unazad */
40     printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);
42
    /* Oslobadja se memorija zauzeta za cvorove liste */
44     oslobodi_listu(&glava, &kraj);
46
    return 0;
}

```

Datoteka 4.18: *main.c.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"
5
   /* 3) Glavni program */
   int main()
7  {
   /* Lista je prazna na pocetku */
9   Cvor *glava = NULL;
   Cvor *kraj = NULL;
11  Cvor *trazeni = NULL;
   int broj;
13
   /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
15    bude uredjena neopadajuće */
   printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
17   while (scanf("%d", &broj) > 0) {
   /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
19    memorije za novi cvor. Memoriju alociranu za cvorove liste
    treba osloboditi pre napustanja programa */
21    if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
        fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23        oslobodi_listu(&glava, &kraj);
        exit(EXIT_FAILURE);
25    }
        printf("\tLista: ");
27        ispisi_listu(glava);
    }
29
   /* Testira se funkcija za pretragu liste */
31   printf("\nUnesite broj koji se trazi u listi: ");
   scanf("%d", &broj);
33
   /* Pokazivac trazeni dobija vrednost rezultata pretrage */
35   trazeni = pretrazi_listu(glava, broj);
   if (trazeni == NULL)

```

```

37     printf("Broj %d se ne nalazi u listi!\n", broj);
    else
39     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

41     /* Testira se funkcija za brisanje elemenata iz liste */
    printf("\nUnesite broj koji se brise iz liste: ");
43     scanf("%d", &broj);

45     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
        procitanom sa ulaza */
47     obrisi_cvor_sortirane_liste(&glava, &kraj, broj);

49     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

51     /* Ispisuje se lista unazad */
53     printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);

55     /* Oslobadja se memorija zauzeta za cvorove liste */
57     oslobodi_listu(&glava, &kraj);

59     return 0;
}

```

### Rešenje 4.14

Datoteka 4.19: *stabla.h*

```

1  #ifndef _STABLA_H_
2  #define _STABLA_H_ 1

4  /* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
    stabla */
6  typedef struct cvor {
    int broj;
8     struct cvor *levo;
    struct cvor *desno;
10 } Cvor;

12 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
    inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
    funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
18 do greske */
    int dodaj_u_stablo(Cvor ** adresa_korena, int broj);
20

    /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */

```

```

22 Cvor *pretrazi_stablo(Cvor * koren, int broj);

24 /* e) Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
    stablu */
26 Cvor *pronadji_najmanji(Cvor * koren);

28 /* f) Funkcija koja pronalazi cvor koji sadrzi najveću vrednost u
    stablu */
30 Cvor *pronadji_najveci(Cvor * koren);

32 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
void obrisi_element(Cvor ** adresa_korena, int broj);

34
/* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
36 postablo - Koren - Desno podstablo ) */
void ispisi_stablo_infiksno(Cvor * koren);

38
/* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
40 Levo podstablo - Desno podstablo ) */
void ispisi_stablo_prefiksno(Cvor * koren);

42
/* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
44 podstablo - Desno postablo - Koren ) */
void ispisi_stablo_postfiksno(Cvor * koren);

46
/* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
48 void oslobodi_stablo(Cvor ** adresa_korena);

50 #endif

```

Datoteka 4.20: *stabla.c*

```

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

4
Cvor *napravi_cvor(int broj)
6 {
    /* Alocira se memorija za novi cvor i proverava se uspesnost
8     alokacije. */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
        return NULL;

12
    /* Inicijalizuju se polja novog cvora. */
14     novi->broj = broj;
    novi->levo = NULL;
16     novi->desno = NULL;

18
    /* Vraca se adresa novog cvora. */
    return novi;

```



```

20 }
22 int dodaj_u_stablo(Cvor ** adresa_korena, int broj)
23 {
24     /* Ako je stablo prazno */
25     if (*adresa_korena == NULL) {
26
27         /* Kreira se novi cvor */
28         Cvor *novi_cvor = napravi_cvor(broj);
29
30         /* Proverava se uspesnost kreiranja */
31         if (novi_cvor == NULL) {
32
33             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost */
34             return 1;
35         }
36         /* Inace ... */
37
38         /* Novi cvor se proglašava korenom stabla */
39         *adresa_korena = novi_cvor;
40
41         /* I vraca se indikator uspesnosti kreiranja */
42         return 0;
43     }
44
45     /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati broj */
46
47     /* Ako je zadata vrednost manja od vrednosti korena */
48     if (broj < (*adresa_korena)->broj)
49     {
50
51         /* Broj se dodaje u levo podstablo */
52         return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
53     }
54     else
55     {
56         /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
57            dodaje u desno podstablo */
58         return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
59     }
60 }
61
62 Cvor *pretrazi_stablo(Cvor * koren, int broj)
63 {
64     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
65     if (koren == NULL)
66         return NULL;
67
68     /* Ako je trazena vrednost sadrzana u korenu */
69     if (koren->broj == broj) {
70
71         /* Prekidamo pretragu */

```

```
72     return koren;
73 }
74
75 /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
76 if (broj < koren->broj)
77
78     /* Pretraga se nastavlja u levom podstablu */
79     return pretrazi_stablo(koren->levo, broj);
80
81 else
82     /* U suprotnom, pretraga se nastavlja u desnom podstablu */
83     return pretrazi_stablo(koren->desno, broj);
84 }
85
86 Cvor *pronadji_najmanji(Cvor * koren)
87 {
88
89     /* Ako je stablo prazno, prekida se pretraga */
90     if (koren == NULL)
91         return NULL;
92
93     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
94        levo od njega */
95
96     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
97        najmanju vrednost */
98     if (koren->levo == NULL)
99         return koren;
100
101     /* Inace, pretragu treba nastaviti u levom podstablu */
102     return pronadji_najmanji(koren->levo);
103 }
104
105 Cvor *pronadji_najveci(Cvor * koren)
106 {
107
108     /* Ako je stablo prazno, prekida se pretraga */
109     if (koren == NULL)
110         return NULL;
111
112     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
113        desno od njega */
114
115     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
116        najveću vrednost */
117     if (koren->desno == NULL)
118         return koren;
119
120     /* Inace, pretragu treba nastaviti u desnom podstablu */
121     return pronadji_najveci(koren->desno);
122 }
123
124 void obrisi_element(Cvor ** adresa_korena, int broj)
```

```

124 {
125     Cvor *pomocni_cvor = NULL;
126
127     /* Ako je stablo prazno, brisanje nije primenljivo */
128     if (*adresa_korena == NULL)
129         return;
130
131     /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
132        stabla, ona se eventualno nalazi u levom podstablu, pa treba
133        rekursivno primeniti postupak na levo podstablo. Koren ovako
134        modifikovanog stabla je nepromenjen. */
135     if (broj < (*adresa_korena)->broj) {
136         obrisi_element(&(*adresa_korena)->levo, broj);
137         return;
138     }
139
140     /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
141        stabla, ona se eventualno nalazi u desnom podstablu pa treba
142        rekursivno primeniti postupak na desno podstablo. Koren ovako
143        modifikovanog stabla je nepromenjen. */
144     if ((*adresa_korena)->broj < broj) {
145         obrisi_element(&(*adresa_korena)->desno, broj);
146         return;
147     }
148
149     /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
150        jednaka broju koji se brise (tj. slucaj kada treba obrisati
151        koren) */
152
153     /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
154        prazno stablo (vraca se NULL) */
155     if ((*adresa_korena)->levo == NULL
156         && (*adresa_korena)->desno == NULL) {
157         free(*adresa_korena);
158         *adresa_korena = NULL;
159         return;
160     }
161
162     /* Ako koren ima samo levog sina, tada se brisanje vrši tako što se
163        brise koren, a novi koren postaje levi sin */
164     if ((*adresa_korena)->levo != NULL
165         && (*adresa_korena)->desno == NULL) {
166         pomocni_cvor = (*adresa_korena)->levo;
167         free(*adresa_korena);
168         *adresa_korena = pomocni_cvor;
169         return;
170     }
171
172     /* Ako koren ima samo desnog sina, tada se brisanje vrši tako što
173        se brise koren, a novi koren postaje desni sin */
174     if ((*adresa_korena)->desno != NULL
175         && (*adresa_korena)->levo == NULL) {

```

```
176     pomocni_cvor = (*adresa_korena)->desno;
177     free(*adresa_korena);
178     *adresa_korena = pomocni_cvor;
179     return;
180 }
181
182 /* Slucaj kada koren ima oba sina - najpre se potrazi sledbenik
183    korena (u smislu poretka) u stablu. To je upravo po vrednosti
184    najmanji cvor u desnom podstablu. On se moze pronaci npr.
185    funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
186    vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
187    broj koji se brise). Zatim se prosto rekursivno pozove funkcija
188    za brisanje na desno podstablo. S obzirom da u njemu treba
189    obrisati najmanji element, a on zasigurno ima najvise jednog
190    potomka, jasno je da ce njegovo brisanje biti obavljeno na jedan
191    od jednostavnijih nacina koji su gore opisani. */
192 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
193 (*adresa_korena)->broj = pomocni_cvor->broj;
194 pomocni_cvor->broj = broj;
195 obris_element(&(*adresa_korena)->desno, broj);
196 }
197
198 void ispisi_stablo_infiksno(Cvor * koren)
199 {
200     /* Ako stablo nije prazno */
201     if (koren != NULL) {
202
203         /* Prvo se ispisuju svi cvorovi levo od korena */
204         ispisi_stablo_infiksno(koren->levo);
205
206         /* Zatim se ispisuje vrednost u korenu */
207         printf("%d ", koren->broj);
208
209         /* Na kraju se ispisuju cvorovi desno od korena */
210         ispisi_stablo_infiksno(koren->desno);
211     }
212 }
213
214 void ispisi_stablo_prefiksno(Cvor * koren)
215 {
216     /* Ako stablo nije prazno */
217     if (koren != NULL) {
218
219         /* Prvo se ispisuje vrednost u korenu */
220         printf("%d ", koren->broj);
221
222         /* Zatim se ispisuju svi cvorovi levo od korena */
223         ispisi_stablo_prefiksno(koren->levo);
224
225         /* Na kraju se ispisuju svi cvorovi desno od korena */
226         ispisi_stablo_prefiksno(koren->desno);
227     }
228 }
```

```

228 }
230 void ispisi_stablo_postfiksno(Cvor * koren)
231 {
232     /* Ako stablo nije prazno */
233     if (koren != NULL) {
234
235         /* Prvo se ispisuju svi cvorovi levo od korena */
236         ispisi_stablo_postfiksno(koren->levo);
237
238         /* Zatim se ispisuju svi cvorovi desno od korena */
239         ispisi_stablo_postfiksno(koren->desno);
240
241         /* Na kraju se ispisuje vrednost u korenu */
242         printf("%d ", koren->broj);
243     }
244 }
246 void oslobodi_stablo(Cvor ** adresa_korena)
247 {
248     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
249     if (*adresa_korena == NULL)
250         return;
251
252     /* Inace ... */
253     /* Oslobadja se memorija zauzeta levim podstablom */
254     oslobodi_stablo(&(*adresa_korena)->levo);
255
256     /* Oslobadja se memorija zauzeta desnim podstablom */
257     oslobodi_stablo(&(*adresa_korena)->desno);
258
259     /* Oslobadja se memorija zauzeta korenom */
260     free(*adresa_korena);
261
262     /* Proglasava se stablo praznim */
263     *adresa_korena = NULL;
264 }

```

Datoteka 4.21: *main.c*

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"
4
5  int main()
6  {
7      Cvor *koren;
8      int n;
9      Cvor *trazeni_cvor;

```

```

11  /* Proglasava se stablo praznim */
    koren = NULL;

13

15  /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
    dodavanja */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
17  while (scanf("%d", &n) != EOF) {
        if (dodaj_u_stablo(&koren, n) == 1) {
19            fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
            oslobodi_stablo(&koren);
21            return 0;
        }
23    }

25  /* Generisu se trazeni ispisi: */
    printf("\nInfiksni ispis: ");
    ispisi_stablo_infiksno(koren);
    printf("\nPrefiksni ispis: ");
    ispisi_stablo_prefiksno(koren);
    printf("\nPostfiksni ispis: ");
    ispisi_stablo_postfiksno(koren);

33  /* Demonstrira se rad funkcije za pretragu */
    printf("\nTrazi se broj: ");
    scanf("%d", &n);
    trazeni_cvor = pretrazi_stablo(koren, n);
37  if (trazeni_cvor == NULL)
        printf("Broj se ne nalazi u stablu!\n");

39  else
41      printf("Broj se nalazi u stablu!\n");

43  /* Demonstrira se rad funkcije za brisanje */
    printf("Brise se broj: ");
    scanf("%d", &n);
    obrisi_element(&koren, n);
    printf("Rezultujuce stablo: ");
    ispisi_stablo_infiksno(koren);
49  printf("\n");

51  /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);

53

    return 0;
55 }

```

### Rešenje 4.15

```

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

```

```

4  #include <ctype.h>

6  #define MAX 50

8  /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
12     char *rec;
14     int broj;
16     struct cvor *levo;
18     struct cvor *desno;
20 } Cvor;

22 /* Funkcija koja kreira novi cvor stabla */
24 Cvor *napravi_cvor(char *rec)
26 {
28     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
30     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
32     if (novi_cvor == NULL)
34         return NULL;

36     /* Alocira se memorija za zadatu rec: potrebno je rezervirati
       memoriju za svaki karakter reci ukljucujuci i terminirajucu nulu
       */
38     novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
40     if (novi_cvor->rec == NULL) {
42         free(novi_cvor);
44         return NULL;
46     }

48     /* Inicijalizuju se polja u novom cvoru */
50     strcpy(novi_cvor->rec, rec);
52     novi_cvor->broj = 1;
54     novi_cvor->levo = NULL;
56     novi_cvor->desno = NULL;

58     /* Vraca se adresa novog cvora */
60     return novi_cvor;
62 }

64 /* Funkcija koja dodaje novu rec u stablo - ukoliko je dodavanje
   uspesno povratna vrednost je 0, u suprotnom povratna vrednost je 1
   */
66 int dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
68 {
70     /* Ako je stablo prazno */
72     if (*adresa_korena == NULL) {
74         /* Kreira se cvor koji sadrzi zadatu rec */
76         Cvor *novi_cvor = napravi_cvor(rec);
78         /* Proverava se uspesnost kreiranja novog cvora */
80         if (novi_cvor == NULL) {

```

```

56      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
57      */
58      return 1;
59  }
60  /* Inace... */
61  /* Novi cvor se proglašava korenom stabla */
62  *adresa_korena = novi_cvor;
63
64  /* I vraca se indikator uspešnog dodavanja */
65  return 0;
66  }
67
68  /* Ako stablo nije prazno, trazi odgovarajuca pozicija za novu rec
69  */
70
71  /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
72  levo podstablo */
73  if (strcmp(rec, (*adresa_korena)->rec) < 0)
74      return dodaj_u_stablo(&(*adresa_korena)->levo, rec);
75
76  else
77      /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
78      desno podstablo */
79      if (strcmp(rec, (*adresa_korena)->rec) > 0)
80          return dodaj_u_stablo(&(*adresa_korena)->desno, rec);
81
82  else
83      /* Ako je rec jednaka reci u korenu, uvecava se njen broj
84      pojavljivanja */
85      (*adresa_korena)->brojac++;
86  }
87
88  /* Funkcija koja oslobadja memoriju zauzetu stablom */
89  void oslobodi_stablo(Cvor ** adresa_korena)
90  {
91      /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
92      if (*adresa_korena == NULL)
93          return;
94
95      /* Inace ... */
96      /* Oslobadja se memorija zauzeta levim podstablom */
97      oslobodi_stablo(&(*adresa_korena)->levo);
98
99      /* Oslobadja se memorija zauzeta desnim podstablom */
100     oslobodi_stablo(&(*adresa_korena)->desno);
101
102     /* Oslobadja se memorija zauzeta korenom */
103     free((*adresa_korena)->rec);
104     free(*adresa_korena);
105
106     /* Stablo se proglašava praznim */
107     *adresa_korena = NULL;

```



```

108 }
109
110 /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
111    sa najvećim brojem pojavljivanja) */
112 Cvor *nadjij_najfrekventniju_rec(Cvor * koren)
113 {
114     Cvor *max, *max_levo, *max_desno;
115
116     /* Ako je stablo prazno, prekida se sa pretragom */
117     if (koren == NULL)
118         return NULL;
119
120     /* Pronalazi se najfrekventnija rec u levom podstablu */
121     max_levo = nadjij_najfrekventniju_rec(koren->levo);
122
123     /* Pronalazi se najfrekventnija rec u desnom podstablu */
124     max_desno = nadjij_najfrekventniju_rec(koren->desno);
125
126     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
127        podstabla, korena i desnog podstabla */
128     max = koren;
129     if (max_levo != NULL && max_levo->brojac > max->brojac)
130         max = max_levo;
131     if (max_desno != NULL && max_desno->brojac > max->brojac)
132         max = max_desno;
133
134     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
135     return max;
136 }
137
138 /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
139    pracen brojem pojavljivanja */
140 void prikazi_stablo(Cvor * koren)
141 {
142     /* Ako je stablo prazno, završava se sa ispisom */
143     if (koren == NULL)
144         return;
145
146     /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz levog
147        podstabla */
148     prikazi_stablo(koren->levo);
149
150     /* Zatim rec iz korena */
151     printf("%s: %d\n", koren->rec, koren->brojac);
152
153     /* I nastavlja se sa ispisom reci iz desnog podstabla */
154     prikazi_stablo(koren->desno);
155 }
156
157 /* Funkcija ucitava sledeću rec iz zadate datoteke f i upisuje je u
158    niz rec. Maksimalna dužina reci je određena argumentom max.
159    Funkcija vraća EOF ako u datoteci nema više reci ili 0 u

```

```

160     suprotnom. Rec je niz malih ili velikih slova. */
161 int procitaj_rec(FILE * f, char rec[], int max)
162 {
163     /* Karakter koji se cita */
164     int c;
165
166     /* Indeks pozicije na koju se smesta procitani karakter */
167     int i = 0;
168
169     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
170        nije stiglo do kraja datoteke... */
171     while (i < max - 1 && (c = fgetc(f)) != EOF) {
172         /* Proverava se da li je procitani karakter slovo */
173         if (isalpha(c))
174             /* Ako jeste, smesta se u niz - pritom se vrsi konverzija u
175                mala slova jer program treba da bude neosetljiv na razliku
176                izmedju malih i velikih slova */
177             rec[i++] = tolower(c);
178
179         else
180             /* Ako nije, proverava se da li je procitano barem jedno slovo
181                nove reci */
182             /* Ako jeste, prekida se sa citanjem */
183             if (i > 0)
184                 break;
185
186         /* U suprotnom se ide na sledecu iteraciju */
187     }
188
189     /* Dodaje se na rec terminirajuca nula */
190     rec[i] = '\0';
191
192     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
193     return i > 0 ? 0 : EOF;
194 }
195
196 int main(int argc, char **argv)
197 {
198     Cvor *koren = NULL, *max;
199     FILE *f;
200     char rec[MAX];
201
202     /* Provera da li je navedeno ime datoteke prilikom pokretanja
203        programa */
204     if (argc < 2) {
205         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
206         exit(EXIT_FAILURE);
207     }
208
209     /* Priprema datoteke za citanje */
210     if ((f = fopen(argv[1], "r")) == NULL) {
211         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",

```

```

212         argv[1]);
        exit(EXIT_FAILURE);
    }
214
    /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
    pretrage uz proveru uspesnosti dodavanja */
216    while (procitaj_rec(f, rec, MAX) != EOF) {
218        if (dodaj_u_stablo(&koren, rec) == 1) {
            fprintf(stderr, "Neuspelo dodavanje reci %s\n", rec);
220            oslobodi_stablo(&koren);
            return 0;
222        }
    }
224
    /* Posto je citanjem reci zavrшено, zatvara se datoteka */
226    fclose(f);

228    /* Prikazuju se sve reci iz teksta i brojevi njihovih
    pojavljivanja. */
230    prikazi_stablo(koren);

232    /* Pronalazi se najfrekventnija rec */
    max = naji_najfrekventniju_rec(koren);
234

    /* Ako takve reci nema... */
236    if (max == NULL)

238        /* Ispisuje se odgovarajuće obavestjenje */
        printf("U tekstu nema reci!\n");
240
    else
242        /* Inace, ispisuje se broj pojavljivanja reci */
        printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
244            max->rec, max->brojac);

246    /* Oslobadja se dinamički alociran prostor za stablo */
    oslobodi_stablo(&koren);
248

    return 0;
250 }

```

### 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
19  /* Funkcija koja kreira novi cvora stabla */
20  Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
    {
22      /* Alocira se memorija za novi cvor i proverava se uspesnost
        alokacije. */
24      Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
        if (novi_cvor == NULL)
26          return NULL;

28      /* Inicijalizuju se polja novog cvora */
        strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
30        strcpy(novi_cvor->telefon, telefon);
        novi_cvor->levo = NULL;
32        novi_cvor->desno = NULL;

34      /* Vraca se adresa novog cvora */
        return novi_cvor;
36    }

38  /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
40    povratna vrednost je 1 */
    int
42    dodaj_u_stablo(Cvor **adresa_korena, char *ime_i_prezime,
        char *telefon)
44    {
        /* Ako je stablo prazno */
46        if (*adresa_korena == NULL) {
            /* Kreira se novi cvor */
            Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
            /* Proverava se uspesnost kreiranja novog cvora */
50            if (novi_cvor == NULL) {
                /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
                    */
52                return 1;
            }
54            /* Inace... */
            /* Novi cvor se proglašava korenom stabla */
            *adresa_korena = novi_cvor;
56
58            /* I vraca se indikator uspesnog dodavanja */
60            return 0;

```

```
62     }
63
64     /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
65        unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
66        treba da bude pretrazivacko po ovom polju */
67
68     /* Ako je zadato ime i prezime leksikografski manje od imena i
69        prezimena sadrzanog u korenu, podaci se dodaju u levo podstablo
70        */
71     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
72         < 0)
73         return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
74                               telefon);
75
76     else
77         /* Ako je zadato ime i prezime leksikografski vece od imena i
78            prezimena sadrzanog u korenu, podaci se dodaju u desno
79            podstablo */
80         if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
81             return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
82                                   telefon);
83     }
84
85     /* Funkcija koja oslobadja memoriju zauzetu stablom */
86     void oslobodi_stablo(Cvor ** adresa_korena)
87     {
88         /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
89         if (*adresa_korena == NULL)
90             return;
91
92         /* Inace ... */
93         /* Oslobadja se memorija zauzeta levim podstablom */
94         oslobodi_stablo(&(*adresa_korena)->levo);
95
96         /* Oslobadja se memorija zauzeta desnim podstablom */
97         oslobodi_stablo(&(*adresa_korena)->desno);
98
99         /* Oslobadja se memorija zauzeta korenom */
100        free(*adresa_korena);
101
102        /* Stablo se proglašava praznim */
103        *adresa_korena = NULL;
104    }
105
106    /* Funkcija koja ispisuje imenik u leksikografskom poretku */
107    /* Napomena: ova funkcija nije trazena u zadatku ali se moze
108       koristiti za proveru da li je stablo lepo kreirano ili ne */
109    void prikazi_stablo(Cvor * koren)
110    {
111        /* Ako je stablo prazno, završava se sa ispisom */
112        if (koren == NULL)
113            return;
```

```
112      /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
114         podstabla */
      prikazi_stablo(koren->levo);
116
      /* Zatim se ispisuju podaci iz korena */
118      printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
120
      /* I nastavlja se sa ispisom podataka iz desnog podstabla */
      prikazi_stablo(koren->desno);
122  }

124  /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje ime
      i prezime i broj telefona u odgovarajuće nizove. Maksimalna dužina
126     imena i prezimena određena je konstantom MAX_IME_PREZIME, a
      maksimalna dužina broja telefona konstantom MAX_CIFARA. Funkcija
128     vraća EOF ako nema više kontakata ili 0 u suprotnom. */
  int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
130  {
      /* Karakter koji se cita */
132      int c;

134      /* Indeks pozicije na koju se smesta procitani karakter */
      int i = 0;
136
      /* Linije datoteke koje se obradjuju su formata Ime Prezime
138         BrojTelefona */

140      /* Preskacu se eventualne praznine sa pocetka linije datoteke */
      while ((c = fgetc(f)) != EOF && isspace(c));
142
      /* Prvo procitano slovo upisuje se u ime i prezime */
144      if (!feof(f))
          ime_i_prezime[i++] = c;
146
      /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
148         cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
      Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
150     procitanog karaktera i da se slucajno ne dodje do kraja datoteke
      */
152      while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
          if (!isdigit(c))
154              ime_i_prezime[i++] = c;

          else if (i > 0)
156              break;
158      }

160      /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
      blanko karaktera */
162      ime_i_prezime[--i] = '\0';
```

```
164  /* I pocinje se sa citanjem broja telefona */
    i = 0;
166
    /* Upisuje se cifra koja je vec procitana */
168  telefon[i++] = c;

170  /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
    karaktera cije prisustvo nije dozvoljeno u broju telefona */
172  while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
    if (c == '/' || c == '-' || isdigit(c))
174      telefon[i++] = c;
    else
176      break;

178  /* Upisuje se terminirajuca nula */
    telefon[i] = '\0';
180
    /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
182  return !feof(f) ? 0 : EOF;
}

184
/* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
*/
186 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
188 {
    /* Ako je imenik prazan, završava se sa pretragom */
190  if (koren == NULL)
    return NULL;

192
    /* Ako je trazeno ime i prezime sadržano u korenu, takodje se
    završava sa pretragom */
194  if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
196  return koren;

198
    /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
    korenu pretraga se nastavlja levo */
200  if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
    return pretrazi_imenik(koren->levo, ime_i_prezime);
202
    else
204  /* u suprotnom, pretraga se nastavlja desno */
    return pretrazi_imenik(koren->desno, ime_i_prezime);
206 }

208 int main(int argc, char **argv)
{
210  char ime_datoteke[MAX_IME_DATOTEKE];
    Cvor *koren = NULL;
212  Cvor *trazeni;
    FILE *f;
214  char ime_i_prezime[MAX_IME_I_PREZIME];
    char telefon[MAX_CIFARA];
```

```

216 char c;
217 int i;
218
219 /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
220 printf("Unesite ime datoteke: ");
221 scanf("%s", ime_datoteke);
222 getchar();
223 if ((f = fopen(ime_datoteke, "r")) == NULL) {
224     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
225             ime_datoteke);
226     exit(EXIT_FAILURE);
227 }
228
229 /* Citaju se podaci iz datoteke i smestanju u binarno stablo
230 pretrage uz proveru uspesnosti dodavanja */
231 while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
232     if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
233         fprintf(stderr, "Neuspelo dodavanje podataka za osobu %s\n",
234                 ime_i_prezime);
235         oslobodi_stablo(&koren);
236         return 0;
237     }
238
239 /* Zatvara se datoteka */
240 fclose(f);
241
242 /* Omogucava se pretraga imenika */
243 while (1) {
244     /* Ucitavaja se ime i prezime */
245     printf("Unesite ime i prezime: ");
246     i = 0;
247     while ((c = getchar()) != '\n')
248         ime_i_prezime[i++] = c;
249     ime_i_prezime[i] = '\0';
250
251     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
252     funkcionalnost */
253     if (strcmp(ime_i_prezime, "KRAJ") == 0)
254         break;
255
256     /* Inace se ispisuje rezultat pretrage */
257     trazeni = pretrazi_imenik(koren, ime_i_prezime);
258     if (trazeni == NULL)
259         printf("Broj nije u imeniku!\n");
260     else
261         printf("Broj je: %s \n", trazeni->telefon);
262 }
263
264 /* Oslobadja se memorija zauzeta imenikom */
265 oslobodi_stablo(&koren);
266
267 return 0;

```



268 }

## Rešenje 4.17

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX 51
6
7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8     studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9     jezika i redom pokazuje na levo i desno podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX];
12     char prezime[MAX];
13     double uspeh;
14     double matematika;
15     double jezik;
16     struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
18 } Cvor;
19
20 /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
22                    double matematika, double jezik)
23 {
24     /* Alocira se memorija za novi cvor i proverava se uspesnost
25        alokacije. */
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;
29
30     /* Inicijalizuju se polja strukture */
31     strcpy(novi->ime, ime);
32     strcpy(novi->prezime, prezime);
33     novi->uspeh = uspeh;
34     novi->matematika = matematika;
35     novi->jezik = jezik;
36     novi->levo = NULL;
37     novi->desno = NULL;
38
39     /* Vraca se adresa kreiranog cvora */
40     return novi;
41 }
42
43 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo -
44    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
45    povratna vrednost je 1 */
46 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
47                    double uspeh, double matematika, double jezik)

```

```

49 {
50     /* Ako je stablo prazno */
51     if (*koren == NULL) {
52         /* Kreira se novi cvor */
53         Cvor *novi_cvor =
54             napravi_cvor(ime, prezime, uspeh, matematika, jezik);
55         /* Proverava se uspesnost kreiranja novog cvora */
56         if (novi_cvor == NULL) {
57             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
58              */
59             return 1;
60         }
61         /* Inace... */
62         /* Novi cvor se proglašava korenom stabla */
63         *koren = novi_cvor;
64
65         /* I vraca se indikator uspesnog dodavanja */
66         return 0;
67     }
68
69     /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
70      sortirano po ukupnom broju poena */
71     if (uspeh + matematika + jezik >
72         (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
73         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
74                               matematika, jezik);
75     else
76         return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
77                               matematika, jezik);
78 }
79
80 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
81 void oslobodi_stablo(Cvor ** koren)
82 {
83     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
84     if (*koren == NULL)
85         return;
86
87     /* Inace ... */
88     /* Oslobadja se memorija zauzeta levim podstablom */
89     oslobodi_stablo(&(*koren)->levo);
90
91     /* Oslobadja se memorija zauzeta desnim podstablom */
92     oslobodi_stablo(&(*koren)->desno);
93
94     /* Oslobadja se memorija zauzeta korenom */
95     free(*koren);
96
97     /* Stablo se proglašava praznim */
98     *koren = NULL;
99 }

```

```
101 /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
103 polozili jednaka 0 ispisuju se informacije o ucenicima koji nisu
    polozili prijemni, a ako je vrednost argumenta razlicita od nule,
105 ispisuju se informacije o ucenicima koji su polozili prijemni */
void stampaj(Cvor * koren, int polozili)
107 {
    /* Stablo je prazno - prekida se sa ispisom */
109     if (koren == NULL)
        return;

    /* Stampaju se informacije iz levog podstabla */
113     stampaj(koren->levo, polozili);

    /* Stampaju se informacije iz korenog cvora */
115     if (polozili && koren->matematika + koren->jezik >= 10)
117         printf("%s %s %.11f %.11f %.11f %.11f\n", koren->ime,
                koren->prezime, koren->uspeh, koren->matematika,
119                koren->jezik,
                koren->uspeh + koren->matematika + koren->jezik);
121     else if (!polozili && koren->matematika + koren->jezik < 10)
        printf("%s %s %.11f %.11f %.11f %.11f\n", koren->ime,
123                koren->prezime, koren->uspeh, koren->matematika,
                koren->jezik,
125                koren->uspeh + koren->matematika + koren->jezik);

    /* Stampaju se informacije iz desnog podstabla */
127     stampaj(koren->desno, polozili);
129 }

131 /* Funkcija koja odredjuje koliko studenata nije polozilo prijemni
    ispit */
133 int nisu_polozili(Cvor * koren)
    {
135         /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
        if (koren == NULL)
137             return 0;

        /* Pretraga se vrši i u levom i u desnom podstablu - ako uslov za
            polaganje nije ispunjen za koreni cvor, broj studenata se
141            uvecava za 1 */
        if (koren->matematika + koren->jezik < 10)
143            return 1 + nisu_polozili(koren->levo) +
                nisu_polozili(koren->desno);

145        return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
147    }

149 int main(int argc, char **argv)
    {
151     FILE *in;
```

```

153   Cvor *koren;
154   char ime[MAX], prezime[MAX];
155   double uspeh, matematika, jezik;
156
157   /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
158   in = fopen("prijemni.txt", "r");
159   if (in == NULL) {
160       fprintf(stderr,
161           "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
162       exit(EXIT_FAILURE);
163   }
164
165   /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
166      dodavanja */
167   koren = NULL;
168   while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
169               &matematika, &jezik) != EOF) {
170       if (dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik)
171           == 1) {
172           fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
173               prezime);
174           oslobodi_stablo(&koren);
175           return 0;
176       }
177   }
178
179   /* Zatvaranje datoteke */
180   fclose(in);
181
182   /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
183   stampaj(koren, 1);
184
185   /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
186      prijemni */
187   if (nisu_polozili(koren) != 0)
188       printf("-----\n");
189
190   /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
191   stampaj(koren, 0);
192
193   /* Oslobadja se memorija zauzeta stablom */
194   oslobodi_stablo(&koren);
195
196   return 0;
197 }

```

### Rešenje 4.18

```

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

```

```
3 #include <string.h>
5 #define MAX_NISKA 51
7 /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
   osobe, dan i mesec rođenja i redom pokazivace na levo i desno
   podstablo */
9 typedef struct cvor_stabla {
11     char ime[MAX_NISKA];
12     char prezime[MAX_NISKA];
13     int dan;
14     int mesec;
15     struct cvor_stabla *levo;
16     struct cvor_stabla *desno;
17 } Cvor;
19 /* Funkcija koja kreira novi cvor */
20 Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
21 {
22     /* Alocira se memorija */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
27     /* Inicijalizuju se polja strukture */
28     strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
30     novi->dan = dan;
31     novi->mesec = mesec;
32     novi->levo = NULL;
33     novi->desno = NULL;
35     /* Vraca se adresa novog cvora */
36     return novi;
37 }
39 /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
   uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
   dodavanje uspesno povratna vrednost je 0, u suprotnom povratna
   vrednost je 1 */
41 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
42                   int dan, int mesec)
43 {
44     /* Ako je stablo prazno */
45     if (*koren == NULL) {
46         /* Kreira se novi cvor */
47         Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
48         /* Proverava se uspesnost kreiranja novog cvora */
49         if (novi_cvor == NULL) {
50             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
51              */
52         }
53     }
```

```

55     return 1;
56 }
57 /* Inace... */
58 /* Novi cvor se proglašava korenom stabla */
59 *koren = novi_cvor;
60
61 /* I vraća se indikator uspešnog dodavanja */
62 return 0;
63 }
64
65 /* Stablo se uređjuje po mesecu, a zatim po danu u okviru istog
66    meseca */
67 if (mesec < (*koren)->mesec)
68     return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
69 else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
70     return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
71 else
72     return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec)
73     ;
74 }
75
76 /* Funkcija vrši pretragu stabla i vraća cvor sa traženim datumom */
77 Cvor *pretrazi(Cvor * koren, int dan, int mesec)
78 {
79     /* Stablo je prazno, obustavlja se pretraga */
80     if (koren == NULL)
81         return NULL;
82
83     /* Ako je traženi datum u korenu */
84     if (koren->dan == dan && koren->mesec == mesec)
85         return koren;
86
87     /* Ako je mesec traženog datuma manji od meseca sadržanog u korenu
88        ili ako su meseci isti ali je dan traženog datuma manji od
89        aktuelnog datuma, pretražuje se levo podstablo - pre toga se
90        svakako proverava da li leva grana postoji - ako ne postoji
91        treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
92        */
93     if (mesec < koren->mesec
94         || (mesec == koren->mesec && dan < koren->dan)) {
95         if (koren->levo == NULL)
96             return koren;
97         else
98             return pretrazi(koren->levo, dan, mesec);
99     }
100
101     /* Inace se nastavlja pretraga u desnom delu */
102     return pretrazi(koren->desno, dan, mesec);
103 }
104
105 /* Funkcija koja pronalazi najmanji datum u stablu */
106 Cvor *pronadji_najmanji_datum(Cvor * koren)

```

```

105 {
106     /* Stablo je prazno, obustavlja se pretraga */
107     if (koren == NULL)
108         return NULL;
109
110     /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
111        sadrzi najmanji datum */
112     if (koren->levo == NULL)
113         return koren;
114     else
115         /* Inace, trazimo manji datum u levom podstablu */
116         return pronadji_najmanji_datum(koren->levo);
117 }
118
119 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
120    */
121 void datum_u_nisku(int dan, int mesec, char datum[])
122 {
123     if (dan < 10) {
124         datum[0] = '0';
125         datum[1] = dan + '0';
126     } else {
127         datum[0] = dan / 10 + '0';
128         datum[1] = dan % 10 + '0';
129     }
130     datum[2] = '.';
131
132     if (mesec < 10) {
133         datum[3] = '0';
134         datum[4] = mesec + '0';
135     } else {
136         datum[3] = mesec / 10 + '0';
137         datum[4] = mesec % 10 + '0';
138     }
139     datum[5] = '.';
140     datum[6] = '\\0';
141 }
142
143 /* Funkcija koja oslobadja memoriju zauzetu stablom */
144 void oslobodi_stablo(Cvor ** adresa_korena)
145 {
146     /* Stablo je prazno */
147     if (*adresa_korena == NULL)
148         return;
149
150     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
151     if ((*adresa_korena)->levo)
152         oslobodi_stablo(&(*adresa_korena)->levo);
153
154     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
155     if ((*adresa_korena)->desno)
156         oslobodi_stablo(&(*adresa_korena)->desno);

```

```
157  /* Oslobadja se memorija zauzeta korenom */
    free(*adresa_korena);
159
    /* Proglasava se stablo praznim */
161  *adresa_korena = NULL;
    }
163
    int main(int argc, char **argv)
165  {
        FILE *in;
        Cvor *koren;
        Cvor *slavljenik;
169  char ime[MAX_NISKA], prezime[MAX_NISKA];
        int dan, mesec;
171  char datum[7];

173  /* Provera da li je zadato ime ulazne datoteke */
    if (argc < 2) {
175      /* Ako nije, ispisuje se poruka i prekida se sa izvršavanjem
          programa */
177      fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
      exit(EXIT_FAILURE);
179  }

181  /* Inace, priprema se datoteka za citanje */
    in = fopen(argv[1], "r");
183  if (in == NULL) {
        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
185             argv[1]);
        exit(EXIT_FAILURE);
187  }

189  /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja
     */
    koren = NULL;
191  while (fscanf
        (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
193      if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
          fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
195             prezime);
          oslobodi_stablo(&koren);
197          return 0;
        }
199

    /* Datoteka se zatvara */
201  fclose(in);

203  /* Omogucuje se pretraga podataka */
    while (1) {
205      /* Ucitava se novi datum */
```



```

207     printf("Unesite datum: ");
    if (scanf("%d.%d.", &dan, &mesece) == EOF)
209         break;

    /* Pretražuje se stablo */
    slavljenik = pretrazi(koren, dan, mesece);

213     /* Ispisuju se pronadjeni podaci */

215     /* Ako slavljenik nije pronadjen, to može znači da: */
    /* 1. drvo je prazno */
217     if (slavljenik == NULL && koren == NULL) {
219         printf("Nema podataka o ovom ni o sledećem rođendanu.\n");
        continue;
221     }
    /* 2. posle datuma koji je unesen, nema podataka u stablu - u
223     ovom slučaju se pretraga vrši počevši od naredne godine i
        ispisuje se najmanji datum */
225     if (slavljenik == NULL) {
        slavljenik = pronadji_najmanji_datum(koren);
227         datum_u_nisku(slavljenik->dan, slavljenik->mesece, datum);
        printf("Slavljenik: %s %s %s\n", slavljenik->ime,
229             slavljenik->prezime, datum);
        continue;
231     }

233     /* Ako je slavljenik pronadjen, razlikuju se slučajevi: */
    /* 1. Pronadjeni su tačni podaci */
235     if (slavljenik->dan == dan && slavljenik->mesece == mesece) {
        printf("Slavljenik: %s %s\n", slavljenik->ime,
237             slavljenik->prezime);
        continue;
239     }

241     /* 2. Pronadjeni su podaci o prvom sledećem rođendanu */
    datum_u_nisku(slavljenik->dan, slavljenik->mesece, datum);
243     printf("Slavljenik: %s %s %s\n", slavljenik->ime,
        slavljenik->prezime, datum);
245 }

247 /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);

249     return 0;
251 }

```

### Rešenje 4.19

```

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

```

```

4  /* Uključuje se biblioteka za rad sa stablima */
   #include "stabla.h"

6

8  /* Funkcija koja proverava da li su dva stabla koja sadrže cele
   brojewe identicna. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
10 int identitet(Cvor * koren1, Cvor * koren2)
   {
12     /* Ako su oba stabla prazna, jednaka su */
       if (koren1 == NULL && koren2 == NULL)
14         return 1;

16     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
       if (koren1 == NULL || koren2 == NULL)
18         return 0;

20     /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
       vrednosti, može se zaključiti da se razlikuju */
22     if (koren1->broj != koren2->broj)
       return 0;

24     /* Inace, proverava se da li vazi i jednakost levih i desnih
       podstabala */
26     return (identitet(koren1->levo, koren2->levo)
       && identitet(koren1->desno, koren2->desno));
   }

30
32 int main()
   {
       int broj;
34     Cvor *koren1, *koren2;

36     /* Učitavaju se elementi prvog stabla */
       koren1 = NULL;
38     printf("Prvo stablo: ");
       scanf("%d", &broj);
40     while (broj != 0) {
       if (dodaj_u_stablo(&koren1, broj) == 1) {
42         fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
         oslobodi_stablo(&koren1);
44         return 0;
       }
46     }
       scanf("%d", &broj);
   }

48
   /* Učitavaju se elementi drugog stabla */
50 koren2 = NULL;
   printf("Drugo stablo: ");
52 scanf("%d", &broj);
   while (broj != 0) {
54     if (dodaj_u_stablo(&koren2, broj) == 1) {
       fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);

```

```

56     oslobodi_stablo(&koren2);
57     return 0;
58 }
59 scanf("%d", &broj);
60 }

62 /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
63    njen rezultat. */
64 if (identitet(koren1, koren2))
65     printf("Stabla jesu identicna.\n");
66 else
67     printf("Stabla nisu identicna.\n");
68
69 /* Oslobadja se memorija zauzeta stablima */
70 oslobodi_stablo(&koren1);
71 oslobodi_stablo(&koren2);
72
73 return 0;
74 }

```

### Rešenje 4.20

```

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

4 /* Ukljucuje se biblioteka za rad sa stablima */
5 #include "stabla.h"

6
7 /* Funkcija kreira novo stablo identicno stablu koje je dato korenom.
8    Povratna vrednost funkcije je 0 ukoliko je kopiranje uspesno,
9    odnosno 1 ukoliko je doslo do greske */
10 int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
11 {
12     /* Izlaz iz rekurzije */
13     if (koren == NULL) {
14         *duplikat = NULL;
15         return 0;
16     }

18     /* Duplira se koren stabla i postavlja da bude koren novog stabla
19        */
20     *duplikat = napravi_cvor(koren->broj);
21     if (*duplikat == NULL) {
22         return 1;
23     }

24     /* Rekurzivno se duplirju levo i desno podstablo i njihove adrese
25        se cuvaju redom u pokazivacima na levo i desno podstablo korena
26        duplikata */
27     int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
28     int kopija_desno =

```

```
        kopiraj_stablo(koren->desno, &(*duplikat)->desno);
30
    /* Ako je uspesno duplirano i levo i desno podstablo */
32    if (kopiya_levo == 0 && kopiya_desno == 0)
        /* Uspesno je duplirano i celo stablo */
34        return 0;
    /* Inace, prijavljuje se da je doslo do greske */
36    return 1;
38}

/* Funkcija izracunava uniju dva skupa predstavljena stablima -
rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
42    uspesno, odnosno 1 ukoliko je doslo do greske */
44    int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
    {
46        /* Ako drugo stablo nije prazno */
        if (koren2 != NULL) {
48            /* 1. Dodaje se njegov koren u prvo stablo */
            if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
50                return 1;
            }

52            /* 2. Rekurzivno se racuna unija levog i desnog podstabla drugog
54            stabla sa prvim stablom */
            int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
56            int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);

58            /* Ako je unija podstabala uspesno kreirana */
            if (unija_levo == 0 && unija_desno == 0)
60                /* Uspesno je kreirana i unija stabala */
                return 0;

62            /* U suprotnom se prijavljuje da je doslo do greske */
64            return 1;
        }

66        /* Ako je drugo stablo prazno, nista se ne preduzima */
68        return 0;
    }

70    /* Funkcija izracunava presek dva skupa predstavljana stablima -
72    rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
    Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka
74    uspesno, odnosno 1 ukoliko je doslo do greske */
    int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
76    {
        /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
78        if (*adresa_korena1 == NULL)
            return 0;
80
```

```

82  /* Inace... */
/* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.
   iz levog i desnog podstabla prvog stabla brisu se svi oni
84  elementi koji ne postoje u drugom stablu */
int presek_levo = kreiraj_presek(&(*adresa_korena1)->levo, koren2);
86  int presek_desno =
    kreiraj_presek(&(*adresa_korena1)->desno, koren2);
88  if (presek_levo == 0 && presek_desno == 0) {
    /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on
90     uklanja iz prvog stabla */
    if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
92     obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

    /* Presek stabala je uspesno kreiran */
    return 0;
96  }
/* Inece, prijavljuje se da je doslo do greske */
98  return 1;
}

100 /* Funkcija izracunava razliku dva skupa predstavljana stablima -
102  rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
    Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike
104  uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
106 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
108  if (*adresa_korena1 == NULL)
    return 0;

110  /* Inace... */
112  /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
    tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
114  elementi koji postoje i u drugom stablu */
    int razlika_levo =
        kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
    int razlika_desno =
        kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
118  if (razlika_levo == 0 && razlika_desno == 0) {
    /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on
120     uklanja se iz prvog stabla */
    if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
122     obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

    /* Razlika stabala je uspesno kreirana */
126  return 0;
    }
128
    /* Inece, prijavljuje se da je doslo do greske */
130  return 1;
}
132

```

```
int main()
{
    Cvor *skup1;
    Cvor *skup2;
    Cvor *pomocni_skup = NULL;
    int n;

    /* Ucitavaju se elementi prvog skupa */
    skup1 = NULL;
    printf("Prvi skup: ");
    while (scanf("%d", &n) != EOF) {
        if (dodaj_u_stablo(&skup1, n) == 1) {
            fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
            oslobodi_stablo(&skup1);
            return 0;
        }
    }

    /* Ucitavaju se elementi drugog skupa */
    skup2 = NULL;
    printf("Drugi skup: ");
    while (scanf("%d", &n) != EOF) {
        if (dodaj_u_stablo(&skup2, n) == 1) {
            fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
            oslobodi_stablo(&skup2);
            return 0;
        }
    }

    /* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
       bi se isti mogao iskoristiti i za preostale operacije */
    if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
        oslobodi_stablo(&skup1);
        oslobodi_stablo(&pomocni_skup);
        return 0;
    }
    if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
        oslobodi_stablo(&pomocni_skup);
        oslobodi_stablo(&skup2);
        return 0;
    }
    printf("Unija: ");
    ispisi_stablo_infiksno(pomocni_skup);
    putchar('\n');

    /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
       operacije */
    oslobodi_stablo(&pomocni_skup);

    /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
       kako bi se isti mogao iskoristiti i za preostale operacije */
    if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
```

```

186     oslobodi_stablo(&skup1);
187     oslobodi_stablo(&pomocni_skup);
188     return 0;
189 }
190 if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
191     oslobodi_stablo(&pomocni_skup);
192     oslobodi_stablo(&skup2);
193     return 0;
194 }
195 printf("Presek: ");
196 ispisi_stablo_infiksno(pomocni_skup);
197 putchar('\n');
198
199 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
200    operacije */
201 oslobodi_stablo(&pomocni_skup);
202
203 /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
204    kako bi se isti mogao iskoristiti i za preostale operacije */
205 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
206     oslobodi_stablo(&skup1);
207     oslobodi_stablo(&pomocni_skup);
208     return 0;
209 }
210 if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
211     oslobodi_stablo(&pomocni_skup);
212     oslobodi_stablo(&skup2);
213     return 0;
214 }
215 printf("Razlika: ");
216 ispisi_stablo_infiksno(pomocni_skup);
217 putchar('\n');
218
219 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
220    operacije */
221 oslobodi_stablo(&pomocni_skup);
222
223 /* Oslobadja se memorija zauzeta polaznim skupovima */
224 oslobodi_stablo(&skup1);
225 oslobodi_stablo(&skup2);
226
227 return 0;
228 }

```

### Rešenje 4.21

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Ukljucuje se biblioteka za rad sa stablima */
5 #include "stabla.h"

```

```

7  #define MAX 50

9  /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
   cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11 su smestene u niz. */
   int kreiraj_niz(Cvor * koren, int a[])
13 {
   int r, s;

15   /* Stablo je prazno - u niz je smesteno 0 elemenata */
   if (koren == NULL)
17     return 0;

19   /* Dodaju se u niz elementi iz levog podstabla */
   r = kreiraj_niz(koren->levo, a);

21   /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
   u niz i na osnovu nje se moze odrediti indeks novog elementa */

23   /* Smesta se vrednost iz korena */
   a[r] = koren->broj;

25   /* Dodaju se elementi iz desnog podstabla */
   s = kreiraj_niz(koren->desno, a + r + 1);

27   /* Racuna se indeks na koji treba smestiti naredni element */
   return r + s + 1;

31 }

33 /* Funkcija sortira niz tako sto najpre elemente niza smesti u
   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
35 desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
   kreiran i sortiran, a 1 ukoliko je doslo do greske.

37   Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
   sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
39   sortiranje vrši u pomocnoj dinamičkoj strukturi, a ne razmenom
   elemenata niza. */

41   int sortiraj(int a[], int n)
   {
43     int i;
     Cvor *koren;

45     /* Kreira se stablo smestanjem elemenata iz niza u stablo */
     koren = NULL;
     for (i = 0; i < n; i++) {
47       if (dodaj_u_stablo(&koren, a[i]) == 1) {
49         oslobodi_stablo(&koren);
         return 1;
51       }
53     }
55   }
57 }

```



```

59  /* Infiksni obilaskom stabla elementi iz stabla se prepisuju u niz
    a */
61  kreiraj_niz(koren, a);

63  /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
    */
65  oslobodi_stablo(&koren);

67  /* Vraca se indikator uspesnog sortiranja */
69  return 0;
71 }

73 int main()
75 {
77     int a[MAX];
79     int n, i;

81     /* Ucitavaju se dimenzija i elementi niza */
83     printf("n: ");
85     scanf("%d", &n);
87     if (n < 0 || n > MAX) {
89         printf("Greska: pogresna dimenzija niza!\n");
91         return 0;
93     }

95     printf("a: ");
97     for (i = 0; i < n; i++)
99         scanf("%d", &a[i]);

101    /* Poziva se funkcija za sortiranje */
103    if (sortiraj(a, n) == 0) {
105        /* Ako je niz uspesno sortiran, ispisuje se rezultujuci niz */
107        for (i = 0; i < n; i++)
109            printf("%d ", a[i]);
111        printf("\n");
113    } else {
115        /* Inace, obavestava se korisnik da je doslo do greske */
117        printf("Greska: problem prilikom sortiranja niza!\n");
119    }

121    return 0;
123 }

```

### Rešenje 4.22

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

6

```

```
8  /* a) Funkcija koja izracunava broj cvorova stabla */
   int broj_cvorova(Cvor * koren)
   {
10  /* Ako je stablo prazno, broj cvorova je nula */
    if (koren == NULL)
12      return 0;

14  /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
    levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
16  zato sto treba racunati i koren */
    return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
18  }

20  /* b) Funkcija koja izracunava broj listova stabla */
   int broj_listova(Cvor * koren)
   {
22  /* Ako je stablo prazno, broj listova je nula */
    if (koren == NULL)
24      return 0;

26  /* Proverava se da li je tekuci cvor list */
    if (koren->levo == NULL && koren->desno == NULL)
28      /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
        uvecati broj listova za 1 */
        return 1;

30  /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
    */
    return broj_listova(koren->levo) + broj_listova(koren->desno);
32  }

34  /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
   void pozitivni_listovi(Cvor * koren)
   {
36  /* Slucaj kada je stablo prazno */
    if (koren == NULL)
38      return;

40  /* Ako je cvor list i sadrzi pozitivnu vrednost */
    if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
42      /* Stampa se */
      printf("%d ", koren->broj);

44  /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
    pozitivni_listovi(koren->levo);
    pozitivni_listovi(koren->desno);
46  }

50  /* d) Funkcija koja izracunava zbir cvorova stabla */
   int zbir_svih_cvorova(Cvor * koren)
   {
52  /* Ako je stablo prazno, zbir cvorova je 0 */
    return 0;
54  }
```

```

60     if (koren == NULL)
        return 0;

62     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
        elemenata u podstablama */
64     return koren->broj + zbir_svih_cvorova(koren->levo) +
        zbir_svih_cvorova(koren->desno);
66 }

68 /* e) Funkcija koja izracunava najveći element stabla */
Cvor *najveci_element(Cvor * koren)
70 {
    /* Ako je stablo prazno, obustavlja se pretraga */
72     if (koren == NULL)
        return NULL;
74
    /* Zbog prirode pretrazivackog stabla, vrednosti veće od korena se
76     nalaze u desnom podstablu */

    /* Ako desnog podstabla nema */
78     if (koren->desno == NULL)
80         /* Najveća vrednost je koren */
        return koren;
82
    /* Inace, najveća vrednost se traži desno */
84     return najveci_element(koren->desno);
}

86 /* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
{
    /* Dubina praznog stabla je 0 */
90     if (koren == NULL)
92         return 0;

    /* Izracunava se dubina levog podstabla */
94     int dubina_levo = dubina_stabla(koren->levo);
96
    /* Izracunava se dubina desnog podstabla */
98     int dubina_desno = dubina_stabla(koren->desno);

100    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
        jer se racuna i koren */
102    return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
104 }

106 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
108 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
{
    /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazene
110     nivoa */

```

```
112  /* Ako nema vise cvorova, nema spustanja niz stablo */
113  if (koren == NULL)
114      return 0;

116  /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
117     rekurzivnih poziva uvecati broj cvorova za 1 */
118  if (i == 0)
119      return 1;

120  /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
121     */
122  return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
123      + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
124  }

126  /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
127  void ispis_nivo(Cvor * koren, int i)
128  {
129      /* Ideja je slicna ideji iz prethodne funkcije */

130      /* Nema vise cvorova, nema spustanja kroz stablo */
131      if (koren == NULL)
132          return;

133      /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
134      if (i == 0) {
135          printf("%d ", koren->broj);
136          return;
137      }

138      /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
139         desnom podstablu */
140      ispis_nivo(koren->levo, i - 1);
141      ispis_nivo(koren->desno, i - 1);
142  }

144  /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
145     stabla */
146  Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
147  {
148      /* Ako je stablo prazno, obustavlja se pretraga */
149      if (koren == NULL)
150          return NULL;

151      /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
152      if (i == 0)
153          return koren;

154      /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
155      Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);

156      /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
```

```
162 Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);
164 /* Trazi se i vraca maksimum izracunatih vrednosti */
166 if (a == NULL && b == NULL)
168     return NULL;
168 if (a == NULL)
170     return b;
170 if (b == NULL)
172     return a;
172 return a->broj > b->broj ? a : b;
174 }
174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
176 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
178 {
178     /* Ako je stablo prazno, zbir je nula */
180     if (koren == NULL)
182         return 0;
182
184     /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
186     if (i == 0)
188         return koren->broj;
188
190     /* Inace, spustanje se nastavlja za jedan nivo nize i traze se sume
192     iz levog i desnog podstabla */
194     return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
196         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
198 }
198
200 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
202 manje ili jednake od date vrednosti x */
204 int zbir_manjih_od_x(Cvor * koren, int x)
206 {
206     /* Ako je stablo prazno, zbir je nula */
208     if (koren == NULL)
210         return 0;
210
212     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
214     prirode pretrazivackog stabla treba obici i levo i desno
216     podstablo */
218     if (koren->broj <= x)
220         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
222             zbir_manjih_od_x(koren->desno, x);
222
224     /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
226     medju njima jedino moze biti onih koje zadovoljavaju uslov */
228     return zbir_manjih_od_x(koren->levo, x);
230 }
230
232 int main(int argc, char **argv)
234 {
```

```

214  /* Analiza argumenata komandne linije */
215  if (argc != 3) {
216      fprintf(stderr,
217          "Greska! Program se poziva sa: ./a.out nivo
218          broj_za_pretragu\n");
219      exit(EXIT_FAILURE);
220  }
221  int i = atoi(argv[1]);
222  int x = atoi(argv[2]);
223
224  /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
225  */
226  Cvor *koren = NULL;
227  int broj;
228  while (scanf("%d", &broj) != EOF) {
229      if (dodaj_u_stablo(&koren, broj) == 1) {
230          fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
231          oslobodi_stablo(&koren);
232          return 0;
233      }
234  }
235
236  /* ispisuju se rezultati rada funkcija */
237  printf("Broj cvorova: %d\n", broj_cvorova(koren));
238  printf("Broj listova: %d\n", broj_listova(koren));
239  printf("Pozitivni listovi: ");
240  pozitivni_listovi(koren);
241  printf("\n");
242  printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
243  if (najveci_element(koren) == NULL)
244      printf("Najveci element: ne postoji\n");
245  else
246      printf("Najveci element: %d\n", najveci_element(koren)->broj);
247
248  printf("Dubina stabla: %d\n", dubina_stabla(koren));
249
250  printf("Broj cvorova na %d. nivou: %d\n", i,
251      broj_cvorova_na_itom_nivou(koren, i));
252  printf("Elementi na %d. nivou: ", i);
253  ispis_nivo(koren, i);
254  printf("\n");
255  if (najveci_element_na_itom_nivou(koren, i) == NULL)
256      printf("Nema elemenata na %d. nivou!\n", i);
257  else
258      printf("Maksimalni element na %d. nivou: %d\n", i,
259          najveci_element_na_itom_nivou(koren, i)->broj);
260
261  printf("Zbir elemenata na %d. nivou: %d\n", i,
262      zbir_cvorova_na_itom_nivou(koren, i));
263  printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
264      zbir_manjih_od_x(koren, x));

```

```

266  /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
268  return 0;
}

```

### Rešenje 4.23

```

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

4  /* Uključuje se biblioteka za rad sa stablima */
   #include "stabla.h"

6  /* Funkcija koja izracunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
   {
10     /* Dubina praznog stabla je 0 */
       if (koren == NULL)
12         return 0;

14     /* Izracunava se dubina levog podstabla */
       int dubina_levo = dubina_stabla(koren->levo);

16     /* Izracunava se dubina desnog podstabla */
       int dubina_desno = dubina_stabla(koren->desno);

18     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
       jer se racuna i koren */
20     return dubina_levo >
           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
22 }
24

26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
   void ispisi_nivo(Cvor * koren, int i)
   {
28     /* Ideja je slicna ideji iz prethodne funkcije */
       /* Nema vise cvorova, nema spustanja niz stablo */
       if (koren == NULL)
30         return;

32     /* Ako se stiglo do trazene nivoa - ispisuje se vrednost */
       if (i == 0) {
34         printf("%d ", koren->broj);
           return;
36     }

38     /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
       podstablu */
       ispisi_nivo(koren->levo, i - 1);
40     ispisi_nivo(koren->desno, i - 1);
42 }

```

```

44  /* Funkcija koja ispisuje stablo po nivoima */
46  void ispisi_stablo_po_nivoima(Cvor * koren)
47  {
48      int i;
49
50      /* Prvo se izracunava dubina stabla */
51      int dubina;
52      dubina = dubina_stabla(koren);
53
54      /* Ispisuje se nivo po nivo stabla */
55      for (i = 0; i < dubina; i++) {
56          printf("%d. nivo: ", i);
57          ispisi_nivo(koren, i);
58          printf("\n");
59      }
60  }
61
62  int main(int argc, char **argv)
63  {
64      Cvor *koren;
65      int broj;
66
67      /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
68       uspesnosti dodavanja */
69      koren = NULL;
70      while (scanf("%d", &broj) != EOF) {
71          if (dodaj_u_stablo(&koren, broj) == 1) {
72              fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
73              oslobodi_stablo(&koren);
74              return 0;
75          }
76      }
77
78      /* Ispisuje se stablo po nivoima */
79      ispisi_stablo_po_nivoima(koren);
80
81      /* Oslobadja se memorija zauzeta stablom */
82      oslobodi_stablo(&koren);
83
84      return 0;
85  }

```

### Rešenje 4.25

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Ukljuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

```



```

7  /* Funkcija koja izracunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
9  {
   /* Dubina praznog stabla je 0 */
11  if (koren == NULL)
       return 0;

13
   /* Izracunava se dubina levog podstabla */
15  int dubina_levo = dubina_stabla(koren->levo);

17  /* Izracunava se dubina desnog podstabla */
   int dubina_desno = dubina_stabla(koren->desno);

19
   /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
21  jer se racuna i koren */
   return dubina_levo >
23         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
   }

25
   /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
27  stablo */
   int avl(Cvor * koren)
29  {
       int dubina_levo, dubina_desno;

31
       /* Ako je stablo prazno, zaustavlja se brojanje */
33  if (koren == NULL) {
           return 0;
35  }

37  /* Izracunava se dubina levog podstabla korena */
       dubina_levo = dubina_stabla(koren->levo);

39
       /* Izracunava se dubina desnog podstabla korena */
41  dubina_desno = dubina_stabla(koren->desno);

43
       /* Ako je uslov za AVL stablo ispunjen */
       if (abs(dubina_desno - dubina_levo) <= 1) {
45           /* Racuna se broj AVL cvorova u levom i desnom podstablu i
               uvecava za jedan iz razloga sto koren ispunjava uslov */
47           return 1 + avl(koren->levo) + avl(koren->desno);
       } else {
49           /* Inace, racuna se samo broj AVL cvorova u podstablama */
           return avl(koren->levo) + avl(koren->desno);
51       }
   }

53
   int main(int argc, char **argv)
55  {
       Cvor *koren;
57       int broj;

```

```

59  /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
    uspesnosti dodavanja */
61  koren = NULL;
    while (scanf("%d", &broj) != EOF) {
63      if (dodaj_u_stablo(&koren, broj) == 1) {
          fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
65          oslobodi_stablo(&koren);
          return 0;
67      }
    }
69
    /* Racuna se i ispisuje broj AVL cvorova */
71    printf("%d\n", avl(koren));

73    /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
75
    return 0;
77 }

```

### Rešenje 4.26

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
    #include "stabla.h"
6
8  /* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
    brojeva hip. Ideja koja ce biti implementirana u osnovi ima
    pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
10   desnog podstabla - ako je vrednost u korenu veka od izracunatih
    vrednosti, uoceni fragment stabla zadovoljava uslov za hip. Zato
12   ce funkcija vratiti maksimalne vrednosti iz uocenog podstabala ili
    vrednost -1 ukoliko se zakljuci da stablo nije hip. */
14  int heap(Cvor * koren)
    {
16      int max_levo, max_desno;

18      /* Prazno sablo je hip - kao rezultat se vraca 0 kao najmanji
        pozitivan broj */
20      if (koren == NULL) {
          return 0;
22      }
        /* Ukoliko je stablo list... */
24      if (koren->levo == NULL && koren->desno == NULL) {
          /* Vraca se njegova vrednost */
26          return koren->broj;
        }
28      /* Inace... */

```

```
30  /* Proverava se svojstvo za levo podstablo */
    max_levo = heap(koren->levo);
32
    /* Proverava se svojstvo za desno podstablo */
34  max_desno = heap(koren->desno);

36  /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
    ni celo stablo */
38  if (max_levo == -1 || max_desno == -1) {
    return -1;
40  }

42  /* U suprotnom proverava se da li svojstvo vazi za uoceni cvor */
    if (koren->broj > max_levo && koren->broj > max_desno) {
44      /* Ako vazi, vraca se vrednost korena */
      return koren->broj;
46  }

48  /* U suprotnom zakljucuje se da stablo nije hip */
    return -1;
50 }

52 int main(int argc, char **argv)
{
54     Cvor *koren;
    int hip_indikator;

56
    /* Kreira se stablo prema zadatoj slici */
58     koren = NULL;
    koren = napravi_cvor(100);
    koren->levo = napravi_cvor(19);
60     koren->levo->levo = napravi_cvor(17);
    koren->levo->levo->levo = napravi_cvor(2);
62     koren->levo->levo->desno = napravi_cvor(7);
    koren->levo->desno = napravi_cvor(3);
    koren->desno = napravi_cvor(36);
64     koren->desno->levo = napravi_cvor(25);
    koren->desno->desno = napravi_cvor(1);
66
68
    /* Poziva se funkcija kojom se proverava da li je stablo hip */
70     hip_indikator = heap(koren);

72
    /* Ispisuje se rezultat */
    if (hip_indikator == -1) {
74         printf("Zadato stablo nije hip!\n");
    } else {
76         printf("Zadato stablo je hip!\n");
    }

78
    /* Oslobadja se memorija zauzeta stablom */
80     oslobodi_stablo(&koren);
```

```
82 |   return 0;  
   | }
```

## 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 za grešku ispisati vrednost -1 i prekinuti izvršavanje programa.

#### *Test 1*

```
Poziv: ./a.out ulaz.txt
```

```
ULAZ.TXT
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
```

```
IzLAZ:
Ispit
Matematika
Programiranje
```

#### *Test 2*

```
Poziv: ./a.out ulaz.txt
```

```
ULAZ.TXT
2
maksimalano
poena
```

```
IzLAZ:
```

Test 3

```
POZIV: ./a.out ulaz.txt
DATOTEKA ULAZ.TXT NE POSTOJI
IZLAZ:
-1
```

Test 4

```
POZIV: ./a.out
IZLAZ:
-1
```

[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  $-1$ .

Test 1

```
ULAZ:
2 8 10 3 6 14 13 7 4 0
IZLAZ:
20
```

Test 2

```
ULAZ:
0 50 14 5 2 4 56 8 52 7 1 0
IZLAZ:
50
```

[Rešenje 5.2]

**Zadatak 5.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost  $-1$  na standardni izlaz za greške.

<p><i>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 2</i></p> <pre> ULAZ: 2 3 0 0 -5 1 2 -4 IZLAZ: 2         </pre>	<p><i>Test 3</i></p> <pre> ULAZ: -2 IZLAZ: -1         </pre>
--	--	--

[Rešenje 5.3]

## 5.2 Programiranje 2, praktični deo ispita, jul 2015.

**Zadatak 5.4** Napisati program koji kao prvi arugment 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 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 2</i></p> <pre> POZIV: ./a.out IZLAZ: -1         </pre>
<p><i>Test 3</i></p> <pre> POZIV: ./a.out ulaz.txt foo DATOTEKA ULAZ.TXT NE POSTOJI IZLAZ: -1         </pre>	<p><i>Test 4</i></p> <pre> POZIV: ./a.out ulaz.txt . DATOTEKA ULAZ.TXT JE PRAZNA IZLAZ: 0         </pre>

[Rešenje 5.4]

**Zadatak 5.5** Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitava trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

```
Test 1
TROUGLOVI.TXT
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 0 0 0 1
IZLAZ:
2 0 2 2 -1 -1
-2 0 0 0 0 1
0 0 0 1.2 1 0
0.3 0.3 0.5 0.5 0.9 1
```

```
Test 2
TROUGLOVI.TXT
3
1.2 3.2 1.1 4.3
IZLAZ:
-1
```

```
Test 3
DATOTEKA TROUGLOVI.TXT NE POSTOJI
IZLAZ:
-1
```

```
Test 4
TROUGLOVI.TXT
0
IZLAZ:
```

[Rešenje 5.5]

**Zadatak 5.6** Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeva. Napisati funkciju  
**int prebrojN(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.

```
Test 1
ULAZ:
1 5 3 6 1 4 7 9
IZLAZ:
1
```

```
Test 2
ULAZ:
2 5 3 6 1 0 4 7 9
IZLAZ:
2
```

```
Test 3
ULAZ:
0 4 2 5
IZLAZ:
0
```



<p><i>Test 4</i></p> <pre>    ULAZ:      3    IZLAZ:      0           </pre>	<p><i>Test 5</i></p> <pre>    ULAZ:     -1 4 5 1 7    IZLAZ:      0           </pre>
--	--

[Rešenje 5.6]

### 5.3 Programiranje 2, praktični deo ispita, septembar 2015.

**Zadatak 5.7** Sa standardnog ulaza se učitavaju neoznačeni celi brojevi x i n. Na standardni izlaz ispisati neoznačen ceo broj koji se dobija od broja x kada se njegov binarni zapis rotira za n mesta udesno (na primer, ako je binarni zapis broja x jednak 000000000000000000000000000000001111, i ako je n=1 tada na standardni izlaz treba ispisati neožnačen broj čiji je binarni zapis jednak 1000000000000000000000000000000011).

Test 1	Test 2	Test 3
ULAZ: 6 1	ULAZ: 15 3	ULAZ: 31 100
IZLAZ: 3	IZLAZ: 3758096385	IZLAZ: 4026531841
Test 4	Test 5	
ULAZ: 4 0	ULAZ: 0 5	
IZLAZ: 4	IZLAZ: 0	

[Rešenje 5.7]

**Zadatak 5.8** Napisati funkciju `int dopuni_listu(Cvor ** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Povratna vrednost funkcije treba da bude 1 ukoliko je došlo greške pri alokaciji memorije, inače 0. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

Test 1

```

|| ULAZ:
| 1 2 3 4 5
|| IZLAZ:
| 1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00

```

Test 2

```

|| ULAZ:
| 12
|| IZLAZ:
| 12.00

```

Test 3

```

|| ULAZ:
| prazna lista
|| IZLAZ:

```

Test 4

```

|| ULAZ:
| 13.3 15.8
|| IZLAZ:
| 13.30 14.55

```

[Rešenje 5.8]

**Zadatak 5.9** Sa standardnog ulaza se učitava dimenzija  $n$  kvadratne celobrojne matrice  $A$  ( $n > 0$ ), a zatim i elementi matrice  $A$ . Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje je suma brojeva u svakom redu i svakoj koloni jednaka). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati  $-$ . Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati  $-1$  na standardni izlaz za grešku. NAPOMENA: *Rešenje koristi biblioteku za rad sa matricama iz zadatka 2.19.*

Test 1

```

|| ULAZ:
| 4
| 1 2 3 4
| 2 1 4 3
| 3 4 2 1
| 4 3 1 2
|| IZLAZ:
| 10

```

Test 2

```

|| ULAZ:
| 3
| 1 1 1
| 1 1 1
| 1 1 1
|| IZLAZ:
| 3

```

Test 3

```

|| ULAZ:
| 2
| 1 1
| 2 2
|| IZLAZ:
| -

```

Test 4

```

|| ULAZ:
| 2
| 1 2
| 1 2
|| IZLAZ:
| -

```

Test 5

```

|| ULAZ:
| 1
| 5
|| IZLAZ:
| 5

```

Test 6

```

|| ULAZ:
| 0
|| IZLAZ:
| -1

```

[Rešenje 5.9]

## 5.4 Rešenja

### Rešenje 5.1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #define MAX 50

6  /* Funkcija vrši dinamičku alokaciju memorije potrebne n linija
   tj. n niski od kojih nijedna nije duža od MAX karaktera. */
8  char **alociranje_memorije(int n)
   {
10     char **linije = NULL;
11     int i, j;
12     /* Alocira se prostor za niz vrsti matrice */
13     linije = (char **) malloc(n * sizeof(char *));
14     /* U slučaju
15     neuspješnog otvaranja ispisuje se -1 na stderr i program završava.
16     */
17     if (linije == NULL)
18         return NULL;
19     /* Alocira se prostor za svaku vrstu matrice */
20     for (i = 0; i < n; i++) {
21         linije[i] = malloc(MAX * sizeof(char));
22     /* Ako alokacija nije prošla uspešno, oslobadjaju se svi
23        prethodno alocirani resursi, i povratna vrednost je NULL */
24         if (linije[i] == NULL) {
25             for (j = 0; j < i; j++) {
26                 free(linije[j]);
27             }
28             free(linije);
29             return NULL;
30         }
31     }
32     return linije;
33 }

34 char **oslobadjanje_memorije(char **linije, int n)
35 {
36     int i;
37     /* Oslobadja se prostor rezervisan za svaku vrstu */
38     for (i = 0; i < n; i++) {
39         free(linije[i]);
40     }
41     /* Oslobadja se memorija za niz pokazivaca na vrste */
42     free(linije);

43     /* Matrica postaje prazna, tj. nealocirana */
44     return NULL;
45 }

```

```

48 }
49
50 int main(int argc, char *argv[])
51 {
52     FILE * ulaz;
53     char **linije;
54     int i, j, n;
55
56     /* Proverava argumenata komandne linije. */
57     if (argc != 2) {
58         fprintf(stderr, "-1\n");
59         exit(EXIT_FAILURE);
60     }
61
62     /* Otvaranje datoteke cije ime je navedeno kao argument komandne
63     linije neposredno nakon imena programa koji se poziva. U slucaju
64     neuspesnog otvaranja ispisuje se -1 na stderr i program zavrшава.
65     */
66     ulaz = fopen(argv[1], "r");
67     if (ulaz == NULL) {
68         fprintf(stderr, "-1\n");
69         exit(EXIT_FAILURE);
70     }
71     /* Ucitavanje broja linija. */
72     fscanf(ulaz, "%d", &n);
73
74     /* Alociranje memorije na osnovu ucitanog broja linija. */
75     linije = alociranje_memorije(n);
76
77     /*U slucaju neuspesne alokacije ispisuje se -1 na stderr i program
78     zavrшава. */
79     if (ulaz == NULL) {
80         fprintf(stderr, "-1\n");
81         exit(EXIT_FAILURE);
82     }
83
84     /* Ucitavanje svih n linija iz datoteke. */
85     for (i = 0; i < n; i++) {
86         fscanf(ulaz, "%s", linije[i]);
87     }
88
89     /* Ispisivanje u odgovarajucem poretku ucitane linije koje
90     zadovoljavaju kriterijum. */
91     for (i = n - 1; i >= 0; i--) {
92         if (isupper(linije[i][0])) {
93             printf("%s\n", linije[i]);
94         }
95     }
96     /* Oslobadjanje memorije koja je dinamicki alocirana. */
97     linije = oslobadjanje_memorije(linije, n);
98
99     /* Zatvaranje datoteku. */

```

```

100     fclose(ulaz);
        exit(EXIT_SUCCESS);
    }

```

## Rešenje 5.2

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Datoteka 5.1: *main.c*

```

#include <stdio.h>
#include <stdlib.h>
#include "stabla.h"

int sumirajN(Cvor * koren, int n)
{
    /* Ako je stablo prazno, suma je nula */
    if (koren == NULL)
        return 0;
    /* Inace ... */
    /* Ako je n jednako nula, vraca se broj iz korena */
    if (n == 0)
        return koren->broj;
    /* Inace, izracunava se suma na (n-1)-om nivou u levom podstablu,
       kao i suma na (n-1)-om nivou u desnom podstablu i vraca se zbir
       te dve izracunate vrednosti jer predstavlja zbir svih cvorova na
       n-tom nivou u pocetnom stablu */
    return sumirajN(koren->levo, n - 1) + sumirajN(koren->desno,
                                                    n - 1);
}

int main()
{
    Cvor * koren = NULL;
    int n;
    int nivo;

    /* Ucitava se vrednost nivoa */
    scanf("%d", &nivo);
    while (1) {
        scanf("%d", &n);
        /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
        if (n == 0)
            break;
        /* Ako nije, dodaje se procitani broj u stablo. */
        if (dodaj_u_stablo(&koren, n) == 1) {
            fprintf(stderr, "-1\n", n);
            oslobodi_stablo(&koren);
            exit(EXIT_FAILURE);
        }
    }
}

```

```

40     }
41 }
42
43 /* Ispisuje se rezultat rada trazene funkcije */
44 printf("%d\n", sumirajN(koren, nivo));
45
46 /* Oslobadja se memorija */
47 oslobodi_stablo(&koren);
48
49 return 0;
50 }

```

### Rešenje 5.3

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX 50
4
5 /* Funkcija ucitava elemente matrice sa standardnog ulaza */
6 void ucitaj_matricu(int m[][MAX], int v, int k)
7 {
8     int i, j;
9     for (i = 0; i < v; i++) {
10         for (j = 0; j < k; j++) {
11             scanf("%d", &m[i][j]);
12         }
13     }
14 }
15
16 /* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
17    koja ima v vrsta */
18 int broj_negativnih_u_koloni(int m[][MAX], int v, int k)
19 {
20     int broj_negativnih = 0;
21     int i;
22     for(i=0; i<v; i++){
23         if(m[i][k]<0)
24             broj_negativnih++;
25     }
26     return broj_negativnih;
27 }
28
29 int max_indeks(int m[][MAX], int v, int k)
30 {
31     int i, j;
32     int broj_negativnih;
33     /* Inicijalizujemo na nulu indeks kolone sa maksimalnim brojem
34        negativnih (max_indeks_kolone), kao i maksimalni broj negativnih
35        elemenata u nekoj koloni (max_broj_negativnih) */
36     int max_indeks_kolone = 0;
37     int max_broj_negativnih = 0;

```

```

38     for (j = 0; j < k; j++) {
39         /* Racunamo broj negativnih u j-oj koloni */
40         broj_negativnih = broj_negativnih_u_koloni(m, v, j);
41         /* Ukoliko broj negativnih u j-toj koloni veci od trenutnog
42            maksimalnog, azuriramo trenutni maksimalni broj negativnih
43            kao i indeks kolone sa maksimalnim brojem negativnih */
44         if(max_broj_negativnih < broj_negativnih ){
45             max_indeks_kolone = j;
46             max_broj_negativnih = broj_negativnih;
47         }
48     }
49     return max_indeks_kolone;
50 }
51
52 int main()
53 {
54     int m[MAX][MAX];
55     int v, k;
56
57     /* Ucitavanje dimenzije matrice */
58     scanf("%d", &v);
59     if (v < 0 || v > MAX) {
60         fprintf(stderr, "-1\n");
61         exit(EXIT_FAILURE);
62     }
63     scanf("%d", &k);
64     if (k < 0 || k > MAX) {
65         fprintf(stderr, "-1\n");
66         exit(EXIT_FAILURE);
67     }
68     /* Ucitavanje elemenata matrice */
69     ucitaj_matricu(m, v, k);
70
71     /* Pronalazenje kolone koja sadrzi najveći broj negativnih
72        elemenata i ispisivanje trazenog rezultata */
73     printf("%d\n", max_indeks(m, v, k));
74     exit(EXIT_SUCCESS);
75 }

```

### Rešenje 5.4

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #define MAX 128

6 int main(int argc, char **argv)
{
8     FILE * f;
    int brojac = 0;

```

```

10 char linija[MAX], *p;

12 /* Provera da li je broj argumenata komandne linije 3 */
13 if (argc != 3) {
14     fprintf(stderr, "-1\n");
15     exit(EXIT_FAILURE);
16 }
17 /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18 linije */
19 if ((f = fopen(argv[1], "r")) == NULL) {
20     fprintf(stderr, "-1\n");
21     exit(EXIT_FAILURE);
22 }
23 /* Ucitavanje iz otvorene datoteke - liniju po liniju */
24 while (fgets(linija, MAX, f) != NULL) {
25     p = linija;
26     while (1) {
27         p = strstr(p, argv[2]);
28
29         /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
30         if (p == NULL)
31             break;
32         /* Inace se uvecava brojac */
33         brojac++;
34         /* p se pomera da bi se u sledecoj iteraciji posmatra ostatak
35 linije nakon uocene podniske */
36         p = p + strlen(argv[2]);
37     }
38 }
39 /* Zatvaranje datoteke */
40 fclose(f);
41 /* Ispisivanje vrednosti brojaca */
42 printf("%d\n", brojac);
43 return 0;
44 }

```

### Rešenje 5.5

```

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

5 /* Struktura trougao */
6 typedef struct _trougao {
7     double xa, ya, xb, yb, xc, yc;
8 } trougao;

9
10 /* Funkcija racuna duzinu duzi */
11 double duzina(double x1, double y1, double x2, double y2)
12 {
13     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));

```



```

15 }
16
17 /* Funkcija racuna povrsinu trougla koristeći Heronov obrazac */
18 double povrsina(trougao t)
19 {
20     /* Racunanje duzina stranica trougla */
21     double a = duzina(t.xb, t.yb, t.xc, t.yc);
22     double b = duzina(t.xa, t.ya, t.xc, t.yc);
23     double c = duzina(t.xa, t.ya, t.xb, t.yb);
24     /* Poluobim */
25     double s = (a + b + c) / 2;
26     /* Primena Heronovog obrasca */
27     return sqrt(s * (s - a) * (s - b) * (s - c));
28 }
29
30 /* Funkcija racuna poredi dva trougla tako da ukoliko je povrsina
31    trougla koji je prvi argument funkcije manja od povrsine trougla
32    koji je drugi element funkcije funkcija vraca 1, ukoliko je veca
33    -1, a ukoliko su povrsine dva trougla jednake vraca nulu. Dakle,
34    funkcija je napisana tako da se moze proslediti funkciji qsort da
35    se niz trouglova sortira po povrsini opadajuće. */
36 int poredi(const void *a, const void *b)
37 {
38     trougao x = *(trougao *) a;
39     trougao y = *(trougao *) b;
40     double xp = povrsina(x);
41     double yp = povrsina(y);
42     if (xp < yp)
43         return 1;
44     if (xp > yp)
45         return -1;
46     return 0;
47 }
48
49 int main()
50 {
51     FILE * f;
52     int n, i;
53     trougao * niz;
54
55     /* Otvaranje datoteke ciji je naziv trouglovi.txt */
56     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
57         fprintf(stderr, "-1\n");
58         exit(EXIT_FAILURE);
59     }
60
61     /* Ucitavanje podataka o broju trouglova iz datoteke */
62     if (fscanf(f, "%d", &n) != 1) {
63         fprintf(stderr, "-1\n");
64         exit(EXIT_FAILURE);
65     }

```

```

67  /* Dinamicka alokacija memotije za niz trouglova duzine n */
    if ((niz = malloc(n * sizeof(trougao))) == NULL) {
        fprintf(stderr, "-1\n");
69      exit(EXIT_FAILURE);
    }

71  /* Ucitavanje podataka u niz iz otvorene datoteke */
73  for (i = 0; i < n; i++) {
        if (fscanf(f, "%lf%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
75            &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6) {
            fprintf(stderr, "-1\n");
77            exit(EXIT_FAILURE);
        }
79    }

81  /* Pozivanje funkcije qsort da sortira niz na osnovu funkcije
        poredi */
83  qsort(niz, n, sizeof(trougao), &poredi);

85  /* Ispisivanje sortiranog niza na standardni izlaz */
    for (i = 0; i < n; i++)
87      printf("%g %g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
            niz[i].yb, niz[i].xc, niz[i].yc);

89  /* Oslobadjanje dinamicki alocirane memorije */
91  free(niz);

93  /* Zatvranje datoteke */
    fclose(f);
95  exit(EXIT_SUCCESS);
}

```

### Rešenje 5.6

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Datoteka 5.2: *main.c*

```

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

4  /* Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
        stablo. Funkcija vraca 1 u slucaju neuspesnog dodavanja elementa u
6        stablo, a inace 0. */
8  int ucitaj_stablo(Cvor ** koren)
    {
10     *koren = NULL;
        int x;

```

```

12  /* Sve dok ima brojeva na standardnom ulazu, ucitani brojevi se
13     dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
14     je i povratna vrednost iz funkcije ucitaj_stablo 1. */
15  while (scanf("%d", &x) == 1)
16      if (dodaj_u_stablo(koren, x) == 1)
17          return 1;
18  return 0;
19  }
20
21  /* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
22  int prebrojN(Cvor * koren, int n)
23  {
24      /* Ukoliko je stablo prazno, rezultat je nula. Takodje, ako je n
25         negativan broj, na tom nivou nema cvorova (rezultat je nula). */
26      if (koren == NULL || n < 0)
27          return 0;
28      /* Ukoliko je n = 0, na tom nivou je samo koren. Ukoliko ima jednog
29         potomka funkcija vraca 1, inace 0 */
30      if (n == 0) {
31          if (koren->levo == NULL && koren->desno != NULL)
32              return 1;
33          if (koren->levo != NULL && koren->desno == NULL)
34              return 1;
35          return 0;
36      }
37      /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
38         (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om nivou
39         levog podstabla*/
40      return prebrojN(koren->levo, n - 1) + prebrojN(koren->desno, n - 1)
41          ;
42  }
43
44  int main()
45  {
46      Cvor * koren;
47      int n;
48      scanf("%d", &n);
49
50      /* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
51         oslobodja se alocirana memorija i izlazi se iz programa. */
52      if (ucitaj_stablo(&koren) == 1) {
53          fprintf(stderr, "-1\n");
54          oslobodi_stablo(&koren);
55          exit(EXIT_FAILURE);
56      }
57
58      /* Ispisivanje rezultata */
59      printf("%d\n", prebrojN(koren, n));
60
61      /* Oslobadjanje dinamiccki alociranog stabla */
62      oslobodi_stablo(&koren);

```

```

64     return 0;
    }

```

### Rešenje 5.7

```

2  #include <stdio.h>
3
4  /* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
   * zapis argumenta x rotira za n mesta udesno */
5  unsigned int Rotiraj(unsigned int x, unsigned int n)
6  {
7      int i;
8      unsigned int maska = 1;
9      /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
   * izgledala: 000...00001111 */
10     for (i = 1; i < n; i++)
11         maska = (maska << 1) | 1;
12     /* Kada se x poremeri za n mesta udesno x >> n, poslednjih n bitova
   * binarne reprezentacije broja x ce "ispasti". Za rotaciju je
   * potrebno da se tih n bitova postavi na pocetak broja. Kreirana
   * maska omogucava da se tih n bitova izdvoji sa (maska & x), a
   * zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta ulevo
   * tih n bitova postavlja na pocetak. Primenom logicke disjunkcije
   * dobija se rotirani broj. */
13     return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
14 }
15
16 int main()
17 {
18     unsigned int x, n;
19
20     /*Ucitavanje brojeva sa standardnog ulaza */
21     scanf("%u%u", &x, &n);
22
23     /* Ispisivanje rezultata */
24     printf("%u\n", Rotiraj(x, n));
25     return 0;
26 }

```

### Rešenje 5.8

Datoteka 5.3: *liste.h*

```

1  #ifndef _LISTE_H_
2  #define _LISTE_H_ 1
3
4  /* Struktura koja predstavlja cvor liste */
5  typedef struct cvor {

```

```

6     double vrednost;
7     struct cvor *sledeci;
8 } Cvor;

10 /* Pomocna funkcija koja kreira cvor. */
11 Cvor * napravi_cvor(double broj);
12
13 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
14    ciji se pocetni cvor nalazi na adresi adresa_glave. */
15 void oslobodi_listu(Cvor ** adresa_glave);
16
17 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
18    ili NULL kao je lista prazna */
19 Cvor * pronadji_poslednji(Cvor * glava);
20
21 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
22    greske pri alokaciji memorije, inace vraca 0. */
23 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);
24
25 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
26    */
27 void ispisi_listu(Cvor * glava);
28
29 #endif

```

Datoteka 5.4: *liste.c*

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "liste.h"

4
5  /* Pomocna funkcija koja kreira cvor. */
6  Cvor * napravi_cvor(double broj)
7  {
8      Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
9      if (novi == NULL)
10         return NULL;
11     /* inicijalizacija polja u novom cvoru */
12     novi->vrednost = broj;
13     novi->sledeci = NULL;
14     return novi;
15 }

16
17 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
18    ciji se pocetni cvor nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave)
20 {
21     Cvor * pomocni = NULL;
22     while (*adresa_glave != NULL) {
23         pomocni = (*adresa_glave)->sledeci;

```

```

    free(*adresa_glave);
25     *adresa_glave = pomocni;
    }
27 }

29 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL kao je lista prazna */
31 Cvor * pronadji_poslednji(Cvor * glava)
    {
33     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom
        slucaju funkcija vraca NULL. */
35     if (glava == NULL)
        return NULL;
37     while (glava->sledeci != NULL)
        glava = glava->sledeci;
39     return glava;
    }
41

43 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
    greske pri alokaciji memorije, inace vraca 0. */
    int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
45 {
    Cvor * novi = napravi_cvor(broj);
47     if (novi == NULL)
        return 1;
49     if (*adresa_glave == NULL) {
        *adresa_glave = novi;
51     }
    Cvor * poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
55 }

57 /* Funkcija prikazuje elemente liste pocev od glave ka kraju
    liste. */
59 void ispisi_listu(Cvor * glava)
    {
61     for (; glava != NULL; glava = glava->sledeci)
        printf("%.2lf ", glava->vrednost);
63     putchar('\n');
    }

```

Datoteka 5.5: *main.c*

```

#include <stdio.h>
2  #include <stdlib.h>
    #include "liste.h"
4
    /* Funkcija umece novi cvor iza tekuceg u listi */
6  void dodaj_iza(Cvor * tekuci , Cvor * novi)
    {

```

```
8  /* Novi cvor se dodaje iza tekućeg cvora. */
   novi ->sledeci = tekuci ->sledeci;
10  tekuci ->sledeci = novi;
   }
12
14  /* Funkcija koja dopunjuje listu na način opisan u tekstu zadatka.
   Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace vraca
16  0. */
   int dopuni_listu(Cvor ** adresa_glave)
18  {
       Cvor * tekuci;
20  Cvor * novi;
       double aritmeticka_sredina;
22  /* U slučaju prazne ili jednoclane liste, funkcija vraca 1 */
       if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
24  return 1;
       /* Promenljiva tekuci se inicijalizuje da pokazuje na početni
26  cvor */
       tekuci = *adresa_glave;
28  /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
       vrednosti u susednim cvorovima i kreira cvor sa tom vrednoscu.
30  U slučaju neupele alokacije novog cvora, funkcija vraca 1.
       Inace, novi cvor se umeće između dva cvora za koje racunata
32  aritmeticka sredina */
       while (tekuci->sledeci != NULL) {
34  aritmeticka_sredina =
           ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
36  novi = napravi_cvor(aritmeticka_sredina);
           if (novi == NULL)
38  return 1;
       /* Poziva se funkcija koja umeće novi cvor iza tekućeg cvora */
40  dodaj_iza(tekuci, novi);
       /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
42  cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz
       polazne liste */
44  tekuci = tekuci->sledeci;
       tekuci = tekuci->sledeci;
46  }
       return 0;
48  }

50  int main()
   {
52  Cvor * glava = NULL;
       double broj;
54
       /* Učitavanje se vrši do kraja ulaza. Elementi se dodaju na kraj
56  liste! */
       while (scanf("%lf", &broj) > 0) {
58  /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
           memorije za nov cvor. Memoriju alociranu za cvorove liste
```

```

60     treba osloboditi. */
61     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
62         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
63         oslobodi_listu(&glava);
64         exit(EXIT_FAILURE);
65     }
66 }

67 /* Pozivanje funkcije da dopuni listu. Ako je funkcija vratila 1,
68    onda je bilo greske pri alokaciji memorije za nov cvor. Memoriju
69    alociranu za cvorove liste treba osloboditi. */
70 if (dopuni_listu(&glava) == 1) {
71     fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
72     oslobodi_listu(&glava);
73     exit(EXIT_FAILURE);
74 }

75 /* Ispisivanje liste */
76 ispisi_listu(glava);

77 /* Oslobadjanje liste */
78 oslobodi_listu(&glava);

82 return 0;
84 }

```

### Rešenje 5.9

```

#include <stdio.h>
#include <stdlib.h>
#include "matrica.h"

/* Funkcija racuna zbir elemenata v-te vrste */
int zbir_vrste(int **M, int n, int v)
{
    int j, zbir = 0;
    for (j = 0; j < n; j++)
        zbir += M[v][j];
    return zbir;
}

/* Funkcija racuna zbir elemenata k-te kolone */
int zbir_kolone(int **M, int n, int k)
{
    int i, zbir = 0;
    for (i = 0; i < n; i++)
        zbir += M[i][k];
    return zbir;
}

```



```
22  /* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
24  argument magican. Ukoliko jeste magican funkcija vraća 1, inace 0.
    Argument funkcije zbir ce sadržati zbir elemenata neke vrste ili
26  kolone ukoliko je kvadrat magican. */
    int magicni_kvadrat(int **M, int n, int *zbirmagicnog)
28  {
        int i, j;
30  int zbir = 0, zbir_pom;
        /* Promenljivu zbir inicijalizujemo na zbir 0-te vrese */
32  zbir = zbir_vrste(M, n, 0);

        /* Racunaju se zbirovi u ostalim vrstama i ako neki razlikuje od
           vrednosti promeljive zbir funkcija vraća 1 */
34  for (i = 1; i < n; i++) {
            zbir_pom = zbir_vrste(M, n, i);
36  if (zbir_pom != zbir)
            return 0;
40  }

        /* Racunaju se zbirovi u svim kolonama i ako neki razlikuje od
           vrednosti promeljive zbir funkcija vraća 1 */
42  for (j = 0; j < n; j++) {
            zbir_pom = zbir_kolone(M, n, j);
44  if (zbir_pom != zbir)
            return 0;
46  }

        /* Inace su zbirovi svih vrsta i kolona jednaki, postavlja se
           vrednost u zbirmagicnog i funkcija vraća 1 */
48  *zbirmagicnog = zbir;
        return 1;
50  }
52  }

54  int main()
    {
56  int n, i, j;
        int **matrica = NULL;
58  int zbir = 0, zbirmagicnog = 0;
        scanf("%d", &n);

60

        /* Provera da li je n strogo pozitivan */
62  if (n <= 0) {
            printf("-1\n");
64  exit(EXIT_FAILURE);
        }

66

        /* Dinamicka alokacija matrice dimenzije nxn */
68  matrica = alociraj_matricu(n);
        if (matrica == NULL) {
70  printf("-1\n");
            exit(EXIT_FAILURE);
72  }
```

```
74  /* Ucitavanje elemenata matrice sa standardnog ulaza */
    ucitaj_matricu(matrica, n);
76
    /* Ispisivanje rezultata na osnovu fukcije magicni_kvadrat */
78  if (magicni_kvadrat(matrica, n, &zbirdmagicnog)) {
    printf("%d\n", zbirdmagicnog);
80  } else
    printf("-\n");
82
    /* Oslobadjanje dinamicki alocirane memorije */
84  matrica = dealociraj_matricu(matrica, n);
86  return 0;
}
```