

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 Mladen Nikolić*, docent 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 . . . . .	19
<b>2</b>	<b>Pokazivači</b>	<b>64</b>
2.1	Pokazivačka aritmetika . . . . .	64
2.2	Višedimenzioni nizovi . . . . .	68
2.3	Dinamička alokacija memorije . . . . .	72
2.4	Pokazivači na funkcije . . . . .	77
2.5	Rešenja . . . . .	79
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>115</b>
3.1	Pretraživanje . . . . .	115
3.2	Sortiranje . . . . .	120
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	129
3.4	Rešenja . . . . .	134
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>208</b>
4.1	Liste . . . . .	208
4.2	Stabla . . . . .	219
4.3	Rešenja . . . . .	228
<b>5</b>	<b>Ispitni rokovi</b>	<b>318</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	318
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	320
5.3	Programiranje 2, praktični deo ispita, septembar 2015. . . . .	322
5.4	Rešenja . . . . .	324

# Glava 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

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

- (a) Definisati 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

[illegible]

## Test 2

```
| ULAZ:  
|      0x80  
| IZLAZ:  
|      00000000000000000000000001000000
```

### 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:

- (a) formiranjem odgovarajuće maske i njenim pomeranjem
- (b) 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
101010100100000000000000000000000
    
```

### Test 2

```

ULAZ:
-15
IZLAZ:
111111111111111111111111111101011
110101111111111111111111111111111
    
```

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 2 10	ULAZ: 5 3	ULAZ: 9 4
IZLAZ: 1024	IZLAZ: 125	IZLAZ: 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 rekurzivnu funkciju koja računa  $n$ -ti element u nizu  $F$ , ali tako da se problemi manje dimenzije rešavaju samo jedan put. Napisati program koji testira napisanu funkciju za vrednosti koeficijenata i prirodan broj  $n$  koji se unose sa standardnog ulaza.

### Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite koeficijente 2 3
|| Unesite koji clan niza se racuna 5
|| F(5) = 61
```

### Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:
|| 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. 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
|| Suma elemenata je 15
```

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:
|| 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 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite tri cela broja:
1 2 3
Unesite elemente niza:
4 1 2 3 4 5
Uneti brojevi jesu uzastopni clanovi niza.
```

### 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 1*

```
ULAZ: 0x7F
IZLAZ: 7
```

## Test 2

```
ULAZ:
    0x00FF00FF
IZLAZ:
    16
```

*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 1*

```
ULAZ:  
    10  
IZLAZ:  
    00000000000000000000000000001010
```

## 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 1*

```

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

```

*Test 2*

```

|| ULAZ:
|| 125
|| IZLAZ:
|| 7

```

*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.

Test 1	Test 2	Test 3
<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 prvog kompleksnog broja u promenljivu z1, a potom
    njegovo ispisivanje na standardni izlaz */
124     ucitaj_kompleksan_broj(&z1);
126     ispisi_kompleksan_broj(z1);
    printf("\n");
128
    /* Ucitavanje drugog kompleksnog broja u promenljivu z2, a potom
    njegovo ispisivanje na standardni izlaz */
130     ucitaj_kompleksan_broj(&z2);
132     ispisi_kompleksan_broj(z2);
    printf("\n");
134     getchar();
136
    printf("Unesite znak (+,-,*):");
    c = getchar();
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");
    }
150     else if(c == '-'){
        /* Oduzimanje kompleksnih brojeva */
152         z = oduzmi(z1, z2);
        printf("\n");
        ispisi_kompleksan_broj(z1);
        printf(" - ");
154         ispisi_kompleksan_broj(z2);
        printf(" = ");
156         ispisi_kompleksan_broj(z);
        printf("\n");
160     }
    else{
162         /* Mnozenje kompleksnih brojeva */
        z = mnozi(z1, z2);
        printf("\n");
164         ispisi_kompleksan_broj(z1);
        printf(" * ");
166         ispisi_kompleksan_broj(z2);
        printf(" = ");
168         ispisi_kompleksan_broj(z);
        printf("\n");
170     }

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

178     /* Racunanje i ispisivanje konjugovano kompleksan broj od z2 */
    printf("\nNjegov konjugovano kompleksan broj: ");
180     ispisi_kompleksan_broj(konjugovan(z));
    printf("\n");
182

184     /* Testiranje funkcije koja racuna argument kompleksnih brojeva */
    printf("\nArgument kompleksnog broja %.f\n", argument(z));
186     return 0;
}

```

## Rešenje 1.2

Datoteka 1.1: *complex.c*

```

1 /* complex.c */

```

```
3  /* Uključuje se zaglavlje neophodno za rad sa kompleksnim brojevima.
   Ovdje je to neophodno jer nam je neophodno da bude poznata
5  definicija tipa KompleksanBroj. Takođe, time su uključena
   zaglavlja standardne biblioteke koja su navedena u complex.h */
7  #include "complex.h"

9  /* Funkcija učitava sa standardnog ulaza realan i imaginaran deo
   kompleksnog broja i smesta ih u strukturu čija adresa je argument
11 funkcije */
void učitaj_kompleksan_broj(KompleksanBroj * z)
13 {
    printf("Unesite realan i imaginaran deo kompleksnog broja: ");
15     scanf("%f", &z->real);
    scanf("%f", &z->imag);
17 }

19 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
   se šalje kao argument u obliku (x + y i) */
21 void ispiši_kompleksan_broj(KompleksanBroj z)
    {
23         printf("(");
        if (z.real != 0) {
25             printf("%.2f", z.real);

27             if (z.imag > 0)
                printf(" + %.2f i", z.imag);
29             else if (z.imag < 0)
                printf(" - %.2f i", -z.imag);
31         } else
            printf("%.2f i", z.imag);
33
35         if (z.imag == 0 && z.real == 0)
            printf("0");

37         printf(")");
    }
39

41 /* Funkcija vraća vrednosti realnog dela kompleksnog broja */
float realan_deo(KompleksanBroj z)
    {
43         return z.real;
    }
45

47 /* Funkcija vraća vrednosti imaginarnog dela kompleksnog broja */
float imaginaran_deo(KompleksanBroj z)
    {
49         return z.imag;
    }
51

53 /* Funkcija vraća vrednost modula kompleksnog broja koji joj se šalje
   kao argument */
```

```
float moduo(KompleksanBroj z)
55 {
    return sqrt(z.real * z.real + z.imag * z.imag);
57 }

/* Funkcija vraca vrednost konjugovano kompleksnog broja koji
   odgovara kompleksnom broju poslatom kao argument */
61 KompleksanBroj konjugovan(KompleksanBroj z)
{
63     KompleksanBroj z1 = z;
    z1.imag *= -1;
65     return z1;
}

/* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
   argumenata funkcije */
69 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
71 {
    KompleksanBroj z = z1;
73
    z.real += z2.real;
75     z.imag += z2.imag;
77
    return z;
}

/* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
   argumenata funkcije */
81 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
83 {
    KompleksanBroj z = z1;
85
    z.real -= z2.real;
87     z.imag -= z2.imag;
89
    return z;
}

/* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
   argumenata funkcije */
93 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
95 {
    KompleksanBroj z;
97
    z.real = z1.real * z2.real - z1.imag * z2.imag;
99     z.imag = z1.real * z2.imag + z1.imag * z2.real;
101
    return z;
}

/* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
   kao argument */
103
105
```

```

float argument(KompleksanBroj z)
107 {
    return atan2(z.imag, z.real);
109 }

```

Datoteka 1.2: *complex.h*

```

1  /* complex.h */
2  /*
3   Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
4   deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavlje
5   nikada ne treba da sadrzi definicije funkcija. Da bi neki program
6   mogao da koristi ove brojeve i funkcije iz ove biblioteke,
7   neophodno je da ukljuci ovo zaglavlje. */
8
9  /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i time
10  onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci. Niska
11  posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
12  narednoj pretprocesorskoj define direktivi. */
13  #ifndef _COMPLEX_H
14  #define _COMPLEX_H
15
16  /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
17  koje se koriste u definicijama funkcija navedenim u complex.c */
18  #include <stdio.h>
19  #include <math.h>
20
21  /* Struktura KompleksanBroj */
22  typedef struct {
23      float real;
24      float imag;
25  } KompleksanBroj;
26
27  /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
28  definisane u complex.c */
29  void ucitaj_kompleksan_broj(KompleksanBroj * z);
30
31  void ispisi_kompleksan_broj(KompleksanBroj z);
32
33  float realan_deo(KompleksanBroj z);
34
35  float imaginaran_deo(KompleksanBroj z);
36
37  float moduo(KompleksanBroj z);
38
39  KompleksanBroj konjugovan(KompleksanBroj z);
40
41  KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
42
43  KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);

```

```

45 KomplexsanBroj mnozi(KomplexsanBroj z1, KomplexsanBroj z2);
47 float argument(KomplexsanBroj z);
49 /* Kraj zakljucanog dela */
    #endif

```

Datoteka 1.3: *main.c*

```

/* main.c */
2
3
4 /******
   Ovaj program koristi korektno definisanu biblioteku kompleksnih
   brojeva. U zaglavlju complex.h nalazi se definicija kompleksnog
   broja i popis deklaracija podrzanih funkcija, a u complex.c se
   nalaze njihove definicije.
   8
   Kompilacija programa se najjednostavnije postize naredbom
10 gcc -Wall -lm -o izvrsni complex.c main.c
   Kompilacija se moze uraditi i na sledeci nacin:
12 gcc -Wall -c -o complex.o complex.c
14 gcc -Wall -c -o main.o main.c
   gcc -lm -o complex complex.o main.o
16 *****/
18
19 #include <stdio.h>
20 /* Ukljucuje aw zaglavlje neophodno za rad sa kompleksnim brojevima
   */
   #include "complex.h"
22
23 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
   polarni oblik */
24 int main()
26 {
   KomplexsanBroj z;
28
   /* Ucitavamo kompleksan broj */
30 ucitaj_komplexsan_broj(&z);
32
   printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f~n",
        moduo(z), argument(z));
34
   return 0;
36 }

```

### Rešenje 1.3

Datoteka 1.4: *polinom.c*

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

```

50 |
51 | /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
52 |    algoritmom */
53 | /*  $x^4+2x^3+3x^2+2x+1 = ((x+2)x + 3)x + 1$  */
54 | double izracunaj(const Polinom * p, double x)
55 | {
56 |     double rezultat = 0;
57 |     int i = p->stepen;
58 |     for (; i >= 0; i--)
59 |         rezultat = rezultat * x + p->koef[i];
60 |     return rezultat;
61 | }
62 |
63 | /* Funkcija koja sabira dva polinoma */
64 | Polinom saberi(const Polinom * p, const Polinom * q)
65 | {
66 |     Polinom rez;
67 |     int i;
68 |
69 |     rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
70 |
71 |     for (i = 0; i <= rez.stepen; i++)
72 |         rez.koef[i] =
73 |             (i > p->stepen ? 0 : p->koef[i]) + (i >
74 |                 q->
75 |                 stepen ? 0 : q->koef[i]);
76 |     return rez;
77 | }
78 |
79 | /* Funkcija mnozi dva polinoma p i q */
80 | Polinom pomnozi(const Polinom * p, const Polinom * q)
81 | {
82 |     int i, j;
83 |     Polinom r;
84 |
85 |     r.stepen = p->stepen + q->stepen;
86 |     if (r.stepen > MAX_STEPEN) {
87 |         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
88 |         exit(EXIT_FAILURE);
89 |     }
90 |
91 |     for (i = 0; i <= r.stepen; i++)
92 |         r.koef[i] = 0;
93 |
94 |     for (i = 0; i <= p->stepen; i++)
95 |         for (j = 0; j <= q->stepen; j++)
96 |             r.koef[i + j] += p->koef[i] * q->koef[j];
97 |
98 |     return r;
99 | }
100 |

```

```

102 /* Funkcija racuna izvod polinoma p */
    Polinom izvod(const Polinom * p)
104 {
    int i;
106     Polinom r;

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

110         for (i = 0; i <= r.stepen; i++)
112             r.koef[i] = (i + 1) * p->koef[i + 1];
    } else
114         r.koef[0] = r.stepen = 0;

116     return r;
    }

118 /* Funkcija racuna n-ti izvod polinoma p */
    Polinom nIzvod(const Polinom * p, int n)
120 {
    int i;
    Polinom r;

124     if (n < 0) {
126         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
        exit(EXIT_FAILURE);
128     }

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

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

136     return r;
138 }

```

Datoteka 1.5: *polinom.h*

```

/* polinom.h */
2
/* Ovim pretrocesorskim direktivama zakljucavamo zaglavlje i time
4     onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci */
#define _POLINOM_H
6
8 #include <stdio.h>
#include <stdlib.h>
10
/* Maksimalni stepen polinoma */

```



```

12 #define MAX_STEPEN 20

14
16 /* Polinome predstavljamo strukturom koja cuva koeficijente (koef[i]
   je koeficijent uz clan x^i) i stepen polinoma */
18 typedef struct {
19     double koef[MAX_STEPEN + 1];
20     int stepen;
21 } Polinom;

22 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku
   Polinom prenosimo po adresi, da bi uštedeli kopiranje cele
24 strukture, vec samo prenosimo adresu na kojoj se nalazi polinom
   kog ispisujemo */
26 void ispisi(const Polinom * p);

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

30
32 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
   algoritmom */
   /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
34 double izracunaj(const Polinom * p, double x);

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

38
40 /* Funkcija mnozi dva polinoma p i q */
   Polinom pomnozi(const Polinom * p, const Polinom * q);

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

44
46 /* Funkcija racuna n-ti izvod polinoma p */
   Polinom nIzvod(const Polinom * p, int n);
   #endif

```

Datoteka 1.6: *main.c*

```

1 /* main.c */

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

5
6 /*
7    Prevodjenje: gcc -o test-polinom polinom.c main.c

9    ili: gcc -c polinom.c gcc -c main.c gcc -o test-polinom polinom.o
   main.o */

11 int main(int argc, char **argv)

```

```

13 {
14     Polinom p, q, r;
15     double x;
16     int n;
17
18     /* Unos polinoma */
19     printf
20         ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
21          najveceg stepena do nultog):\n");
22     p = ucitaj();
23
24     /* Ispis polinoma */
25     ispisi(&p);
26
27     /* Unesimo drugi polinom */
28     printf
29         ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
30          od najveceg stepena do nultog):\n");
31     q = ucitaj();
32
33     /* Sabiramo polinome i ispisujemo zbir ta dva polinoma */
34     r = saberi(&p, &q);
35     printf("Zbir polinoma je: ");
36     ispisi(&r);
37
38     /* Mnozimo polinome i ispisujemo proizvod ta dva polinoma */
39     r = pomnozi(&p, &q);
40     printf("Prozvod polinoma je polinom r:\n");
41     ispisi(&r);
42
43     printf("Unesite tacku u kojoj racunate vrednost polinoma r\n");
44     scanf("%lf", &x);
45
46     /* Ispisujemo vrednost polinoma u toj tacki */
47     printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&r, x));
48
49     /* Izvod polinoma */
50     printf("Unesite izvod polinoma koji zelite:\n");
51     scanf("%d", &n);
52     r = nIzvod(&p, n);
53     printf("%d. izvod polinoma r je: ", n);
54     ispisi(&r);
55
56     /* Uspesno završavamo program */
57     return 0;
58 }

```

## Rešenje 1.5

```

1 #include <stdio.h>
2

```

```

4  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji. Bitove koji predstavljaju binarnu
6  reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
   najveće težine ka bitu najmanje težine. */
   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 najveće težine sadrži jedinicu, a na svim ostalim
16     mestima sadrži nulu. U svakoj iteraciji maska se menja tako sto
   se jedini bit jedinica pomera udesno, kako bi se ocitao naredni
18     bit broja x koji je argument funkcije. Odgovarajući karakter,
   ('0' ili '1'), ispisuje se na standardnom izlazu. Neophodno je
20     da promenljiva maska bude deklarirana kao neoznačen ceo broj
   kako bi se siftovanjem u desno vrsilo logicko siftovanje
22     (popunjavanje nulama) a ne aritmeticko siftovanje (popunjavanje
   znakom broja). */
   for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
24     putchar(x & maska ? '1' : '0');

26     putchar('\n');
   }
30

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

38     return 0;
   }

```

## Rešenje 1.6

```

1  #include <stdio.h>

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

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

```

```
13     putchar('\n');
14 }
15
16 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
17    kreiranjem odgovarajuce maske i njenim pomeranjem */
18 int count_bits1(int x)
19 {
20     int br = 0;
21     unsigned wl = sizeof(unsigned) * 8 - 1;
22
23     /* Formiranje se maska cija binarna reprezentacija izgleda
24        100000...0000000, koja služi za ocitavanje bita najveće težine.
25        U svakoj iteraciji maska se pomera u desno za 1 mesto, i
26        ocitavamo sledeci bit. Petlja se završava kada više nema
27        jedinica tj. kada maska postane nula. */
28     unsigned maska = 1 << wl;
29     for (; maska != 0; maska >>= 1)
30         x & maska ? br++ : 1;
31
32     return br;
33 }
34
35 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
36    formiranjem odgovarajuce maske i pomeranjem promenljive x */
37 int count_bits2(int x)
38 {
39     int br = 0;
40     unsigned wl = sizeof(int) * 8 - 1;
41
42     /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1
43        vrsila bi aritmeticko pomeranje u desno, tj. popunjavanje bita
44        najveće težine bitom znaka. U tom slucaju nikad ne bi bio
45        ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
46        petlji. Zbog toga se koristi pomeranje broja x ulevo i maska
47        koja ocitava bit najveće težine. */
48
49     unsigned maska = 1 << wl;
50     for (; x != 0; x <<= 1)
51         x & maska ? br++ : 1;
52
53     return br;
54 }
55
56 int main()
57 {
58     int x, i;
59     printf("Unesite broj:\n");
60     scanf("%x", &x);
61     printf("Unesite redni broj funkcije:\n");
62     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;

9     /* Formira se maska 100000...00000000 */
10    unsigned maska = 1 << (velicina - 1);

12    /* Rezultat se inicijalizuje vrednoscu 0 */
13    unsigned rezultat = 0;

15    /* Promenljiva x se pomera u levo sve dok postoje jedinice u njenoj
       binarnoj reprezentaciji (tj. sve dok je promenljiva x razlicita
       od nule). */
17    for (; x != 0; x <<= 1) {
19        /* Za svaku jedinicu koja se koriscenjem maske detektuje na
           poziciji najveće težine u binarnoj reprezentaciji promenljive
           x, potiskuje se jedna nova jedinicu sa leva u rezultat */
21        if (x & maska) {
23            rezultat >>= 1;
            rezultat |= maska;
25        }
26    }

27    return rezultat;
29 }

31 /* Funkcija vraca najmanji neoznaceni broj sastavljen od istih bitova
   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     /* Promenljiva x se pomera u desno sve dok postoje jedinice u
       njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
       razlicita od nule). */
39     for (; x != 0; x >>= 1) {
41

```

```

43     /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
44        detektuje na poziciji najmanje tezine u binarnoj
45        reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
46        sa desna u rezultat */
47     if (x & 1) {
48         rezultat <= 1;
49         rezultat |= 1;
50     }
51     return rezultat;
52 }
53
54 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
55    celog broja u memoriji */
56 void print_bits(int x)
57 {
58     unsigned velicina = sizeof(int) * 8;
59     unsigned maska;
60
61     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
62         putchar(x & maska ? '1' : '0');
63
64     putchar('\n');
65 }
66
67 int main()
68 {
69     int broj;
70     scanf("%x", &broj);
71
72     printf("Najveci:\n");
73     print_bits(najveci(broj));
74
75     printf("Najmanji:\n");
76     print_bits(najmanji(broj));
77
78     return 0;
79 }

```

### 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     Na primer, za n=5 i p=10 cilj je maska oblika
18     1111 1111 1111 1111 1111 1000 0011 1111
19     To se postize na sledeci nacin:
20     ~0                1111 1111 1111 1111 1111 1111 1111 1111
21     (~0 << n)         1111 1111 1111 1111 1111 1111 1110 0000
22     ~(~0 << n)        0000 0000 0000 0000 0000 0000 0001 1111
23     ~(~0 << n) << (p-n+1) 0000 0000 0000 0000 0000 0111 1100 0000
24     ~(~0 << n) << (p-n+1) 1111 1111 1111 1111 1111 1000 0011 1111
25 *****/
26     unsigned maska = ~(~0 << n) << (p - n + 1);
27
28     return x & maska;
29 }
30
31 /*****
32     Funkcija postavlja na 1 n bitova pocev od pozicije p.
33     Pozicije se broje pocev od pozicije najnižeg bita, pri cemu
34     se broji od nule . Npr, za n=5, p=10
35     1010 1011 1100 1101 1110 1010 1110 0111
36     1010 1011 1100 1101 1110 1111 1110 0111
37 *****/
38     unsigned set(unsigned x, unsigned n, unsigned p)
39     {
40         /*****
41             Cilj je samo odredjenih n bitova postaviti na 1, dok
42             ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
43             formira se maska oblika
44             0000 0000 0000 0000 0000 0111 1100 0000
45             prateci vrlo slican postupak kao za prethodnu funkciju
46 *****/
47         unsigned maska = ~(~0 << n) << (p - n + 1);
48
49         return x | maska;
50     }
51
52 /*****
53     Funkcija vraća celobrojno polje bitova, desno poravnato, koje
54     predstavlja n bitova pocev od pozicije p u binarnoj
55     reprezentaciji broja x, pri cemu se pozicija broji sa desna
56     ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
57     i broj cija je binarna reprezentacija:

```

```

62     1010 1011 1100 1101 1110 1010 1110 0111
        trazi se
64     0000 0000 0000 0000 0000 0000 0000 1011
        *****/
66     unsigned get_bits(unsigned x, unsigned n, unsigned p)
        {
68
69     /*
70     Kreira se maska kod koje su poslednjih n bitova 1, a
        ostali su 0. Na primer za n=5
72     0000 0000 0000 0000 0000 0000 0001 1111
        *****/
74     unsigned maska = ~(~0 << n);

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

82
83     /* Funkcija vraća broj x kome su n bitova pocev od pozicije p
        postavljeni na vrednosti n bitova najnize tezine binarne
        reprezentacije broja y */
84
85     unsigned set_n_bits(unsigned x, unsigned n, unsigned p, unsigned y)
        {
86
87     /*
88     Kreira se maska kod kod koje su poslednjih n bitova 1, a
        ostali su 0. Na primer za n=5
90     0000 0000 0000 0000 0000 0000 0001 1111
        *****/
92     unsigned last_n_1 = ~(~0 << n);
93
94     /*
95     Kao sto je i u funkciji reset, i ovde se kreira masku koja ima n
        bitova postavljenih na 0 pocevsi od pozicije p, dok su
96     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 pocev od p */
        unsigned x_reset = x & middle_n_0;

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

```



```

114     return x_reset ^ y_shift_middle;
115 }
116
117 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
118    njih n */
119 unsigned invert(unsigned x, unsigned n, unsigned p)
120 {
121     /******
122        Formira se maska sa n jedinica pocev od pozicije p.
123        Na primer za n=5 i p=10
124        0000 0000 0000 0000 0000 0111 1100 0000
125        *****/
126     unsigned maska = ~(~0 << n) << (p - n + 1);
127
128     /* Operator ekskluzivno ili invertuje sve bitove gde je
129        odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
130     return maska ^ x;
131 }
132
133
134 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
135    celog broja u memoriji */
136 void print_bits(int x)
137 {
138     unsigned velicina = sizeof(int) * 8;
139     unsigned maska;
140
141     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
142         putchar(x & maska ? '1' : '0');
143
144     putchar('\n');
145 }
146
147
148
149
150 int main()
151 {
152     unsigned x, p, n, y;
153     printf("Unesite neoznaceni broj x:\n");
154     scanf("%u", &x);
155     printf("Unesite neoznaceni broj n:\n");
156     scanf("%u", &n);
157     printf("Unesite neoznaceni broj p:\n");
158     scanf("%u", &p);
159     printf("Unesite neoznaceni broj y:\n");
160     scanf("%u", &y);
161
162     /*Stampanje broja x i y*/
163     printf("x = %u = ", x);
164     print_bits(x);

```

```

166 printf("y = %u = ", y);
    print_bits(y);
168
    printf("reset(%u,%u,%u) = ", x, n, p);
170 print_bits(reset(x, n, p));
172
    printf("set(%u,%u,%u) = ", x, n, p);
    print_bits(set(x, n, p));
174
    printf("get_bits(%u,%u,%u) = ", x, n, p);
176 print_bits(get_bits(x, n, p));
178
    printf("set_n_bits(%u,%u,%u,%u) = ", x, n, p, y);
    print_bits(set_n_bits(x, n, p, y));
180
    printf("invert(%u,%u,%5u) = ", x, n, p);
182 print_bits(invert(x, n, p));
184
    return 0;
}

```

### Rešenje 1.9

```

1 #include <stdio.h>
2
3 /*****
4  Funkcija binarnu reprezentaciju svog argumenta x rotira u
5  levo za n mesta i vraća odgovarajući neoznačen ceo broj čija
6  je binarna reprezentacija dobijena nakon rotacije.
7  Na primer za n=5 i x čija je interna reprezentacija
8  1010 1011 1100 1101 1110 0001 0010 0011
9  funkcija vraća neoznačen ceo broj čija je binarna
10 reprezentacija:
11 0111 1001 1011 1100 0010 0100 0111 0101
12 *****/
13 unsigned rotate_left(int x, unsigned n)
14 {
15     unsigned first_bit;
16     /* Maska koja ima samo najviši bit postavljen na 1 neophodna da bi
17     pre siftovanja u levo za 1 najviši bit bio sačuvan. */
18     unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
19     int i;
20
21     /* n puta se vrši rotaciju za jedan bit u levo. U svakoj iteraciji
22     se odredi prvi bit, a potom se pomera binarna reprezentacija
23     trenutne vrednosti promenljive x u levo za 1. Nakon toga, potom
24     najniži bit se postavlja na vrednost koju je imao prvi bit koji
25     je istisnut siftovanjem */
26     for (i = 0; i < n; i++) {
27         first_bit = x & first_bit_mask;
28         x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
29     }
30 }

```

```

29     }
    return x;
31 }

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

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

    57     return x;
    59 }

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

67

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

        79     if (x < 0)
    /*****/

```

```

81         Siftovanjem u desno broja koji je negativan dobija se 1
83         kao bit na najvisoj poziciji. Na primer ako je x
85         1010 1011 1100 1101 1110 0001 0010 001b
            (sa b je oznacen ili 1 ili 0 na najnižoj poziciji)
87         Onda je sadržaj promenljive last_bit:
            0000 0000 0000 0000 0000 0000 0000 000b
89         Nakon siftovanja sadržaja promenljive x za 1 u desno
            1101 0101 1110 0110 1111 0000 1001 0001
91         Kako bi umesto 1 na najvisoj poziciji u trenutnoj
            binarnoj reprezentaciji x bilo postavljeno b nije
93         dovoljno da se siftuje na najvisu poziciju jer bi se
            time dobile 0, a u ovom slučaju su potrebne jedinice
            zbog bitovskog & zato se prvo vrši komplementiranje, a
            zatim siftovanje
95         ~last_bit << (sizeof(int)*8 -1)
            B000 0000 0000 0000 0000 0000 0000 0000
97         gde B oznacava ~b.
            Potom se ponovo vrši 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 ekran 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 neoznaceni broj x:");
129     scanf("%x", &x);
    printf("Unesite neoznaceni 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 vraca vrednost cija je binarna reprezentacija slika
5   u ogledalu binarne reprezentacije broja x koji se prosledjuje
   kao argument funkcije. Na primer za x
7   cija binarna reprezentacija izgleda ovako
   101010111100110111100100100100011
9   funkcija treba da vrati broj cija binarna reprezentacija
   izgleda:
11  11000100100001111011001111010101
   *****/
13 unsigned mirror(unsigned x)
{
15     unsigned najnizi_bit;
    unsigned rezultat = 0;

17     int i;
    /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
19     vrednosti broja x se određuje i pamti u promenljivoj
    najnizi_bit, nakon čega se na promenljivu x primeni siftovanje u
21     desno. */
    for (i = 0; i < sizeof(x) * 8; i++) {
23         najnizi_bit = x & 1;
        x >>= 1;
        /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
25         postavljeni bitovi dobijaju veću poziciju. Novi bit se
        postavlja na najnižu poziciju */
27         rezultat <<= 1;
        rezultat |= najnizi_bit;
29     }
    return rezultat;
31 }
33 }

35 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju

```

```

37     celog broja u memoriji */
void print_bits(int x)
39 {
    unsigned velicina = sizeof(int) * 8;
    unsigned maska;
    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
43     putchar(x & maska ? '1' : '0');

    putchar('\n');
45 }
47
int main()
49 {
    int broj;
    scanf("%x", &broj);
51
    /* Ispisuje se binarna reprezentaciju unetog broja */
    print_bits(broj);
53
    /* Ispisuje se binarna reprezentaciju broja dobijenog pozivom
       funkcije mirror */
    print_bits(mirror(broj));
55
57
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 {
    int broj_nula, broj_jedinica;
    unsigned int maska;
8
    broj_nula = 0;
    broj_jedinica = 0;
10
    /* Maska je inicijalizovana tako da moze da analizira bit najvece
       tezine */
12     maska = 1 << (sizeof(unsigned int) * 4 - 1);
14
    /* 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) {
20
22         /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili

```

```

24     1 i uveca se odgovarajuci brojac */
25     if (n & maska) {
26         broj_jedinica++;
27     } else {
28         broj_nula++;
29     }
30
31     /* Pomera se maska u desnu stranu */
32     maska = maska >> 1;
33 }
34
35 /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
36    suprotnom vraca 0 */
37 return (broj_jedinica > broj_nula) ? 1 : 0;
38 }
39
40 int main()
41 {
42     unsigned int n;
43
44     scanf("%u", &n);
45
46     printf("%d\n", Broj01(n));
47
48     return 0;
49 }
50

```

### Rešenje 1.12

```

1  #include <stdio.h>
2
3  int broj_parova(unsigned int x)
4  {
5
6      int broj_parova;
7      unsigned int maska;
8
9      /* Vrednost promenljive koja predstavlja broj parova se
10         inicijalizuje na 0 */
11     broj_parova = 0;
12
13     /* Postavlja se maska tako da moze da procitamo da li su dva
14        najmanja bita u zapisu broja x 11 */
15     /* Binarna reprezentacija broja 3 je 000....00011 */
16     maska = 3;
17
18     while (x != 0) {
19
20         /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
21            */

```

```

22     if ((x & maska) == maska) {
        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
int main()
36 {
    unsigned int x;
38
    scanf("%u", &x);
40
    printf("%d\n", broj_parova(x));
42
    return 0;
44
}

```

### Rešenje 1.14

```

#include <stdio.h>
2
/*
4     Niska koja se formiramo je duzine (sizeof(unsigned int)*8)/4 +1
        jer su za svaku heksadekadnu cifru potrebne 4 binarne cifre i
6     jedna dodatna pozicija za terminirajucu nulu.

8     Prethodni izraz je identican sa sizeof(unsigned int)*2+1. */
10 #define MAX_DUZINA sizeof(unsigned int)*2 +1
12
void prevod(unsigned int x, char s[])
14 {
16     int i;
    unsigned int maska;
18     int vrednost;

20     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
        maska za citanje 4 uzastopne cifre */
22     maska = 15;

24     /*****

```



```

26     Broj se posmatra od pozicije najmanje tezine ka poziciji
    najvece tezine. Na primer za broj
    00000000001101000100001111010101
28     u prvom koraku se citaju bitovi izdvojeni sa <...>:
    0000000000110100010000111101<0101>
30     u drugom koraku:
    000000000011010001000011<1101>0101
32     u trecem koraku:
    00000000001101000100<0011>11010101 i tako redom
34
    Indeks i oznacava poziciju na koju se smesta vrednost.
36
    */
37 for (i = MAX_DUZINA - 2; i >= 0; i--) {
38     /* Vrednost izdvojene cifre */
40     vrednost = x & maska;
42
43     /* 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
    odgovarajuce slovo 'A', 'B', ... 'F') */
48     if (vrednost < 10) {
        s[i] = vrednost + '0';
50     } else {
        s[i] = vrednost - 10 + 'A';
52     }
54
55     /* Primenljiva x se pomera za 4 bita u desnu stranu i time ce u
    narednoj iteraciji biti posmatrane sledece 4 cifre */
56     x = x >> 4;
57 }
58
59 s[MAX_DUZINA - 1] = '\0';
60 }
61
62 int main()
63 {
64
65     unsigned int x;
66     char s[MAX_DUZINA];
67
68     scanf("%u", &x);
69
70     prevod(x, s);
71
72     printf("%s\n", s);
73
74     return 0;
75 }

```

## Rešenje 1.17

```

2  #include <stdio.h>

4  /*****
   Linearno resenje se zasniva na cinjenici:
6   x^0 = 1 x^k = x * x^(k-1)
   *****/
8  int stepen(int x, int k)
   {
10     // printf("Racunam stepen (%d, %d)\n", x, k);
    if (k == 0)
12         return 1;

14     return x * stepen(x, k - 1);
   }

16  /*****
   Celo telo funkcije se moze ovako kratko zapisati
18   return k == 0 ? 1 : x * stepen(x,k-1);
   *****/

20   Druga verzija prethodne funkcije. Obratiti paznju na
22   efikasnost u odnosu na prvu verziju!
   Logaritamsko resenje je zasnovano na cinjenicama:
24   x^0 =1;
   x^k = x * (x^2 )^(k/2) , za neparno k
26   x^k = (x^2)^(k/2) , za parno k
   Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
28   doslo do rezultata, i stoga je efikasnije.
   *****/
30  int stepen2(int x, int k)
   {
32     // printf("Racunam stepen2 (%d, %d)\n",x,k);
    if (k == 0)
34         return 1;

36     /* Ako je stepen paran */
    if ((k % 2) == 0)
38         return stepen2(x * x, k / 2);
    /* Inace (ukoliko je stepen neparan) */
40     return x * stepen2(x * x, k / 2);
   }

42   /* U prethodnim funkcijama iskomentarisano je poziv funkcije printf
44   koji ispisuje odgovarajucu poruku prilikom svakog ulaska u
   funkciju. Odkomentarisati pozive printf funkcije u obe funkcije da
46   uocite razliku u broju rekurzivnih poziva obe verzije. */

48  int main()
   {
50     int x, k;

```

```

scanf("%d%d", &x, &k);

52 printf("%d\n", stepen(x, k));
54 // printf("\n-----\n");
// printf("%d\n", stepen2(2,10));
56 return 0;
}

```

### Rešenje 1.18

```

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

4  #define MAX 100

6  /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
   (posrednu) rekurziju. */

8
10 /* Deklaracija funkcije neparan mora da bude navedena jer se ta
   funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
12 definicije. Funkcija je mogla biti deklarirana i u telu funkcije
   paran. */

14 unsigned neparan(unsigned n);

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

24
26 /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace vraca
   0. */
unsigned neparan(unsigned n)
28 {
30     if (n <= 9)
        return 1;
    else
32     return paran(n / 10);
}

34
36 /* Glavna funkcija za testiranje */
int main()
{
38     int n;
    scanf("%d", &n);
    printf("Uneti broj ima %s paran broj cifara\n",
40

```

```

    (paran(n) == 1 ? "" : "ne"));
42 return 0;
}

```

### Rešenje 1.19

```

1  #include <stdio.h>
    /* Pomocna funkcija koja izracunava n! * result. Koristi repnu
3     rekurziju. Result je argument u kome se akumulira do tada
        izracunatu vrednost faktoriijela. Kada dodje do izlaza iz
5     rekurzije iz rekurzije potrebno je da vratimo result. */
int faktorijelRepna(int n, int result)
7 {
    if (n == 0)
9     return result;

11    return faktorijelRepna(n - 1, n * result);
}

13
14 /* U sledece dve funkcije je prikazan postupak oslobadjanja od repne
15    rekurzije koja postoji u funkciji faktorijelRepna, koristeći
        algoritam sa predavanja.

17
18    Najpre, funkcija se transformise tako sto rekurzivni poziv zemeni
19    sa naredbama kojima se vrednost argumenta funkcije postavlja na
        vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
21    goto naredbe za vracanje na pocetak tela funkcije. */

23 int faktorijelRepna_v1(int n, int result)
    {
25     pocetak:
        if (n == 0)
27         return result;

29     result = n * result;
        n = n - 1;
31     goto pocetak;
    }

33
34 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
35    praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
        iterativno resenje bez bezuslovnih skokova: */
37 int faktorijelRepna_v2(int n, int result)
    {
39     while (n != 0) {
        result = n * result;
41         n = n - 1;
    }

43     return result;
45 }

```

```

47  /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
49  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
51  zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
    sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
53  faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
    funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
55  poziv faktorijelRepna sa pozivom faktorijelRepna_v1, a zatim sa
    pozivom funkcije faktorijelRepna_v2. */
57  int faktorijel(int n)
    {
59      return faktorijelRepna(n, 1);
    }
61
62  /* Test program */
63  int main()
    {
64      int n;
65
66      printf("Unesite n (<= 12): ");
67      scanf("%d", &n);
68      printf("%d! = %d\n", n, faktorijel(n));
69
70      while(1);
71
72      return 0;
73  }

```

### Rešenje 1.20

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Funkcija izracunava n-ti element u nizu F */
5  int F_rekurzivna(int n, int a, int b)
6  {
7      /* Izlaz iz rekurzije */
8      if (n == 0 || n == 1)
9          return n;
10
11     /* Rekurzivni pozivi */
12     return a * F_rekurzivna(n - 1, a, b) + b * F_rekurzivna(n - 2,
13                                                             a, b);
14 }
15
16 /* NAPOMENA: U slucaju da se rekurzijom problem svodi na vise manjih
17 podproblema koji se mogu preklapati, postoji opasnost da se
18 pojedini podproblemi manjih dimenzija resavaju veci broj puta.
    Npr. F(20) = a*F(19) + b*F(18), a F(19) = a*F(18) + b*F(17), tj.

```

```
20 |   problem fibonacci(18) se resava dva puta!! Problemi manjih
22 |   dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
24 |   problem je kombinacija rekurzije sa dinamickim programiranjem.
26 |   Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
28 |   memoriji (obicno u nizovima ili matricama), odakle se koriste ako
30 |   tokom resavanja ponovo budu potrebni.
32 |
34 |   U narednoj funkciji vec izracunati clanovi niza se cuvaju u
36 |   statickom nizu celih brojeva, jer taj niz onda nece biti smesten
38 |   na stek, kao sto je slucaj sa lokalnim promenljivama, vec u
40 |   statickoj memoriji odakle ce biti dostupan svim pozivima
42 |   rekurzivne funkcije. */
44 |
46 |   /* Funkcija izracunava n-ti fibonaccijev broj */
48 |   int F_napredna(int n, int a, int b)
50 |   {
52 |       /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
54 |          staticke promenljive na podrazumevane vrednosti. Stoga, elemente
56 |          celobrojnog niza inicijalizuje na 0 */
58 |       static int f[20];
60 |
62 |       /* Ako je podproblem vec ranije resen, koristi se resenje koje je
64 |          vec izracunato i */
66 |       if (f[n] != 0)
68 |           return f[n];
70 |
72 |       /* Izlaz iz rekurzije */
74 |       if (n == 0 || n == 1)
76 |           return f[n] = n;
78 |
80 |       /* Rekurzivni pozivi */
82 |       return f[n] =
84 |           a * F_napredna(n - 1, a, b) + b * F_napredna(n - 2, a, b);
86 |   }
88 |
90 |   /* Iterativna verzija */
92 |   int F_iterativna(int n, int a, int b)
94 |   {
96 |       int i;
98 |       int F_0 = 0;
100 |       int F_1 = 1;
102 |       int tmp;
104 |
106 |       if (n == 0)
108 |           return 0;
110 |
112 |       for (i = 2; i <= n; i++) {
114 |           tmp = a * F_1 + b * F_0;
116 |           F_0 = F_1;
118 |           F_1 = tmp;
120 |       }
```

```

72     return F_1;
73 }
74
75 /* Test program */
76 int main()
77 {
78     int n;
79     int a;
80     int b;
81
82     printf("Unesite koeficijente\n");
83     scanf("%d%d", &a, &b);
84     printf("Unesite koji clan niza se racuna\n");
85     scanf("%d", &n);
86
87     /* Testirati program za razlicite vrednosti promenljive n. Na
88        primer za n=20, 30, 40, 50, 55, 60 ... */
89     printf("F(%d) = %d\n", n, F_iterativna(n, a, b));
90     // printf("F( %d ) = %d\n", n, F_napredna(n,a,b));
91     // printf("F( %d ) = %d\n", n, F_rekurzivna(n,a,b));
92
93     return 0;
94 }

```

### Rešenje 1.21

```

1 #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));
23
24     return 0;
25 }

```

## Rešenje 1.22

```

1  #include <stdio.h>
   #define MAX_DIM 1000
3
   /*****
5     Ako je n==0, onda je suma(a,0) = 0
     Ako je n>0, onda je suma(a,n) = a[n-1] + suma(a,n-1)
7     Suma celog niza je jednaka sumi prvih n-1 elementa uvecenoj
     za poslednji element celog niza.
9     *****/
   int sumaNiza(int *a, int n)
11  {
13     /* Nije postavljena stroga jednakost n==0, za slucaj da
     korisnik prilikom prvog poziva, posalje negativan broj
     za velicinu niza. */
15     if (n <= 0)
         return 0;
17
     return a[n - 1] + sumaNiza(a, n - 1);
19 }

21 /*****
     Funkcija napisana na drugi nacin:
23     n==0, suma(a,0) = 0
     n >0, suma(a,n) = a[0] + suma(a+1,n-1)
25     Suma celog niza je jednaka zbiru prvog elementa niza i sume
     preostalih n-1 elementa.
27     *****/
   int sumaNiza2(int *a, int n)
29  {
31     if (n <= 0)
         return 0;

33     return a[0] + sumaNiza2(a + 1, n - 1);
   }

35
   int main()
37  {
     int a[MAX_DIM];
39     int n, i = 0;

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

45     /* Ucitava se n elemenata niza. */
     printf("Unesite elemente niza:");
47     for (i = 0; i < n; i++)

```



```

scanf("%d", &a[i]);
49
printf("Suma elemenata je %d\n", sumaNiza(a, n));
51 /* printf("Suma elemenata je %d\n", sumaNiza2(a, n)); */
53
return 0;
}

```

### Rešenje 1.23

```

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

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

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

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

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

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

34     /* Stampa se maksimum unetog niza brojeva */
    printf("%d\n", maksimum_niza(brojevi, n));
36     return 0;
}

```

### Rešenje 1.24

```

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

3
   int skalarno(int a[], int b[], int n)
5  {
   /* Izlazak iz rekurzije */
7   if (n == 0)
       return 0;

9   /* Na osnovu resenja problema dimenzije n-1, resava se problem
11      dimenzije n */
       else
13      return a[n - 1] * b[n - 1] + skalarno(a, b, n - 1);
   }

15
   int main()
17  {
       int i, a[MAX_DIM], b[MAX_DIM], n;

19
       /* Unosi se dimenzija nizova. */
21      printf("Unesite dimenziju nizova:");
       scanf("%d", &n);

23
       /* A zatim i elementi nizova. */
25      printf("Unesite elemente prvog niza:");
       for (i = 0; i < n; i++)
27         scanf("%d", &a[i]);

29      printf("Unesite elemente drugog niza:");
       for (i = 0; i < n; i++)
31         scanf("%d", &b[i]);

33      /* Ispisuje se rezultat skalarnog proizvoda dva učitana niza. */
       printf("Skalarni proizvod je %d\n", skalarno(a, b, n));

35
       return 0;
37  }

```

### Rešenje 1.25

```

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

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

8
10  int bp = br_pojave(x, a, n - 1);

```

```

12     return a[n - 1] == x ? 1 + bp : bp;
13 }
14
15 int main()
16 {
17     int x, a[MAX_DIM];
18     int n, i = 0;
19
20     printf("Unesite ceo broj:");
21     scanf("%d", &x);
22
23     /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz;
24        Promenljiva i predstavlja indeks tekućeg broja */
25     printf("Unesite elemente niza:");
26     i = 0;
27     while (scanf("%d", &a[i]) != EOF) {
28         i++;
29     }
30     n = i;
31
32     printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
33     return 0;
34 }

```

### Rešenje 1.26

```

1 #include<stdio.h>
2 #define MAX_DIM 256
3
4 int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
5 {
6     /* Ako niz ima manje od tri elementa izlazi se iz rekurzije */
7     if (n < 3)
8         return 0;
9
10    else
11        return ((a[n - 3] == x) && (a[n - 2] == y)
12                && (a[n - 1] == z))
13                || tri_uzastopna_clana(x, y, z, a, n - 1);
14 }
15
16 int main()
17 {
18     int x, y, z, a[MAX_DIM];
19     int n;
20
21     /* Ucitavaju se tri cela broja za koje se ispituje da li su
22        uzastopni clanovi niza */
23     printf("Unesite tri cela broja:");
24     scanf("%d%d%d", &x, &y, &z);

```

```

26 printf("Unesite elemente niza:");
   int i = 0;
28 while (scanf("%d", &a[i]) != EOF) {
   i++;
30 }
   n = i;
32
   if (tri_uzastopna_clana(x, y, z, a, n))
34     printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
   else
36     printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
38 return 0;
}

```

### Rešenje 1.27

```

#include <stdio.h>

2
/*****
4 Funkcija koja broji bitove svog argumenta

6 ako je x ==0, onda je count(x) = 0
   inace count(x) = najvisi_bit +count(x<<1)

8
   Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
10 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 stane 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
}

```

```

36 }
37
38 /*****
39      Telo prethodne funkcije je moglo biti zapisano i krace:
40      jednolinijska return naredba sa proverom i rekurzivnim pozivom
41      return ((x & (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1);
42      *****/
43
44 int main()
45 {
46     int x;
47     scanf("%x", &x);
48     printf("%d\n", count(x));
49
50     return 0;
51 }

```

### Rešenje 1.29

```

1  #include<stdio.h>
2
3  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
4     broju */
5
6  int max_oktalna_cifra(unsigned x)
7  {
8      /* Izlazak iz rekurzije */
9      if (x == 0)
10         return 0;
11
12     /* Odredjivanje poslednje heksadekadne cifre u broju */
13     int poslednja_cifra = x & 7;
14     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
15        izbrise poslednja oktalna cifra */
16     int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
17     return poslednja_cifra >
18         max_bez_poslednje_cifre ? poslednja_cifra :
19         max_bez_poslednje_cifre;
20 }
21
22 int main()
23 {
24     unsigned x;
25     scanf("%u", &x);
26     printf("%d\n", max_oktalna_cifra(x));
27     return 0;
28 }

```

### Rešenje 1.30

```

#include<stdio.h>

2
/* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju */
int max_heksadekadna_cifra(unsigned x)
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
{
    /* Izlazak iz rekurzije */
    if (x == 0)
        return 0;
    /* Odredjivanje poslednje heksadekadne cifre u broju */
    int poslednja_cifra = x & 15;
    /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
    njega izbrise poslednja heksadekadna cifra */
    int max_bez_poslednje_cifre = max_heksadekadna_cifra(x >> 4);
    return poslednja_cifra >
        max_bez_poslednje_cifre ? poslednja_cifra :
        max_bez_poslednje_cifre;
}

int main()
{
    unsigned x;
    scanf("%u", &x);
    printf("%d\n", max_heksadekadna_cifra(x));
    return 0;
}

```

### Rešenje 1.31

```

#include<stdio.h>
#include<string.h>
2
/* Niska može imati najviše 32 karaktera + 1 za terminalnu nulu */
#define MAX_DIM 33
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
int palindrom(char s[], int n)
{
    if ((n == 1) || (n == 0))
        return 1;
    return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
}

int main()
{
    char s[MAX_DIM];
    int n;
    scanf("%s", s);
    /* Odredjuje se duzina niske */
}

```

```

22     n = strlen(s);

24     /* Ispisuje se poruka da li je niska palindrom ili nije */
25     if (palindrom(s, n))
26         printf("da\n");
27     else
28         printf("ne\n");

30     return 0;
}

```

### Rešenje 1.32

```

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

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

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

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

    return 0;
}

/* 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
   permutaciji smesta jedan od preostalih celih brojeva n- je
   velicina skupa koji se permutuje Funkciju se poziva sa argumentom
   m=1 jer formiranje permutacije pocinje od 1. pozicije. Stoga, nece
   se koristiti a[0]. */
void permutacija(int a[], int m, int n)
{
    int i;

    /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
       premasila velicinu skupa, onda se svi brojevi vec nalaze u
       permutaciji i ispisuje se permutacija. */
}

```

```
40     if (m > n) {
41         ispisiNiz(a, n);
42         return;
43     }
44
45     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
46        mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
47        Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
48        ovako postavljenom pocetku permutacije. Kada se to završi, vrši
49        se provera da li postoji jos neki broj koji moze da se stavi na
50        m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
51        funkcija završava sa radom. Ukoliko takav broj postoji, onda se
52        ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
53        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 veci 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     scanf("%d", &n);
76     if (n < 0 || n >= MAX_DUZINA_NIZA) {
77         fprintf(stderr,
78             "Duzina permutacije mora biti broj veci od 0 i manji od %
79             d!\n",
80             MAX_DUZINA_NIZA);
81         exit(EXIT_FAILURE);
82     }
83
84     permutacija(a, 1, n);
85
86     exit(EXIT_SUCCESS);
87 }
```

## Rešenje 1.33



```

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

4  /* Rekurzivna funkcija za racunanje binomnog koeficijenta. */
/* ako je k=0 ili k=n, onda je binomni koeficijent 0 ako je k izmedju
6   0 i n, onda je bk(n,k) = bk(n-1,k-1) + bk(n-1,k) */
int binomniKoeficijent(int n, int k)
8  {
    return (0 < k
10         && k < n) ? binomniKoeficijent(n - 1,
                                           k - 1) +
12         binomniKoeficijent(n - 1, k) : 1;
    }

14  /*****
16   Iterativno izracunavanje datog binomnog koeficijenta.

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  *****/

26  /* Prostim opažanjem se uocava da se svaki element n-te hipotenuze
   (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1 hipotenuze. Uz
28  pomenute dve nove ivicne jedinice lako se zakljucuje da ce suma
   elementa n-te hipotenuze biti tacno 2 puta veca. */
30  int sumaElemenataHipotenuze(int n)
31  {
32      return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
33  }

34  int main()
35  {
36      int n, k, i, d, r;

38

40      scanf("%d %d", &d, &r);

42      /* Ispisivanje Paskalovog trougla */
      putchar('\n');
44      for (n = 0; n <= d; n++) {
          for (i = 0; i < d - n; i++)
46              printf(" ");
          for (k = 0; k <= n; k++)
48              printf("%4d", binomniKoeficijent(n, k));
          putchar('\n');
50      }

```

```
52  if (r < 0) {  
53      fprintf(stderr,  
54          "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"  
55      );  
56      exit(EXIT_FAILURE);  
57  }  
58  printf("%d\n", sumaElemenataHipotenuze(r));  
59  exit(EXIT_SUCCESS);  
60 }
```

## 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 « $\ll$ < HEAD koja sadrži datu).  
 ===== koja sadrži datu) NAPOMENA: *Koristiti Varšalov algoritam.*  
 »»»> origin/master

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** 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.

### 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 2.19]

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

### 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 2.21** 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 2.21]

**Zadatak 2.22** 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 2.23** 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.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
```

```
Unesite elemente niza, nulu za kraj:
1 2 3 0
Tražena 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
Tražena matrica je:
-5 -2 -4 -1
-2 -4 -1 -5
-4 -1 -5 -2
-1 -5 -2 -4
```

**Zadatak 2.24** 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 2.25** 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 2.26** Napisati program koji tabelarno štampa vrednosti proizvoljne realne funkcije sa jednim realnim argumentom, odnosno izračunava i ispisuje vrednosti date funkcije na diskretnoj ekvidistantnoj mreži od  $n$  tačaka intervala  $[a, b]$ . Realni brojevi  $a$  i  $b$  ( $a < b$ ) kao i ceo broj  $n$  ( $n \geq 2$ ) se učitavaju sa standardnog ulaza. Ime funkcije se zadaje kao argument komandne linije (`sin`, `cos`, `tan`, `atan`, `acos`, `asin`, `exp`, `log`, `log10`, `sqrt`, `floor`, `ceil`, `sqr`).

### 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 2.26]

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

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

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

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

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

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

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

Granice intervala i  $n$  su argumenti funkcije. Napisati program, koji kao argumente komandne linije prihvata ime funkcije iz zaglavlja `math.h`, krajeve intervala 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  int main()
54  {
55      /* Deklarise se niz od najvise MAX elemenata */
56      int a[MAX];

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

60
61      /* Pokazivac na elemente niza */
62      int *p;

63
64      printf("Unesite dimenziju niza: ");
65      scanf("%d", &n);

66
67      /* Proverava se da li je doslo do prekoračenja ograničenja
68      dimenzije */
69      if (n <= 0 || n > MAX) {
70          fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
71          exit(EXIT_FAILURE);
72      }

73
74      printf("Unesite elemente niza:\n");
75      for (p = a; p - a < n; p++)
76          scanf("%d", p);

77
78      obrni_niz_v1(a, n);

79
80      printf("Nakon obrtanja elemenata, niz je:\n");

81
82      for (p = a; p - a < n; p++)
83          printf("%d ", *p);
84      printf("\n");

85
86      obrni_niz_v2(a, n);

```

```
89     printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
91     for (p = a; p - a < n; p++)
92         printf("%d ", *p);
93     printf("\n");
95     return 0;
96 }
```

## Rešenje 2.2

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

```
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     /* Drugi nacin
31     while (prvi < poslednji) {
32         (*prvi++)++;
33         (*poslednji--)--;
34     } */
35 }
36
37 int main()
38 {
39     int a[MAX];
40     int n;
41     int *p;
42
43     printf("Unesite dimenziju niza: ");
44     scanf("%d", &n);
45
46     /* Proverava se da li je doslo do prekoračenja ograničenja
47        dimenzije */
48     if (n <= 0 || n > MAX) {
49         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
50         exit(EXIT_FAILURE);
51     }
```

```

printf("Unesite elemente niza:\n");
53 for (p = a; p - a < n; p++)
    scanf("%d", p);
55
povecaj_smanji(a, n);
57
printf("Transformisan niz je:\n");
59 for (p = a; p - a < n; p++)
    printf("%d ", *p);
61 printf("\n");
63
return 0;
}

```

## Rešenje 2.4

```

1 #include <stdio.h>
3 int main(int argc, char *argv[])
{
5     int i;
    char tip_ispisa;
7
    printf("Broj argumenata komandne linije je %d.\n",
9         argc);
11
    printf("Kako zelite da ispisete argumente, koriscenjem"
13         " indeksne ili pokazivacke sintakse (I ili P)? ");
    scanf("%c", &tip_ispisa);
15
    printf("Argumenti komandne linije su:\n");
    if (tip_ispisa == 'I') {
17         /* Ispisuju se argumenti komandne linije koriscenjem indeksne
            sintakse */
19         for (i = 0; i < argc; i++)
            printf("%d %s\n", i, argv[i]);
21     } else if (tip_ispisa == 'P') {
        /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
23         sintakse */
        i = argc;
25         for (; argc > 0; argc--)
            printf("%d %s\n", i - argc, *argv++);
27
        /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje na
29         polje u memoriji koje se nalazi iza poslednjeg argumenta
            komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
31         broja argumenta komandne linije to sada moze ponovo da se
            postavi "argv" da pokazuje na nulti argument komandne linije
            */
33         argv = argv - i;
        argc = i;
    }
}

```

```

35 }

37 printf("Pocetna slova argumenata komandne linije:\n");
38 if (tip_ispisa == 'I') {
39     /* koristeći indeksnu sintaksu */
40     for (i = 0; i < argc; i++)
41         printf("%c ", argv[i][0]);
42     printf("\n");
43 } else if (tip_ispisa == 'P') {
44     /* koristeći pokazivačku sintaksu */
45     for (i = 0; i < argc; i++)
46         printf("%c ", **argv++);
47     printf("\n");
48 }
49
50 return 0;
51 }

```

### Rešenje 2.5

```

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

5  /* Funkcija ispituje da li je niska palindrom, odnosno da li
6     se isto cita spreda 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     /* Multi argument komandne linije je ime izvrsnog programa */
20     for (i = 1; i < argc; i++)
21         if (palindrom(*(argv + i)))
22             n++;

23
24     printf
25         ("Broj argumenata komandne linije koji su palindromi je %d.\n",
26          n);
27     return 0;
28 }
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      */
22     if (argc < 3) {
23         printf("Greska: ");
24         printf("Nedovoljan broj argumenata komandne linije.\n");
25         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
26             argv[0]);
27         exit(EXIT_FAILURE);
28     }
29
30     /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
31      komandne linije. */
32     in = fopen(*(argv + 1), "r");
33     if (in == NULL) {
34         fprintf(stderr, "Greska: ");
35         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
36         exit(EXIT_FAILURE);
37     }
38
39     n = atoi(*(argv + 2));
40
41     /* Broje se reci cija je duzina jednaka broju zadatom drugim
42      argumentom komandne linije */
43     while (fscanf(in, "%s", rec) != EOF)
44         if (duzina(rec) == n)
45             br++;
46
47     printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);
48
49     /* Zatvara se datoteka */
50     fclose(in);
51 }
```

```
50     return 0;
    }
```

## Rešenje 2.7

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Implementacija funkcije strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
8  {
9      int i;
10     for (i = 0; *(src + i); i++)
11         *(dest + i) = *(src + i);
12 }
13
14 /* Implementacija funkcije strcmp() iz standardne biblioteke */
15 int poredjenje_niski(char *s, char *t)
16 {
17     int i;
18     for (i = 0; *(s + i) == *(t + i); i++)
19         if (*(s + i) == '\0')
20             return 0;
21     return *(s + i) - *(t + i);
22 }
23
24 /* Implementacija funkcije strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
26 {
27     int i;
28     for (i = 0; *(s + i); i++);
29     return i;
30 }
31
32 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
33    sufiks niske zadate prvi argumentom funkcije */
34 int sufiks_niske(char *niska, char *sufiks)
35 {
36     int duzina_sufiksa = duzina_niske(sufiks);
37     int duzina_niske_pom = duzina_niske(niska);
38     if (duzina_sufiksa <= duzina_niske_pom &&
39         poredjenje_niski(niska + duzina_niske_pom -
40                         duzina_sufiksa, sufiks) == 0)
41         return 1;
42     return 0;
43 }
44
45 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
46    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  
17 /* Funkcija izracunava euklidsku normu matrice */  
  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 elemenati 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 ispisu_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

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 64
5
6  /* Funkcija proverava da li je relacija refleksivna. Relacija je
7   * refleksivna ako je svaki element u relaciji sa sobom, odnosno ako
8   * se u matrici relacije na glavnoj dijagonali nalaze jedinice */
9  int refleksivnost(int m[][MAX], int n)
10 {
11     int i;
12
13     for (i = 0; i < n; i++) {
14         if (m[i][i] != 1)
15             return 0;
16     }
17
18     return 1;
19 }
20
21 /* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono je
22 * odredjeno matricom koja sadrzi sve elemente polazne matrice
23 * dopunjene jedinicama na glavnoj dijagonali */
24 void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
25 {
26     int i, j;
27
28     /* Prepisuju se vrednosti elemenata pocetne matrice */
29     for (i = 0; i < n; i++)
30         for (j = 0; j < n; j++)
31             zatvorenje[i][j] = m[i][j];
32
33     /* Na glavnoj dijagonali se postavljaju jedinice */
34     for (i = 0; i < n; i++)
35         zatvorenje[i][i] = 1;
36 }

```



```
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
    elementom "i". Ovakve matrice su simetricne u odnosu na glavnu
42    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 }
58 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono je
    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
    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 */
88                for (k = 0; k < n; k++)
                    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
8  indeksa kolone jednak n-1 */
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 */
int indeks_min(int m[][MAX], int n)
22 {
    int i, j;
    int min = m[0][0], indeks_kolone = 0;

24    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
26            if (m[i][j] < min) {
                min = m[i][j];
                indeks_kolone = j;
30            }

32    return indeks_kolone;
}

34 /* Funkcija izracunava indeks vrste najvećeg elementa */
int indeks_max(int m[][MAX], int n)
36 {
    int i, j;
    int max = m[0][0], indeks_vrste = 0;

40    for (i = 0; i < n; i++)
42

```

```
44     for (j = 0; j < n; j++)
45         if (m[i][j] > max) {
46             max = m[i][j];
47             indeks_vrstice = i;
48         }
49     return indeks_vrstice;
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
106     return 0;
108 }

```

### 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
13     for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
             scanf("%d", &m[i][j]);
15 }
17
   /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
      */
17 void ispis_matricu(int m[][MAX], int n)
   {
19     int i, j;
21
23     for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++)
             printf("%d ", m[i][j]);
         printf("\n");
25     }
27 }
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
33 int ortonormirana(int m[][MAX], int n)

```

```
{
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 ispisi_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
10    printf("Unesite dimenziju niza: ");
11    scanf("%d", &n);
12
13    /* Alocira se prostor za n celih brojeva */
14    if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
15        fprintf(stderr, "malloc(): ");
16        fprintf(stderr, "greska pri alokaciji memorije.\n");
17        exit(EXIT_FAILURE);
18    }
19
20    printf("Unesite elemente niza: ");
21    for (i = 0; i < n; i++)
22        scanf("%d", &p[i]);
23
24    printf("Niz u obrnutom poretku je: ");
25    for (i = n - 1; i >= 0; i--)
26        printf("%d ", p[i]);
27    printf("\n");
28
29    /* Oslobadja se prostor rezervisan funkcijom malloc() */
30    free(p);
31
32    return 0;
33 }
```

### 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
11    /* Velicina alocirane memorije */
12    int alocirano;
13
14    /* Broj elemenata niza */
15    int n;
```

```
16  /* Broj koji se ucitava sa ulaza */
17  int x;
18  int i;
19  int *b = NULL;
20  char realokacija;

22  /* Inicijalizacija */
23  alocirano = n = 0;

24
25  printf("Unesite zeljeni nacin realokacije (M ili R):\n");
26  scanf("%c", &realokacija);

28  printf("Unesite brojeve, nulu za kraj:\n");
29  scanf("%d", &x);

30  while (x != 0) {
31      if (n == alocirano) {
32          alocirano = alocirano + KORAK;

34          if (realokacija == 'M') {
35              /* Vrsi se realokacija memorije sa novom velicinom */
36              b = (int *) malloc(alocirano * sizeof(int));

38              if (b == NULL) {
39                  fprintf(stderr, "malloc(): ");
40                  fprintf(stderr, "greska pri alokaciji memorije.\n");
41                  free(a);
42                  exit(EXIT_FAILURE);
43              }

44              /* Svih n elemenata koji pocinju na adresi a prepisujemo na
45               novu adresu b */
46              for (i = 0; i < n; i++)
47                  b[i] = a[i];

48              free(a);

49              /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
50               bloka koji je prilikom alokacije zapamcen u promenljivoj b
51              */
52              a = b;
53          } else if (realokacija == 'R') {

54              /* Zbog funkcije realloc je neophodno da i u prvoj iteraciji
55               "a" bude inicijalizovano na NULL */

56              a = (int*) realloc(a, alocirano * sizeof(int));
57              if (a == NULL) {
58                  fprintf(stderr, "realloc(): ");
59                  fprintf(stderr, "greska pri alokaciji memorije.\n");
60                  exit(EXIT_FAILURE);
61              }
62          }
63      }
64  }
```

```

    }
68 }

70 a[n++] = x;

72 scanf("%d", &x);
}

74

76 printf("Niz u obrnutom poretku je: ");
77 for (n--; n >= 0; n--)
78     printf("%d ", a[n]);
79 printf("\n");

80 /* Oslobadja se dinamicki alocirana memorija */
81 free(a);

82

84 exit(EXIT_SUCCESS);
}

```

### Rešenje 2.17

```

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

4

#define MAX 1000

6

/* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
8  nadovezivanja niski. Adresa niza se vraca kao povratna vrednost.
   */
char *nadovezi(char *s, char *t)
10 {
    char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
12                          * sizeof(char));

14     /* Proverava se da li je memorija uspesno alocirana */
    if (p == NULL) {
16         fprintf(stderr, "malloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");
18         exit(EXIT_FAILURE);
    }

20     /* Kopiraju se i nadovezuju niske karaktera */
    strcpy(p, s);
    strcat(p, t);

24     return p;
26 }

28 int main()
{

```

```

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 */
32  for (i = 0; i < n; i++) {
33      A[i] = malloc(sizeof(double) * m);

34      /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
35         potrebno je osloboditi svih i-1 prethodno alociranih vrsta, i
36         alocirani niz pokazivaca */
37      if (A[i] == NULL) {
38          for (j = 0; j < i; j++)
39              free(A[j]);
40          free(A);
41          exit(EXIT_FAILURE);
42      }
43  }

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

48  /* Izracunava se trag matrice, odnosno suma elemenata na glavnoj
49     dijagonali */
50  trag = 0.0;

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

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

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

57  /* Oslobadja se memorija za niz pokazivaca na vrste */
58  free(A);

59  return 0;
60 }

```

### Rešenje 2.19

```

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

4
5  /* Funkcija ucitava matricu sa ulaza */
6  void ucitaj_matricu(int **M, int n, int m)
7  {
8      int i, j;
9

```

```
11     for (i = 0; i < n; i++)
12         for (j = 0; j < m; j++)
13             scanf("%d", &M[i][j]);
14 }
15
16 /* Funkcija ispisuje elemente matrice ispod glavne dijagonale */
17 void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
18 {
19     int i, j;
20
21     for (i = 0; i < n; i++) {
22         for (j = 0; j <= i; j++)
23             printf("%d ", M[i][j]);
24         printf("\n");
25     }
26 }
27
28 int main()
29 {
30     int m, n, i, j;
31     int **matrica = NULL;
32
33     printf("Unesite broj vrsta i broj kolona matrice:\n ");
34     scanf("%d %d", &n, &m);
35
36     /* Alocira se prostor za niz pokazivaca na vrste matrice */
37     matrica = (int **) malloc(n * sizeof(int *));
38     if (matrica == NULL) {
39         fprintf(stderr, "malloc(): Neuspela alokacija\n");
40         exit(EXIT_FAILURE);
41     }
42
43     /* Alocira se prostor za svaku vrstu matrice */
44     for (i = 0; i < n; i++) {
45         matrica[i] = (int *) malloc(m * sizeof(int));
46
47         if (matrica[i] == NULL) {
48             fprintf(stderr, "malloc(): Neuspela alokacija\n");
49             for (j = 0; j < i; j++)
50                 free(matrica[j]);
51             free(matrica);
52             exit(EXIT_FAILURE);
53         }
54     }
55
56     printf("Unesite elemente matrice, vrstu po vrstu:\n");
57     ucitaj_matricu(matrica, n, m);
58
59     printf("Elementi ispod glavne dijagonale matrice:\n");
60     ispisi_elemente_ispod_dijagonale(matrica, n, m);
61
62     /* Oslobadja se dinamicki alocirana memorija za matricu. Prvo se
```

```

        oslobadja memorija rezervisana za svaku vrstu */
63 for (j = 0; j < n; j++)
    free(matrica[j]);
65
/* Zatim se oslobadja memorija za niz pokazivaca na vrste matrice
   */
67 free(matrica);
69 return 0;
}

```

### Rešenje 2.21

```

#include <stdio.h>
2  #include <stdlib.h>
   #include <math.h>
4
/* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni(float **a, int n)
{
8     int i, j;

10    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
12        if (i < j)
            a[i][j] /= 2;
14        else if (i > j)
            a[i][j] *= 2;
16 }

18 /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
   sporedne dijagonale. Element se nalazi ispod sporedne dijagonale
   ukoliko je zbir indeksa vrste i indeksa kolone elementa veci od
   n-1 */
22 float zbir_ispod_sporedne_dijagonale(float **m, int n)
{
24     int i, j;
    float zbir = 0;

26     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
28            if (i + j > n - 1)
30                zbir += fabs(m[i][j]);

32     return zbir;
}
34

/* Funkcija ucitava elemente kvadratne matrice dimenzije n iz zadate
   datoteke */
36 void ucitaj_matricu(FILE * ulaz, float **m, int n)
38 {

```



```
40     int i, j;

42     for (i = 0; i < n; i++)
43         for (j = 0; j < n; j++)
44             fscanf(ulaz, "%f", &m[i][j]);
45 }

46 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
47    standardni izlaz */
48 void ispisi_matricu(float **m, int n)
49 {
50     int i, j;

52     for (i = 0; i < n; i++) {
53         for (j = 0; j < n; j++)
54             printf("%.2f ", m[i][j]);
55         printf("\n");
56     }
57 }

58 /* Funkcija alokira memoriju za kvadratnu matricu dimenzije n */
60 float **alociraj_memoriju(int n)
61 {
62     int i, j;
63     float **m;

64     m = (float **) malloc(n * sizeof(float *));
65     if (m == NULL) {
66         fprintf(stderr, "malloc(): Neuspela alokacija\n");
67         exit(EXIT_FAILURE);
68     }

70     for (i = 0; i < n; i++) {
71         m[i] = (float *) malloc(n * sizeof(float));

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

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

```
92     for (i = 0; i < n; i++)
93         free(m[i]);
94     free(m);
95 }
96
97 int main(int argc, char *argv[])
98 {
99     FILE *ulaz;
100     float **a;
101     int n;
102
103     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska */
104     if (argc < 2) {
105         printf("Greska: ");
106         printf("Nedovoljan broj argumenata komandne linije.\n");
107         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
108         exit(EXIT_FAILURE);
109     }
110
111     /* Otvara se datoteka za citanje */
112     ulaz = fopen(argv[1], "r");
113     if (ulaz == NULL) {
114         fprintf(stderr, "Greska: ");
115         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
116         exit(EXIT_FAILURE);
117     }
118
119     /* Cita se dimenzija matrice */
120     fscanf(ulaz, "%d", &n);
121
122     /* Alocira se memorija */
123     a = alociraj_memoriju(n);
124
125     /* Ucitavaju se elementi matrice */
126     ucitaj_matricu(ulaz, a, n);
127
128     float zbir = zbir_ispod_sporedne_dijagonale(a, n);
129
130     /* Poziva se funkcija za transformaciju matrice */
131     izmeni(a, n);
132
133     /* Ispisuje se rezultat */
134     printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
135     printf("je %.2f.\n", zbir);
136
137     printf("Transformisana matrica je:\n");
138     ispisi_matricu(a, n);
139
140     /* Oslobadja se memorija */
141     oslobodi_memoriju(a, n);
```

```
142  /* Zatvara se datoteka */
    fclose(ulaz);
144
    return 0;
146 }
```

## Rešenje 2.26

```
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)
{
27     return a * a;
}

29
int main(int argc, char *argv[])
31 {
    double a, b;
33    int n;

35    char ime_fje[6];

37    /* Pokazivac na funkciju koja ima jedan argument tipa double i
       povratnu vrednost istog tipa */
39    double (*fp) (double);

41    /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
       */
    if (argc < 2) {
```

```

43     printf("Greska: ");
44     printf("Nedovoljan broj argumenata komandne linije.\n");
45     printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
46           argv[0]);
47     exit(EXIT_FAILURE);
48 }
49
50 /* Niska ime_fje sadrzi ime trazene funkcije koja je navedena u
51    komandnoj liniji */
52 strcpy(ime_fje, argv[1]);
53
54 /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
55    */
56 if (strcmp(ime_fje, "sin") == 0)
57     fp = &sin;
58 else if (strcmp(ime_fje, "cos") == 0)
59     fp = &cos;
60 else if (strcmp(ime_fje, "tan") == 0)
61     fp = &tan;
62 else if (strcmp(ime_fje, "atan") == 0)
63     fp = &atan;
64 else if (strcmp(ime_fje, "acos") == 0)
65     fp = &acos;
66 else if (strcmp(ime_fje, "asin") == 0)
67     fp = &asin;
68 else if (strcmp(ime_fje, "exp") == 0)
69     fp = &exp;
70 else if (strcmp(ime_fje, "log") == 0)
71     fp = &log;
72 else if (strcmp(ime_fje, "log10") == 0)
73     fp = &log10;
74 else if (strcmp(ime_fje, "sqrt") == 0)
75     fp = &sqrt;
76 else if (strcmp(ime_fje, "floor") == 0)
77     fp = &floor;
78 else if (strcmp(ime_fje, "ceil") == 0)
79     fp = &ceil;
80 else if (strcmp(ime_fje, "sqr") == 0)
81     fp = &sqr;
82 else {
83     printf("Program jos uvek ne podrzava trazenu funkciju!\n");
84     exit(EXIT_SUCCESS);
85 }
86
87 printf("Unesite krajeve intervala:\n");
88 scanf("%lf %lf", &a, &b);
89
90 printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
91 printf("(ukljucujuci krajeve intervala)?\n");
92 scanf("%d", &n);
93
94 /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora

```

```
95     uneti broj veci od 2 */
96     if (n < 2) {
97         fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
98         exit(EXIT_FAILURE);
99     }
100
101     /* Ispisuje se ime funkcije */
102     printf("      x %10s(x)\n", ime_fje);
103
104     /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
105        komandne linije */
106     tabela(a, b, n, fp);
107
108     exit(EXIT_SUCCESS);
109 }
```

## Glava 3

# Algoritmi pretrage i sortiranja

### 3.1 Pretraživanje

**Zadatak 3.1** Napisati iterativne funkcije pretraga nizova. Svaka funkcija treba da vrati indeks pozicije na kojoj je pronađen traženi 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 složenosti. 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.



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

[Rešenje 3.5]

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

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

[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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ:  151 44 5 -12 -13 -15 IZLAZ:  3 </pre>	<pre> ULAZ:  100 55 15 0 -15 -124  -155 -258 -315 -516 IZLAZ:  4 </pre>	<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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ:  4 IZLAZ:  2 2 </pre>	<pre> ULAZ:  17 IZLAZ:  4 4 </pre>	<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 Sortiranje

**Zadatak 3.11** 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.*

*Test 1*

```

ULAZ:
23 64 123 76 22 7
IZLAZ:
1

```

*Test 2*

```

ULAZ:
21 654 65 123 65 12 61
IZLAZ:
0

```

*Test 3*

```

ULAZ:
34 30
IZLAZ:
4

```

[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 za sortirane niske proveriti da li su identične.*

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite prvu nisku anagram
Unesite drugu nisku ramgana
jesu

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite prvu nisku anagram
Unesite drugu nisku anagram
nisu

```

*Primer 3*

```

INTERAKCIJA SA PROGRAMOM:
Unesite prvu nisku test
Unesite drugu nisku tset
jesu

```

[Rešenje 3.12]

**Zadatak 3.13** Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza

i neće biti duži od 256 i kraći od jednog elemenata. UPUTSTVO: *Prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.*

Test 1	Test 2	Test 3
ULAZ: 4 23 5 2 4 6 7 34 6 4 5 IZLAZ: 4	ULAZ: 2 4 6 2 6 7 99 1 IZLAZ: 2	ULAZ: 123 IZLAZ: 123

[Rešenje 3.13]

**Zadatak 3.14** 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.*

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

[Rešenje 3.14]

**Zadatak 3.15** Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži **selection**, **merge**, **quick**, **bubble**, **insertion** i **shell sort**. Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za **merge**, **-q** za **quick**, **-b** za **bubble**, **-i** za **insertion** ili **-s** za **shell sort**. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati **selection sort** algoritmom. 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**.

<p><i>Test 1</i></p> <pre> Poziv: time ./a.out 200000 IZLAZ:   real 0m42.168s   user 0m42.100s   sys 0m0.000s </pre>	<p><i>Test 2</i></p> <pre> Poziv: time ./a.out 400000 IZLAZ:   real 2m48.395s   user 2m48.128s   sys 0m0.000s </pre>	<p><i>Test 3</i></p> <pre> Poziv: time ./a.out 800000 IZLAZ:   real 11m13.703s   user 11m12.636s   sys 0m0.000s </pre>
<p><i>Test 4</i></p> <pre> Poziv: time ./a.out 800000 -r IZLAZ:   real 11m21.533s   user 11m20.436s   sys 0m0.020s </pre>	<p><i>Test 5</i></p> <pre> Poziv: time ./a.out 800000 -q IZLAZ:   real 0m0.159s   user 0m0.156s   sys 0m0.000s </pre>	<p><i>Test 6</i></p> <pre> Poziv: time ./a.out 800000 -m IZLAZ:   real 0m0.137s   user 0m0.136s   sys 0m0.000s </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.

<p><i>Primer 1</i></p> <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>	<p><i>Primer 2</i></p> <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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> BIRACKI-SPISAK.TXT Bojan Golubovic Andrija Petrovic Anja Ilic Aleksandra Cvetic Dragan Markovic Ivana Markovic Lazar Micic Marija Stankovic Filip Dukic  IZLAZ: 3 </pre>	<pre> BIRACKI-SPISAK.TXT Milan Milicevic  IZLAZ: 1 </pre>	<pre> DATOTEKA BIRACKI-SPISAK.TXT NE POSTOJI  IZLAZ: Problem pri otvaranju datoteke. </pre>

[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.

<i>Test 1</i>	
<pre> Poziv: ./a.out in.txt out.txt  IN.OUT Petar Petrovic 2007 Milica Antonic 2008 Ana Petrovic 2007 Ivana Ivanovic 2009 Dragana Markovic 2010 Marija Antic 2007 </pre>	<pre> OUT.TXT Marija Antic 2007 Ana Petrovic 2007 Petar Petrovic 2007 Milica Antonic 2008 Ivana Ivanovic 2009 Dragana Markovic 2010 </pre>
<pre> Poziv: ./a.out in.txt out.txt  IN.OUT Milijana Maric 2009 </pre>	<pre> OUT.TXT Milijana Maric 2009 </pre>

**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 trazenim 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 studenata 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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.27** 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	Test 2	Test 3
ULAZ: R	ULAZ: E	ULAZ: X
IzLAZ: Dusko Radovic	IzLAZ: Dobrica Eric	IzLAZ: -1

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

Primer 1	Primer 2
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	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 3.28]

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

Primer 1	Primer 2	Primer 3
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	INTERAKCIJA SA PROGRAMOM: Uneti dimenziju niza: 1 Uneti elemente niza: 234 Sortirani niz u rastucem poretku prema broju delilaca 234	INTERAKCIJA SA PROGRAMOM: Uneti dimenziju niza: 0 Uneti elemente niza: Sortirani niz u rastucem poretku prema broju delilaca: delilaca:

[Rešenje 3.29]

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

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

Niske se učitavaju iz 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 3.30]

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

[Rešenje 3.31]

**Zadatak 3.32** Napisati program koji korišćenjem bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova ili poruka ukoliko nema takvog. Potom, sa 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 3.32]

**Zadatak 3.33** Uraditi zadatak 3.12, ali korišćenjem bibliotečke `qsort` funkcije.

[Rešenje 3.33]

**Zadatak 3.34** Napisati program koji sa standardnog ulaza učitava prvo ceo broj  $n$  ( $n \leq 10$ ), a zatim niz  $S$  od  $n$  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 3.34]

**Zadatak 3.35** 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 3.35]

**Zadatak 3.36** Definisati 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
```



**Zadatak 3.37** Za zadanu celobrojnu matricu dimenzije  $n \times m$  napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrsti. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu.

*Test 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 3.37]

**Zadatak 3.38** Za zadanu 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.

*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
```

## 3.4 Rešenja

### Rešenje 3.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
7  -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
10 njemu element x. Pretraga se vrši prostom iteracijom kroz niz. Ako
11 se element pronadje funkcija vraca indeks pozicije na kojoj je
12 pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
13 pronadjen u nizu, funkcija vraca -1, kao indikator neuspesne
14 pretrage. */
15 int linearna_pretraga(int a[], int n, int x)
16 {
17     int i;
18     for (i = 0; i < n; i++)
19         if (a[i] == x)
20             return i;
21     return -1;
22 }
23
24 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
25 pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
26 int binarna_pretraga(int a[], int n, int x)
27 {
28     int levi = 0;
29     int desni = n - 1;
30     int srednji;
31     /* Dokle god je indeks levi levo od indeksa desni */
32     while (levi <= desni) {
33         /* Srednji indeks je njihova aritmeticka sredina */
34         srednji = (levi + desni) / 2;
35         /* Ako je element sa sredisnjim indeksom veci od x, tada se x
36 mora nalaziti u levoj polovini niza */
37         if (x < a[srednji])
38             desni = srednji - 1;
39         /* Ako je element sa sredisnjim indeksom manji od x, tada se x
40 mora nalaziti u desnoj polovini niza */
41         else if (x > a[srednji])
42             levi = srednji + 1;
43         else
44             /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
45 x pronadjen na poziciji srednji */
46             return srednji;
47     }
48     /* Ako element x nije pronadjen, vraca se -1 */
49     return -1;
50 }
51
```

```

53  /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
    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 */
99              return srednji;
100      }
101      /* U slucaju neuspesne pretrage vraca se -1 */
102      return -1;
103  }

```

```

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);
    /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
143 linearnu pretragu */
    clock_gettime(CLOCK_REALTIME, &time2);
145 if (i == -1)
        printf("Element nije u nizu\n");
147 else
        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);

```

```

155 clock_gettime(CLOCK_REALTIME, &time4);
156 if (i == -1)
157     printf("Element nije u nizu\n");
158 else
159     printf("Element je u nizu na poziciji %d\n", i);
160
161 /* Interpolaciona pretraga */
162 printf("Interpolaciona pretraga:\n");
163 clock_gettime(CLOCK_REALTIME, &time5);
164 i = interpolaciona_pretraga(a, n, x);
165 clock_gettime(CLOCK_REALTIME, &time6);
166 if (i == -1)
167     printf("Element nije u nizu\n");
168 else
169     printf("Element je u nizu na poziciji %d\n", i);
170
171 /* Podaci o izvršavanju programa bivaju upisani u log fajl */
172 if ((f = fopen("vremena.txt", "a")) == NULL) {
173     fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
174     exit(EXIT_FAILURE);
175 }
176
177 fprintf(f, "Dimenzija niza: %d\n", n);
178 fprintf(f, "\tLinearna: %10ld ns\n",
179         (time2.tv_sec - time1.tv_sec) * 1000000000 +
180         time2.tv_nsec - time1.tv_nsec);
181 fprintf(f, "\tBinarna: %19ld ns\n",
182         (time4.tv_sec - time3.tv_sec) * 1000000000 +
183         time4.tv_nsec - time3.tv_nsec);
184 fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
185         (time6.tv_sec - time5.tv_sec) * 1000000000 +
186         time6.tv_nsec - time5.tv_nsec);
187
188 /* Zatvaranje datoteke */
189 fclose(f);
190
191 return 0;
192 }

```

### Rešenje 3.2

```

#include <stdio.h>
2
#define MAX 1024
4
int lin_pretraga_rek_sufiks(int a[], int n, int x)
6 {
    int tmp;
8     /* Izlaz iz rekurzije */
    if (n <= 0)
10         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;
59      /* Pozicija na kojoj se trazi vrednost x */
60      p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
61      if (a[p] == x)
62          return p;
```

```
64     if (a[p] < x)
65         return interp_pretraga_rek(a, p + 1, d, x);
66     else
67         return interp_pretraga_rek(a, l, p - 1, x);
68 }
69
70 int main()
71 {
72     int a[MAX];
73     int x;
74     int i, indeks;
75
76     /* Ucitavanje trazenog broja */
77     printf("Unesite trazeni broj: ");
78     scanf("%d", &x);
79
80     /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
81        korisnik pritisne CTRL+D za naznaku kraja */
82     i = 0;
83     printf("Unesite sortiran niz elemenata: ");
84     while (scanf("%d", &a[i]) == 1) {
85         i++;
86     }
87
88     /* Linearna pretraga */
89     printf("Linearna pretraga\n");
90     indeks = lin_pretraga_rek_sufiks(a, i, x);
91     if (indeks == -1)
92         printf("Element se ne nalazi u nizu.\n");
93     else
94         printf("Pozicija elementa je %d.\n", indeks);
95
96     /* Binarna pretraga */
97     printf("Binarna pretraga\n");
98     indeks = bin_pretraga_rek(a, 0, i - 1, x);
99     if (indeks == -1)
100         printf("Element se ne nalazi u nizu.\n");
101     else
102         printf("Pozicija elementa je %d.\n", indeks);
103
104     /* Interpolaciona pretraga */
105     printf("Interpolaciona pretraga\n");
106     indeks = interp_pretraga_rek(a, 0, i - 1, x);
107     if (indeks == -1)
108         printf("Element se ne nalazi u nizu.\n");
109     else
110         printf("Pozicija elementa je %d.\n", indeks);
111
112     return 0;
113 }
```

## Rešenje 3.3

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
7
8 /* 0 svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
9
10 typedef struct {
11     /* Indeks mora biti tipa long jer su podaci u datoteci preveliki za
12      int, npr. 20140123 */
13     long indeks;
14     char ime[MAX_DUZINA];
15     char prezime[MAX_DUZINA];
16 } Student;
17
18 /* 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 /* 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. */
22
23 int binarna_pretraga(Student a[], int n, long x)
24 {
25     int levi = 0;
26     int desni = n - 1;
27     int srednji;
28     /* Dokle god je indeks levi levo od indeksa desni */
29     while (levi <= desni) {
30         /* Racuna se srednja pozicija */
31         srednji = (levi + desni) / 2;
32         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
           se trazeni indeks mora nalaziti u levoj polovini niza */
33         if (x < a[srednji].indeks)
34             desni = srednji - 1;
35         /* Ako je pak manji od trazenog, tada se on mora nalaziti u
           desnoj polovini niza */
36         else if (x > a[srednji].indeks)
37             levi = srednji + 1;
38         else
39             /* Ako je jednak trazenom indeksu x, tada je pronadjen student
               sa trazenom indeksom na poziciji srednji */
40             return srednji;
41     }
42     /* Ako nije pronadjen, vraca se -1 */
43     return -1;
44 }
```



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

62 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
64    prosledjuju kao argumenti komandne linije */
int main(int argc, char *argv[])
66 {
    Student dosije[MAX_STUDENATA];
68     FILE *fin = NULL;
    int i;
70     int br_studenata = 0;
    long trazen_indeks = 0;
72     char trazeno_prezime[MAX_DUZINA];
    int bin_pretraga;
74
    /* Provera da li je korisnik prilikom poziva programa prosledio ime
76     datoteke sa informacijama o studentima i opciju pretrage */
    if (argc != 3) {
78         fprintf(stderr,
            "Greska: Program se poziva sa %s ime_datoteke opcija\n",
80             argv[0]);
        exit(EXIT_FAILURE);
82     }

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

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

102    /* Citanje se vrši sve dok postoji red sa informacijama o studentu
```

```

104     */
105     i = 0;
106     while (1) {
107         if (i == MAX_STUDENATA)
108             break;
109         if (fscanf
110             (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
111              dosije[i].prezime) != 3)
112             break;
113         i++;
114     }
115     br_studenata = i;
116
117     /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
118     fclose(fin);
119
120     /* Pretraga po indeksu */
121     if (bin_pretraga) {
122         /* Unos indeksa koji se binarno trazi u nizu */
123         printf("Unesite indeks studenta cije informacije zelite: ");
124         scanf("%ld", &trazen_indeks);
125         i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
126         /* Rezultat binarne pretrage */
127         if (i == -1)
128             printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
129         else
130             printf("Indeks: %ld, Ime i prezime: %s %s\n",
131                    dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132     }
133     /* Pretraga po prezimenu */
134     else {
135         /* Unos prezimena koje se linearno trazi u nizu */
136         printf("Unesite prezime studenta cije informacije zelite: ");
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
#define MAX_STUDENATA 128
6
#define MAX_DUZINA 16

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

14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
16 {
    /* Ako je pozicija elementa na levom kraju veca od pozicije
18    elementa na desnom kraju dela niza koji se pretrazuje, onda se
    zapravo pretrazuje prazan deo niza. U praznom delu niza nema
20    trazenog elementa pa se vraca -1 */
    if (levi > desni)
22        return -1;
    /* Racunanje pozicije srednjeg elementa */
    int srednji = (levi + desni) / 2;
24    /* Da li je srednji bas onaj trazeni */
    if (a[srednji].indeks == x) {
26        return srednji;
    }
    /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
30    poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
    je poznato da je niz sortiran po indeksu u rastucem poretку. */
32    if (x < a[srednji].indeks)
        return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34    /* Inace ga treba traziti u desnoj polovini */
    else
36        return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
}

38
int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
    /* Ako je niz prazan, vraca se -1 */
42    if (n == 0)
        return -1;
    /* Kako se trazi prvi student sa trazenim prezimenom, pocinje se sa
44    prvim studentom u nizu. */
    if (strcmp(a[0].prezime, x) == 0)
46        return 0;
    int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
48    return i >= 0 ? 1 + i : -1;
50 }

52
int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
{
54    /* Ako je niz prazan, vraca se -1 */
    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 }

63
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;

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

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     }

93
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     }

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)
```

```

108         break;
109     if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
110         dosije[i].prezime) != 3)
        break;
112     i++;
113 }
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 cijje informacije zelite: ");
    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 cijje informacije zelite: ");
    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>
  #include <string.h>
3 #include <math.h>
  #include <stdlib.h>
5

```

```
/* Struktura koja opisuje tacku u ravni */
7 typedef struct Tacka {
    float x;
9    float y;
} Tacka;

11
/* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13 pocetka (0,0) */
float rastojanje(Tacka A)
15 {
    return sqrt(A.x * A.x + A.y * A.y);
17 }

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

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

49
/* 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;
    int i;
55     najbliza = t[0];
    for (i = 1; i < n; i++) {
57
```

```
59     if (fabs(t[i].y) < fabs(najbliza.y)) {
60         najbliza = t[i];
61     }
62     return najbliza;
63 }

64 #define MAX 1024

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

71     /* Očekuje se da korisnik prosledi barem ime izvršnog programa i
72        ime datoteke sa tackama */
73     if (argc < 2) {
74         fprintf(stderr,
75             "koriscenje programa: %s ime_datoteke\n", argv[0]);
76         return EXIT_FAILURE;
77     }

78     /* Otvaranje datoteke za citanje */
79     ulaz = fopen(argv[1], "r");
80     if (ulaz == NULL) {
81         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
82             argv[1]);
83         return EXIT_FAILURE;
84     }

85     /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
86        tackama; i predstavlja indeks tekuće tacke */
87     i = 0;
88     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
89         i++;
90     }
91     n = i;

92     /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
93        dodatnih argumenata */
94     if (argc == 2)
95         /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
96         najbliza = najbliza_koordinatnom(tacke, n);
97     /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
98        pitanju opcija -x */
99     else if (strcmp(argv[2], "-x") == 0)
100         /* Racuna se rastojanje u odnosu na x osu */
101         najbliza = najbliza_x_osi(tacke, n);
102     /* Ako je u pitanju opcija -y */
103     else if (strcmp(argv[2], "-y") == 0)
```

```
111     /* Racuna se rastojanje u odnosu na y osu */
    najbliza = najbliza_y_osi(tacke, n);
113 else {
    /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
    korisnika i prekida se izvršavanje programa */
115     fprintf(stderr, "Pogresna opcija\n");
    return EXIT_FAILURE;
117 }

119 /* Stampanje koordinata trazene tacke */
    printf("%g %g\n", najbliza.x, najbliza.y);
121
    /* Zatvaranje datoteke */
123     fclose(ulaz);

125     return 0;
}
```

### Rešenje 3.6

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

4 /* Tacnost */
#define EPS 0.001

6
8 int main()
9 {
10     double l, d, s;

12     /* Kako je u pitanju interval [0, 2] leva granica je 0, a desna 2
    */
    l = 0;
    d = 2;

14     /* Sve dok se ne pronadje trazena vrednost argumenta */
16     while (1) {
        /* Polovi se interval */
18         s = (l + d) / 2;
        /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
        tacnosti, prekida se pretraga */
20         if (fabs(cos(s)) < EPS) {
            break;
22         }
        /* Ako je nula u levom delu intervala, nastavlja se pretraga na
        [l, s] */
24         if (cos(l) * cos(s) < 0)
            d = s;
        else
26             /* Inace, na intervalu [s, d] */
            l = s;
28
30     }
```



```
    }  
32  
    /* Stampanje 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;  
10    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  
           prethodnik manji ili jednak nuli, pretraga je završena */  
16        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  
           polovina niza */  
20        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>
2  #include <stdlib.h>
3
4  #define MAX 256
5
6  int prvi_manji_od_nule(int niz[], int n)
7  {
8      /* Granice pretrage */
9      int l = 0, d = n - 1;
10     int s;
11     /* Sve dok je leva manja od desne granice */
12     while (l <= d) {
13         /* Racuna se sredisnja pozicija */
14         s = (l + d) / 2;
15         /* Ako je broj na toj poziciji manji od nule, a eventalni njegov
16            prethodnik veci ili jednak nuli, pretraga se završava */
17         if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
18             return s;
19         /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
20            niza */
21         if (niz[s] >= 0)
22             l = s + 1;
23         /* A inace leva */
24         else
25             d = s - 1;
26     }
27     return -1;
28 }
29
30 int main()
31 {
32     int niz[MAX];
33     int n = 0;
34
35     /* Unos niza */
36     while (scanf("%d", &niz[n]) == 1)
37         n++;
38
39     /* Stapanje rezultata */
40     printf("%d\n", prvi_manji_od_nule(niz, n));
41
42     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) {
46         fprintf(stderr, "Logaritam od nule nije definisan\n");
47         exit(EXIT_FAILURE);
48     }
49
50     /* Ispis povratnih vrednosti funkcija */
```

```

51     printf("%u %u\n", logaritam_a(x), logaritam_b(x));
53     return 0;
}

```

### Rešenje 3.11

```

1  #include<stdio.h>
   #define MAX 256
3
   /* Iterativna verzija funkcije koja sortira niz celih brojeva,
   primenom algoritma Selection Sort */
5  void selectionSort(int a[], int n)
7  {
   int i, j;
   int min;
   int pom;
11
   /* U svakoj iteraciji ove petlje se pronalazi 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++) {
17       /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
       najmanji od elemenata a[i],...,a[n-1]. */
19       min = i;
       for (j = i + 1; j < n; j++)
21           if (a[j] < a[min])
               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];
           a[i] = a[min];
           a[min] = pom;
27       }
29   }
31 }

33 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
   sortiranom nizu celih brojeva */
35 int najmanje_rastojanje(int a[], int n)
   {
37     int i, min;
     min = a[1] - a[0];
39     for (i = 2; i < n; i++)
         if (a[i] - a[i - 1] < min)
41             min = a[i] - a[i - 1];
     return min;
43 }

```

```
45 int main()
46 {
47     int i, a[MAX];
48
49     /* Ucitavaju se elementi niza sve do kraja ulaza */
50     i = 0;
51     while (scanf("%d", &a[i]) != EOF)
52         i++;
53
54     /* Sortiranje */
55     selectionSort(a, i);
56
57     /* Ispis rezultata */
58     printf("%d\n", najmanje_rastojanje(a, i));
59
60     return 0;
61 }
```

### Rešenje 3.12

```
1 #include<stdio.h>
2 #include<string.h>
3
4 #define MAX_DIM 128
5
6 /* Funkcija za sortiranje niza karaktera */
7 void selectionSort(char s[])
8 {
9     int i, j, min;
10    char pom;
11    for (i = 0; s[i] != '\0'; i++) {
12        min = i;
13        for (j = i + 1; s[j] != '\0'; j++)
14            if (s[j] < s[min])
15                min = j;
16        if (min != i) {
17            pom = s[i];
18            s[i] = s[min];
19            s[min] = pom;
20        }
21    }
22 }
23
24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
25 int anagrami(char s[], char t[])
26 {
27     int i;
28
29     /* Ako dve niske imaju razlicit broj karaktera onda one nisu
```

```

30     anagrami */
31     if (strlen(s) != strlen(t))
32         return 0;

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

38     /* Dve sortirane niske su anagrami ako i samo ako su jednake */
39     for (i = 0; s[i] != '\0'; i++)
40         if (s[i] != t[i])
41             return 0;
42     return 1;
43 }

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

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

54     /* Poziv funkcije */
55     if (anagrami(s, t))
56         printf("jesu\n");
57     else
58         printf("nisu\n");

60     return 0;
61 }

```

### Rešenje 3.13

```

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

3
4  /* Funkcija za sortiranje niza */
5  void selectionSort(int s[], int n)
6  {
7      int i, j, min;
8      char pom;
9      for (i = 0; i < n; i++) {
10         min = i;
11         for (j = i + 1; j < n; j++)
12             if (s[j] < s[min])
13                 min = j;
14         if (min != i) {
15             pom = s[i];

```

```

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

```

### Rešenje 3.14

```

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

```

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

21 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
22    u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
23    poretku */
24 int binarna_pretraga(int a[], int n, int x)
25 {
26     int levi = 0, desni = n - 1, srednji;
27
28     while (levi <= desni) {
29         srednji = (levi + desni) / 2;
30         if (a[srednji] == x)
31             return 1;
32         else if (a[srednji] > x)
33             desni = srednji - 1;
34         else if (a[srednji] < x)
35             levi = srednji + 1;
36     }
37     return 0;
38 }

39
40 int main()
41 {
42     int a[MAX_DIM], n = 0, zbir, i;
43
44     /* Ucitava se trazeni zbir */
45     printf("Unesite trazeni zbir: ");
46     scanf("%d", &zbir);
47
48     /* Ucitavaju se elementi niza sve do kraja ulaza */
49     i = 0;
50     printf("Unesite elemente niza: ");
51     while (scanf("%d", &a[i]) != EOF)
52         i++;
53     n = i;
54
55     /* Sortira se niz */
56     selectionSort(a, n);
```



```

57     for (i = 0; i < n; i++)
59         /* Za i-ti element niza binarno se pretrazuje da li se u ostatku
           niza nalazi element koji sabran sa njim ima ucitanu vrednost
           zbira */
61         if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
63             printf("da\n");
           return 0;
65         }
       printf("ne\n");
67
       return 0;
69 }

```

### Rešenje 3.15

```

/* Datoteka sort.h */
2 #ifndef __SORT_H__
  #define __SORT_H__ 1
4
/* Selection sort */
6 void selectionsort(int a[], int n);
/* Insertion sort */
8 void insertionsort(int a[], int n);
/* Bubble sort */
10 void bubblesort(int a[], int n);
/* Shell sort */
12 void shellsort(int a[], int n);
/* Merge sort */
14 void mergesort(int a[], int l, int r);
/* Quick sort */
16 void quicksort(int a[], int l, int r);
18 #endif

```

```

/* Datoteka sort.c */
2
#include "sort.h"
4
#define MAX 1000000
6
/* Funkcija sortira niz celih brojeva metodom sortiranja izborom.
   Ideja algoritma je sledeca: U svakoj iteraciji pronalazi se
   najmanji element i premesta se na pocetak niza. Dakle, u prvoj
10 iteraciji, pronalazi se najmanji element, i dovodi na nulto mesto
   u nizu. U i-toj iteraciji najmanjih i elemenata su vec na svojim
12 pozicijama, pa se od i+1 do n-1 elementa trazi najmanji, koji se
   dovodi na i+1 poziciju. */
14 void selectionsort(int a[], int n)
   {

```

```

16  int i, j;
17  int min;
18  int pom;

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

32      /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
33         su (i) i min razliciti, inace je nepotrebno. */
34      if (min != i) {
35          pom = a[i];
36          a[i] = a[min];
37          a[min] = pom;
38      }
39  }
40 }

42 /* Funkcija sortira niz celih brojeva metodom sortiranja umetanjem.
43    Ideja algoritma je sledeca: neka je na pocetku i-te iteracije niz
44    prvih i elemenata (a[0],a[1],...,a[i-1]) sortirano. U i-toj
45    iteraciji treba element a[i] umetnuti na pravu poziciju medju
46    prvih i elemenata tako da se dobije niz duzine i+1 koji je
47    sortiran. Ovo se radi tako sto se i-ti element najpre uporedi sa
48    njegovim prvim levim susedom (a[i-1]). Ako je a[i] vece, tada je
49    on vec na pravom mestu, i niz a[0],a[1],...,a[i] je sortiran, pa
50    se moze preci na sledecu iteraciju. Ako je a[i-1] vece, tada se
51    zamenjuju a[i] i a[i-1], a zatim se proverava da li je potrebno
52    dalje potiskivanje elementa u levo, poredeci ga sa njegovim novim
53    levim susedom. Ovim uzastopnim premestanjem se a[i] umece na pravo
54    mesto u nizu. */
55 void insertionsort(int a[], int n)
56 {
57     int i, j;

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

62         /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
63            potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...,a[i]
64            bude sortiran. Indeks j je trenutna pozicija na kojoj se
65            element koji se umece nalazi. Petlja se zavrшава ili kada

```

```

        element dodje do levog kraja (j==0) ili kada se naidje na
68     element a[j-1] koji je manji od a[j]. */
    for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
70         int temp = a[j];
        a[j] = a[j - 1];
72         a[j - 1] = temp;
    }
74 }
}

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

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

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

102                 /* Promenljiva ind registruje da je bilo premestanja. Samo u
104 tom slucaju ima smisla ici na sledecu iteraciju, jer ako
nije bilo premestanja, znaci da su svi elementi vec u
106 dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
108 ispravno, ali manje efikasano, jer bi se cesto nepotrebno
vrsila mnoga uporedjivanja, kada je vec jasno da je
110 sortiranje završeno. */
                ind = 1;
112             }
    }
114 }

116 /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
sastoji u tome da se kroz algoritam umetanja prolazi vise puta; u
118 prvom prolazu, umesto koraka 1 uzima se neki korak h koji je manji

```

```

120     od n (sto omogućuje razmenu udaljenih elemenata) i tako se dobija
122     h-sortiran niz, tj. niz u kome su elementi na rastojanju h
124     sortirani, mada susedni elementi to ne moraju biti. U drugom
126     prolazu kroz isti algoritam sprovodi se isti postupak ali za manji
128     korak h. Sa prolazima se nastavlja sve do koraka h = 1, u kome se
130     dobija potpuno sortirani niz. Izbor pocetne vrednosti za h, i
132     nacina njegovog smanjivanja menja u nekim slucajevima brzinu
134     algoritma, ali bilo koja vrednost ce rezultovati ispravnim
136     sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
138     vrednost 1. */
140 void shellsort(int a[], int n)
142 {
144     int h = n / 2, i, j;
146     while (h > 0) {
148         /* Insertion sort sa korakom h */
150         for (i = h; i < n; i++) {
152             int temp = a[i];
154             j = i;
156             while (j >= h && a[j - h] > temp) {
158                 a[j] = a[j - h];
160                 j -= h;
162             }
164             a[j] = temp;
166         }
168         h = h / 2;
170     }
172 }

174 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem. Sortiranje
176 se vrši od elementa na poziciji l do onog na poziciji d. Na
178 pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a d je
180 jednako poslednjem validnom indeksu u nizu. Funkcija niz podeli na
182 dve polovine, levu i desnu, koje zatim rekurzivno sortira. Od ova
184 dva sortirana podniza, sortiran niz se dobija ucesljavanjem, tj.
186 istovremenim prolaskom kroz oba niza i izborom trenutnog manjeg
188 elementa koji se smesta u pomocni niz. Na kraju algoritma,
190 sortirani elementi su u pomocnom nizu, koji se kopira u originalni
192 niz. */
194 void mergesort(int a[], int l, int d)
196 {
198     int s;
199     static int b[MAX];          /* Pomocni niz */
200     int i, j, k;
202     /* Izlaz iz rekurziije */
203     if (l >= d)
204         return;
206     /* Odredjivanje sredisnjeg indeksa */
207     s = (l + d) / 2;
209     /* Rekurzivni pozivi */

```

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

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

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

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

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

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

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

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

```

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

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

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

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

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

```

```

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

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

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

```

```

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

```

```

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

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

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

```



```

126  /*****
128  Ovaj deo je 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 3.16

```

#include <stdio.h>
#define MAX_DIM 256

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

    /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog
       od njih */
    while (i < dim1 && j < dim2) {
        if (niz1[i] < niz2[j])
            niz3[k++] = niz1[i++];
        else
            niz3[k++] = niz2[j++];
    }
    /* Ostatak prvog niza prepisujemo u treci */
    while (i < dim1)
        niz3[k++] = niz1[i++];

    /* Ostatak drugog niza prepisujemo u treci */
    while (j < dim2)
        niz3[k++] = niz2[j++];
    return dim1 + dim2;
}

int main()
{

```

```

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        */
38     printf("Unesite elemente prvog niza: ");
39     while (1) {
40         scanf("%d", &niz1[i]);
41         if (niz1[i] == 0)
42             break;
43         i++;
44     }
45     printf("Unesite elemente drugog niza: ");
46     while (1) {
47         scanf("%d", &niz2[j]);
48         if (niz2[j] == 0)
49             break;
50         j++;
51     }
52
53     /* Poziv trazene funkcije */
54     dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);
55
56     /* Ispis niza */
57     for (k = 0; k < dim3; k++)
58         printf("%d ", niz3[k]);
59     printf("\n");
60
61     return 0;
62 }

```

### Rešenje 3.17

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[])
6 {
7     FILE *fin1 = NULL, *fin2 = NULL;
8     FILE *fout = NULL;
9     char ime1[11], ime2[11];
10    char prezime1[16], prezime2[16];
11    int kraj1 = 0, kraj2 = 0;
12
13    /* Ako nema dovoljno argumenata komandne linije */
14    if (argc < 3) {
15        fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0])
16        ;
17        exit(EXIT_FAILURE);
18    }
19 }

```

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

```

70     druge datoteke, onda ima jos studenata u prvoj datoteci, koje
       treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
72     */
       while (!kraj1) {
           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);
89     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

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

21 /* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
       pocetka */
23 void sortiraj_po_rastojanju(Tacka t[], int n)

```

```
{
25     int min, i, j;
    Tacka tmp;
27
    for (i = 0; i < n - 1; i++) {
29         min = i;
        for (j = i + 1; j < n; j++) {
31             if (rastojanje(t[j]) < rastojanje(t[min])) {
                min = j;
33             }
        }
35         if (min != i) {
            tmp = t[i];
37             t[i] = t[min];
            t[min] = tmp;
39         }
    }
41 }

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

63
65 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
void sortiraj_po_y(Tacka t[], int n)
67 {
    int min, i, j;
    Tacka tmp;
69
    for (i = 0; i < n - 1; i++) {
71         min = i;
        for (j = i + 1; j < n; j++) {
73             if (abs(t[j].y) < abs(t[min].y)) {
                min = j;
75             }
        }
    }
}
```

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

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

```

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

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

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

```

### Rešenje 3.19

```

#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;

```

```
20   Gradjanin pom;
21
22   for (i = 0; i < n - 1; i++) {
23       /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
24          najmanji od elemenata a[i].ime,...,a[n-1].ime. */
25       min = i;
26       for (j = i + 1; j < n; j++)
27           if (strcmp(a[j].ime, a[min].ime) < 0)
28               min = j;
29       /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
30          su (i) i min razliciti, inace je nepotrebno. */
31       if (min != i) {
32           pom = a[i];
33           a[i] = a[min];
34           a[min] = pom;
35       }
36   }
37
38   /* Funkcija sortira niz gradjana rastuce po prezimenima */
39   void sort_prezime(Gradjanin a[], int n)
40   {
41       int i, j;
42       int min;
43       Gradjanin pom;
44
45       for (i = 0; i < n - 1; i++) {
46           /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
47              najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
48           min = i;
49           for (j = i + 1; j < n; j++)
50               if (strcmp(a[j].prezime, a[min].prezime) < 0)
51                   min = j;
52           /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
53              su (i) i min razliciti, inace je nepotrebno. */
54           if (min != i) {
55               pom = a[i];
56               a[i] = a[min];
57               a[min] = pom;
58           }
59       }
60   }
61
62   /* Pretraga niza Gradjana */
63   int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64   {
65       int i;
66       for (i = 0; i < n; i++)
67           if (strcmp(a[i].ime, x->ime) == 0
68               && strcmp(a[i].prezime, x->prezime) == 0)
69               return i;
70       return -1;
71   }
```



```

72 }
74 int main()
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;
78     int i, n;
79     FILE *fp = NULL;
80
81     /* Otvaranje datoteke */
82     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
83         fprintf(stderr,
84             "Neuspesno otvaranje datoteke biracki-spisak.txt.\n");
85         exit(EXIT_FAILURE);
86     }
87
88     /* Citanje sadrzaja */
89     for (i = 0;
90         fscanf(fp, "%s %s", spisak1[i].ime,
91             spisak1[i].prezime) != EOF; i++)
92         spisak2[i] = spisak1[i];
93     n = i;
94
95     /* Zatvaranje datoteke */
96     fclose(fp);
97
98     sort_ime(spisak1, n);
99
100     /******
101     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
102     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
103
104     printf("Biracki spisak [uredjen prema imenima]:\n");
105     for(i=0; i<n; i++)
106         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107     *****/
108
109     sort_prezime(spisak2, n);
110
111     /******
112     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
113     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
114
115     printf("Biracki spisak [uredjen prema prezimenima]:\n");
116     for(i=0; i<n; i++)
117         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
118     *****/
119
120     /* Linearno pretrazivanje nizova */
121     for (i = 0; i < n; i++)
122         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))

```

```

124         isti_rbr++;

126     /* Alternativno (efikasnije) resenje */
127     /*****
128         for(i=0; i<n ;i++)
129             if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
130                 strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
131                 isti_rbr++;
132     *****/

132     /* Ispis rezultata */
133     printf("%d\n", isti_rbr);

136     exit(EXIT_SUCCESS);
}

```

### Rešenje 3.21

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>

5  #define MAX_BR_RECII 128
6  #define MAX_DUZINA_RECII 32

7  /* Funkcija koja izracunava broj suglasnika u reci */
9  int broj_suglasnika(char s[])
10 {
11     char c;
12     int i;
13     int suglasnici = 0;
14     /* Prolaz karakter po karakter kroz zadatu nisku */
15     for (i = 0; s[i]; i++) {
16         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
17            pokriven slucaj i malih i velikih suglasnika. */
18         if (isalpha(s[i])) {
19             c = toupper(s[i]);
20             /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika.
21            */
22             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
23                 suglasnici++;
24         }
25     }
26     /* Vraca se izracunata vrednost */
27     return suglasnici;
28 }

29 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
30    duzini reci se mora proslediti zbog pravilnog upravljanja
31    memorijom */
32 void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)

```

```

33 {
34     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35         duzina_j, duzina_min;
36     char tmp[MAX_DUZINA_RECII];
37     for (i = 0; i < n - 1; i++) {
38         min = i;
39         for (j = i; j < n; j++) {
40             /* Prvo se upoređuje broj suglasnika */
41             broj_suglasnika_j = broj_suglasnika(reci[j]);
42             broj_suglasnika_min = broj_suglasnika(reci[min]);
43             if (broj_suglasnika_j < broj_suglasnika_min)
44                 min = j;
45             else if (broj_suglasnika_j == broj_suglasnika_min) {
46                 /* Zatim, recima koje imaju isti broj suglasnika upoređuju
47                    se duzine */
48                 duzina_j = strlen(reci[j]);
49                 duzina_min = strlen(reci[min]);
50
51                 if (duzina_j < duzina_min)
52                     min = j;
53                 else
54                     /* Ako reci imaju i isti broj suglasnika i iste duzine,
55                        upoređuju se leksikografski */
56                     if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
57                         min = j;
58             }
59         }
60         if (min != i) {
61             strcpy(tmp, reci[min]);
62             strcpy(reci[min], reci[i]);
63             strcpy(reci[i], tmp);
64         }
65     }
66 }
67
68 int main()
69 {
70     FILE *ulaz;
71     int i = 0, n;
72
73     /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
74        a drugi maksimalnu duzinu pojedinačne reci */
75     char reci[MAX_BR_RECII][MAX_DUZINA_RECII];
76
77     /* Otvaranje datoteke niske.txt za citanje */
78     ulaz = fopen("niske.txt", "r");
79     if (ulaz == NULL) {
80         fprintf(stderr,
81             "Greska prilikom otvaranja datoteke niske.txt!\n");
82         return 0;
83     }

```

```

85  /* Sve dok se moze procitati sledeca rec */
    while (fscanf(ulaz, "%s", reci[i]) != EOF) {
87      /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
         prekida se ucitavanje */
89      if (i == MAX_BR_REC)
          break;
91      /* Priprema brojaca za narednu iteraciju */
      i++;
93  }

95  /* n je duzina niza reci i predstavlja poslednju vrednost
     koriscenog brojaca */
97  n = i;
  /* Poziv funkcije za sortiranje reci */
99  sortiraj_reci(reci, n);

101 /* Ispis sortiranog niza reci */
    for (i = 0; i < n; i++) {
103         printf("%s ", reci[i]);
    }
105     printf("\n");

107 /* Zatvaranje datoteke */
    fclose(ulaz);

109     return 0;
111 }

```

### Rešenje 3.22

```

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

4

  #define MAX_ARTIKALA 100000

6

  /* Struktura koja predstavlja jedan artikal */
8  typedef struct art {
    long kod;
10     char naziv[20];
    char proizvođač[20];
12     float cena;
  } Artikal;

14

  /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
     traženim bar kodom */
16  int binarna_pretraga(Artikal a[], int n, long x)
18  {
    int levi = 0;
20     int desni = n - 1;

```

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

42  /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44  void selection_sort(Artikal a[], int n)
  {
46    int i, j;
    int min;
48    Artikal pom;

50    for (i = 0; i < n - 1; i++) {
      min = i;
52      for (j = i + 1; j < n; j++)
        if (a[j].kod < a[min].kod)
54          min = j;
      if (min != i) {
56        pom = a[i];
        a[i] = a[min];
58        a[min] = pom;
      }
60    }
  }

62  int main()
64  {
    Artikal asortiman[MAX_ARTIKALA];
66    long kod;
    int i, n;
68    float racun;

70    FILE *fp = NULL;

72    /* Otvaranje datoteke */
    if ((fp = fopen("artikli.txt", "r")) == NULL) {
```

```

74     fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
       exit(EXIT_FAILURE);
76 }

78 /* Ucitavanje artikala */
       i = 0;
80     while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
                    &asortiman[i].naziv, &asortiman[i].proizvodjac,
82                    &asortiman[i].cena) == 4)
           i++;
84
       /* Zatvaranje datoteke */
86     fclose(fp);

88     n = i;

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

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

114    kod = 0;
       while (1) {
116         printf("-----\n");
         printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
118         printf("- Za nov racun unesite kod artikla!\n");
         /* Unos bar koda provog artikla sledeceg kupca */
120         if (scanf("%ld", &kod) == EOF)
             break;
122         /* Trenutni racun novog kupca */
         racun = 0;
124         /* Za sve artikle trenutnog kupca */

```

```

126 while (1) {
    /* Vrsi se njihov pronalazak u nizu */
128     if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
        printf("\tGRESKA: Ne postoji proizvod sa traženim kodom!\n");
    } else {
130         printf("\tTrazili ste:\t%s %s %12.2f\n",
            asortiman[i].naziv, asortiman[i].proizvodjac,
132             asortiman[i].cena);
        /* I dodavanje na ukupan racun */
134         racun += asortiman[i].cena;
    }
136     /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
        nema vise artikla */
138     printf("Unesite kod artikla [ili 0 za prekid]: \t");
    scanf("%ld", &kod);
140     if (kod == 0)
        break;
142 }
    /* Stampanje ukupnog racuna trenutnog kupca */
144     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
}

146 printf("Kraj rada kase!\n");
148 exit(EXIT_SUCCESS);
150 }

```

### Rešenje 3.23

```

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

5  #define MAX 500

7  /* Struktura sa svim informacijama o pojedinacnom studentu */
    typedef struct {
9      char ime[20];
        char prezime[25];
11     int prisustvo;
        int zadaci;
13 } Student;

15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
    rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
    {
19     int i, j;
        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;
28
29         if (min != i) {
30             pom = niz[min];
31             niz[min] = niz[i];
32             niz[i] = pom;
33         }
34     }
35 }
36
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 }
60
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, "Neuspješno otvaranje datoteke aktivnost.txt.\n");
101         exit(EXIT_FAILURE);
102     }
103
104     /* Ucitavanje sadržaja */
105     for (i = 0;
106         fscanf(fp, "%s%s%d%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, "Neuspješno otvaranje datoteke dat1.txt.\n");
119         exit(EXIT_FAILURE);
120     }
121     /* Upis niza u datoteku */
122     fprintf
123     (fp, "Studenti sortirani po imenu leksikografski rastuće:\n");
124     for (i = 0; i < br_studenata; i++)
125         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
126               praktikum[i].prezime, praktikum[i].prisustvo,
127               praktikum[i].zadaci);
```

```

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

129

131  /* Kreiranje drugog spiska studenata po drugom kriterijumu */
    sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
133  if ((fp = fopen("dat2.txt", "w")) == NULL) {
        fprintf(stderr, "Neuspješno otvaranje datoteke dat2.txt.\n");
135  exit(EXIT_FAILURE);
    }

137  /* Upis niza u datoteku */
    fprintf(fp, "Studenti sortirani po broju zadataka opadajuće,\n");
139  fprintf(fp, "pa po dužini imena raste:\n");
    for (i = 0; i < br_studenata; i++)
141        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
                    praktikum[i].prezime, praktikum[i].prisustvo,
143                    praktikum[i].zadaci);
    /* Zatvaranje datoteke */
145  fclose(fp);

147  /* Kreiranje trećeg spiska studenata po trećem kriterijumu */
    sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum, br_studenata);
149  /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat3.txt", "w")) == NULL) {
151        fprintf(stderr, "Neuspješno otvaranje datoteke dat3.txt.\n");
        exit(EXIT_FAILURE);
153    }
    /* Upis niza u datoteku */
155  fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
    fprintf(fp, "pa po broju zadataka,\n");
157  fprintf(fp, "pa po prezimenima leksikografski opadajuće:\n");
    for (i = 0; i < br_studenata; i++)
159        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
                    praktikum[i].prezime, praktikum[i].prisustvo,
161                    praktikum[i].zadaci);
    /* Zatvaranje datoteke */
163  fclose(fp);

165  return 0;
}

```

### Rešenje 3.24

```

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

4
  #define KORAK 10

6
  /* Struktura koja opisuje jednu pesmu */
8  typedef struct {

```

```
10  char *izvodjac;
    char *naslov;
    int broj_gledanja;
12 } Pesma;

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

20     return p2->broj_gledanja - p1->broj_gledanja;
22 }

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

30     return strcmp(p1->naslov, p2->naslov);
32 }

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

40     return strcmp(p1->izvodjac, p2->izvodjac);
42 }

44 int main(int argc, char *argv[])
    {
46     FILE *ulaz;
        Pesma *pesme;                                /* Pokazivac na deo memorije za
                                                         cuvanje pesama */
48     int alocirano_za_pesme;                          /* Broj mesta alociranih za pesme */
        int i;                                         /* Redni broj pesme cije se
                                                         informacije citaju */
50     int n;                                           /* Ukupan broj pesama */
        int j, k;
        char c;
52     int alocirano;                                /* Broj mesta alociranih za propratne
                                                         informacije o pesmama */
54     int broj_gledanja;

56     /* Priprema datoteke za citanje */
60     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
```

```
62     if (ulaz == NULL) {
63         printf("Greska pri otvaranju ulazne datoteke!\n");
64         return 0;
65     }
66
67     /* Citanje informacija o pesmama */
68     pesme = NULL;
69     alocirano_za_pesme = 0;
70     i = 0;
71
72     while (1) {
73
74         /* Proverava da li je dostignut kraj datoteke */
75         c = fgetc(ulaz);
76         if (c == EOF) {
77             /* Nema vise sadrzaja za citanje */
78             break;
79         } else {
80             /* Inace, vracamo procitani karakter nazad */
81             ungetc(c, ulaz);
82         }
83
84         /* Provera da li postoji dovoljno memorije za citanje nove pesme */
85         if (alocirano_za_pesme == i) {
86
87             /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
88              * novih KORAK mesta */
89             alocirano_za_pesme += KORAK;
90             pesme =
91                 (Pesma *) realloc(pesme,
92                                   alocirano_za_pesme * sizeof(Pesma));
93
94             /* Proverava da li je nova memorija uspesno realocirana */
95             if (pesme == NULL) {
96                 /* Ako nije ispisuje se obavestenje */
97                 printf("Problem sa alokacijom memorije!\n");
98                 /* I oslobadja sva memorija zauzeta do ovog koraka */
99                 for (k = 0; k < i; k++) {
100                     free(pesme[k].izvodjac);
101                     free(pesme[k].naslov);
102                 }
103                 free(pesme);
104                 return 0;
105             }
106
107             /* Ako jeste, nastavlja se sa citanjem pesama ... */
108             /* Cita se ime izvodjaca */
109             j = 0;
110             /* Pozicija na koju treba smestiti
111              * procitani karakter */
112             alocirano = 0;
113             /* Broj alociranih mesta */
```

```
112 pesme[i].izvodjac = NULL; /* Memorija za smestanje procitanih
114 karaktera */
116 /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
118 izvodjaca) citaju se karakteri iz datoteke */
120 while ((c = fgetc(ulaz)) != ' ') {
122     /* Proverav da li postoji dovoljno memorije za smestanje
124     procitanog karaktera */
126     if (j == alocirano) {
128         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
130         se novih KORAK mesta */
132         alocirano += KORAK;
134         pesme[i].izvodjac =
136             (char *) realloc(pesme[i].izvodjac,
138                             alocirano * sizeof(char));
140
142         /* Provera da li je nova alokacija uspesna */
144         if (pesme[i].izvodjac == NULL) {
146             /* Ako nije oslobadja se sva memorija zauzeta do ovog
148             koraka */
150             for (k = 0; k < i; k++) {
152                 free(pesme[k].izvodjac);
154                 free(pesme[k].naslov);
156             }
158             free(pesme);
160             /* I prekida sa izvršavanjem programa */
162             return 0;
164         }
166     }
168     /* Ako postoji dovoljno memorije, smestamo procitani karakter
170     */
172     pesme[i].izvodjac[j] = c;
174     j++;
176     /* I nastavlja se sa citanjem */
178 }
180
182 /* Upis terminirajuće nule na kraj reci */
184 pesme[i].izvodjac[j] = '\\0';
186
188 /* Preskace se karakter - */
190 fgetc(ulaz);
192
194 /* Preskace se razmak */
196 fgetc(ulaz);
198
200 /* Cita se naslov pesme */
202 j = 0; /* Pozicija na koju treba smestiti
204         procitani karakter */
206 alocirano = 0; /* Broj alociranih mesta */
208 pesme[i].naslov = NULL; /* Memorija za smestanje procitanih
```

```

164                                     karaktera */
166 /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
167    karakteri iz datoteke */
168 while ((c = fgetc(ulaz)) != ',') {
169     /* Provera da li postoji dovoljno memorije za smestanje
170        procitanog karaktera */
171     if (j == alocirano) {
172         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
173            se novih KORAK mesta */
174         alocirano += KORAK;
175         pesme[i].naslov =
176             (char *) realloc(pesme[i].naslov,
177                             alocirano * sizeof(char));
178
179         /* Provera da li je nova alokacija uspesna */
180         if (pesme[i].naslov == NULL) {
181             /* Ako nije, oslobadja se sva memorija zauzeta do ovog
182                koraka */
183             for (k = 0; k < i; k++) {
184                 free(pesme[k].izvodjac);
185                 free(pesme[k].naslov);
186             }
187             free(pesme[i].izvodjac);
188             free(pesme);
189
190             /* I prekida izvršavanje programa */
191             return 0;
192         }
193     }
194     /* Ako postoji dovoljno memorije, smesta se procitani karakter
195     */
196     pesme[i].naslov[j] = c;
197     j++;
198     /* I nastavlja dalje sa citanjem */
199 }
200 /* Upisuje se terminirajuca nula na kraj reci */
201 pesme[i].naslov[j] = '\0';
202
203 /* Preskace se razmak */
204 fgetc(ulaz);
205
206 /* Cita se broj gledanja */
207 broj_gledanja = 0;
208
209 /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
210    datoteke */
211 while ((c = fgetc(ulaz)) != '\n') {
212     broj_gledanja = broj_gledanja * 10 + (c - '0');
213 }
214 pesme[i].broj_gledanja = broj_gledanja;

```

```

214     /* Prelazi se na citanje sledece pesme */
215     i++;
216 }

218 /* Informacija o broju procitanih pesama */
219 n = i;
220 /* Zatvaranje nepotrebne datoteke */
221 fclose(ulaz);

222 /* Analiza argumenta komandne linije */
223 if (argc == 1) {
224     /* Nema dodatnih opcija => sortiranje po broju gledanja */
225     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
226 } else {
227     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
228         /* Sortiranje po naslovu */
229         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
230     } else {
231         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
232             /* Sortiranje po izvodjacu */
233             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
234         } else {
235             printf("Nedozvoljeni argumenti!\n");
236             free(pesme);
237             return 0;
238         }
239     }
240 }

242 /* Ispis rezultata */
243 for (i = 0; i < n; i++) {
244     printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
245           pesme[i].broj_gledanja);
246 }

248 /* Oslobadjanje memorije */
249 for (i = 0; i < n; i++) {
250     free(pesme[i].izvodjac);
251     free(pesme[i].naslov);
252 }
253 free(pesme);

254 return 0;
255 }

```

### Rešenje 3.28

```

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

```

```
5
7
9 #define MAX 100
11
13 /* Funkcija poredjenja dva cela broja */
15 int compare_int(const void *a, const void *b)
17 {
19     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
21        se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
23
25     /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
27        oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */
29
31     int b1 = *((int *) a);
33     int b2 = *((int *) b);
35
37     if (b1 > b2)
39         return 1;
41     else if (b1 < b2)
43         return -1;
45     else
47         return 0;
49 }
51
53 int compare_int_desc(const void *a, const void *b)
55 {
57     /* Za obrnuti poredak treba samo oduzimati a od b */
59     /* return *((int *)b) - *((int *)a); */
61
63     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
65        funkcija */
67     return -compare_int(a, b);
69 }
71
73 int main()
75 {
77     size_t n;
79     int i, x;
81     int a[MAX], *p = NULL;
83
85     /* Unos dimenzije */
87     printf("Uneti dimenziju niza: ");
89     scanf("%ld", &n);
91     if (n > MAX)
93         n = MAX;
95
97     /* Unos elementa niza */
99     printf("Uneti elemente niza:\n");
101     for (i = 0; i < n; i++)
103         scanf("%d", &a[i]);
105
107     /* Sortiranje niza celih brojeva */
109     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");
   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 3.29

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
   #define MAX 100
7
   /* Funkcija racuna broj delilaca broja x */
9  int no_of_deviders(int x)
   {
11     int i;
   int br;
13
   /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
15  if (x < 0)
   x = -x;
17  if (x == 0)
   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;
48     else
49         return 0;
50 }
51
52 int main()
53 {
54     size_t n;
55     int i;
56     int a[MAX];
57
58     /* Unos dimenzije */
59     printf("Uneti dimenziju niza: ");
60     scanf("%ld", &n);
61     if (n > MAX)
62         n = MAX;
63
64     /* Unos elementa niza */
65     printf("Uneti elemente niza:\n");
66     for (i = 0; i < n; i++)
67         scanf("%d", &a[i]);
68
69     /* Sortiranje niza celih brojeva prema broju delilaca */
70     qsort(a, n, sizeof(int), &compare_no_deviders);
```

```

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

```

### Rešenje 3.30

```

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
18   nalazi na adresi koja je za 4 veca od prve reci, a za 4 manja od
19   adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
20   i ona je tipa char*.
21
22   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23   koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
24   reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25   slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
26   nad njima.
27   *****/
28   int poredi_leksikografski(const void *a, const void *b)
29   {
30       return strcmp((char *) a, (char *) b);
31   }
32
33   /* Funkcija slicna prethodnoj, osim sto elemente ne uporeduje
34      leksikografski, vec po duzini */
35   int poredi_duzine(const void *a, const void *b)
36   {
37       return strlen((char *) a) - strlen((char *) b);
38   }
39

```

```

41 int main()
42 {
43     int i;
44     size_t n;
45     FILE *fp = NULL;
46     char niske[MAX_NISKI][MAX_DUZINA];
47     char *p = NULL;
48     char x[MAX_DUZINA];
49
50     /* Otvaranje datoteke */
51     if ((fp = fopen("niske.txt", "r")) == NULL) {
52         fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
53         exit(EXIT_FAILURE);
54     }
55
56     /* Citanje sadrzaja datoteke */
57     for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
58
59     /* Zatvaranje datoteke */
60     fclose(fp);
61     n = i;
62
63     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
64        prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
65        niske po duzini */
66     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);
67
68     printf("Leksikografski sortirane niske:\n");
69     for (i = 0; i < n; i++)
70         printf("%s ", niske[i]);
71     printf("\n");
72
73     /* Unos trazene niske */
74     printf("Uneti trazenu nisku: ");
75     scanf("%s", x);
76
77     /* Binarna pretraga */
78     /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
79        je niz vec sortiran leksikografski. */
80     p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
81                &poredi_leksikografski);
82
83     if (p != NULL)
84         printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
85                p, (p - (char *) niske) / MAX_DUZINA);
86     else
87         printf("Niska nije pronadjena u nizu\n");
88
89     /* Sortiranje po duzini */
90     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
91
92     printf("Niske sortirane po duzini:\n");

```

```

93     for (i = 0; i < n; i++)
        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 3.31

```

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 pokazivaca na karaktere ovog potpisa
11   *  char *niske[3];
12   *  posle alokacije u main-u se moze graficki predstaviti ovako:
13   *
14   *  | X | -----> | a | b | c | \0 |
15   *  | Y | -----> | d | e | \0 |
16   *  | Z | -----> | f | g | h | \0 |
17   *
18   *  Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
19   *  njegov element pokazivac koji pokazuje na alocirane karaktere i-te
20   *  reci. Njegov tip je char*.
21   *
22   *  Kako pokazivaci a i b u sledecoj funkciji sadrže adrese elemenata
23   *  koji trebaju biti upoređeni (recimo adresu od X i adresu od Z), i
24   *  kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
25   *  moraju i kastovati. Da bi se leksikografski uporedili elementi X i
26   *  Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
27   *  u sledecoj funkciji poziva strcmp() nad onim na šta pokazuju a i b,
28   *  kastovani na odgovarajući tip.
29   *
30   *  *****/
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 uporeduje
    leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
    {
41     return strlen(*(char **) a) - strlen(*(char **) b);
    }

43
45 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
    element u nizu sa kojim se poredi, pa njega treba kastovati na
    char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
47 ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
    main funkciji je to x, koji je tipa char*, tako da pokazivac a
49 ovde samo treba kastovati i ne dereferencirati. */
    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;
59     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, "Neupesno 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. Bibliotečkoj funkciji qsort se
88        prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
89        niske po duzini */
    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",
104              *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",
112              *p, p - niske);
    else
115        printf("Niska nije pronadjena u nizu\n");

117    /* Sortiramo po duzini */
    qsort(niske, n, sizeof(char *), &poredi_duzine);

119    printf("Niske sortirane po duzini:\n");
121    for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
123    printf("\n");

125    /* Oslobadjanje zauzete memorije */
    for (i = 0; i < n; i++)
127        free(niske[i]);

129    exit(EXIT_SUCCESS);
}

```

### Rešenje 3.32

```

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

```

```
4 #include <string.h>
5 #include <search.h>
6
7 #define MAX 500
8
9 /* Struktura sa svim informacijama o pojedinacnom studentu */
10 typedef struct {
11     char ime[21];
12     char prezime[21];
13     int bodovi;
14 } Student;
15
16 /* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
17 istim brojem bodova se dodatno sortiraju leksikografski po
18 prezimenu */
19 int poredi1(const void *a, const void *b)
20 {
21     Student *prvi = (Student *) a;
22     Student *drugi = (Student *) b;
23
24     if (prvi->bodovi > drugi->bodovi)
25         return -1;
26     else if (prvi->bodovi < drugi->bodovi)
27         return 1;
28     else
29         /* Ako su jednaki po broju bodova, treba ih uporediti po
30         prezimenu */
31         return strcmp(prvi->prezime, drugi->prezime);
32 }
33
34 /* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
35 Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
36 parametar je element niza ciji se bodovi porede. */
37 int poredi2(const void *a, const void *b)
38 {
39     int bodovi = *(int *) a;
40     Student *s = (Student *) b;
41     return s->bodovi - bodovi;
42 }
43
44 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
45 Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
46 parametar je element niza cije se prezime porede. */
47 int poredi3(const void *a, const void *b)
48 {
49     char *prezime = (char *) a;
50     Student *s = (Student *) b;
51     return strcmp(prezime, s->prezime);
52 }
53
54 int main(int argc, char *argv[])
55 {
```



```
Student kolokvijum[MAX];
56 int i;
size_t br_studenata = 0;
58 Student *nadjen = NULL;
FILE *fp = NULL;
60 int bodovi;
char prezime[21];
62

/* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64 informacija korisniku kako se program koristi i prekida se
izvrsavanje. */
66 if (argc < 2) {
    fprintf(stderr,
68         "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
        argv[0]);
70    exit(EXIT_FAILURE);
}

72

/* Otvaranje datoteke */
74 if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
76    exit(EXIT_FAILURE);
}

78

/* Ucitavanje sadrzaja */
80 for (i = 0;
        fscanf(fp, "%s%s%d", kolokvijum[i].ime,
82             kolokvijum[i].prezime,
                &kolokvijum[i].bodovi) != EOF; i++);
84

/* Zatvaranje datoteke */
86 fclose(fp);
br_studenata = i;
88

/* Sortiranje niza studenata po broju bodova, gde se unutar grupe
90 studenata sa istim brojem bodova sortiranje vrši po prezimenu */
qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
92

printf("Studenti sortirani po broju poena opadajuće, ");
94 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

/* Pretrazivanje studenata po broju bodova se vrši binarnom
100 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
    /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116       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
132    return 0;
}

```

### Rešenje 3.33

```

#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 */
34  printf("Unesite prvu nisku: ");
    scanf("%s", s);
36  printf("Unesite drugu nisku: ");
    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 3.34

```

1  #include <stdio.h>
    #include <string.h>
3  #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
    int main()
15  {
    int i, n;
17  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++)
26         scanf("%s", S[i]);
27
28     /* Sortiranje niza niski */
29     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
30
31     /******
32     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
33     sortiranih niski. Koriscen je samo u fazi testiranja programa.
34
35     printf("Sortirane niske su:\n");
36     for(i = 0; i < n; i++)
37         printf("%s ", S[i]);
38     *****/
39
40     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
41     niza biti jedna do druge */
42     for (i = 0; i < n - 1; i++)
43         if (strcmp(S[i], S[i + 1]) == 0) {
44             printf("ima\n");
45             return 0;
46         }
47
48     printf("nema\n");
49     return 0;
50 }

```

### Rešenje 3.35

```

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

```

```

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

49 int uporedi_bsearch(const void *a, const void *b)
    {
51     /* Nalog studenta koji se trazi */
        char *nalog = (char *) a;
53     /* Kljuc 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 otvarnja 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 */
115     if (strcmp(argv[1], "-p") == 0)
        qsort(niz_studenata, br_studenata, sizeof(Student),
117             &uporedi_poeni);
    /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
119     else if (strcmp(argv[1], "-n") == 0)
        qsort(niz_studenata, br_studenata, sizeof(Student),
121             &uporedi_nalog);

123     /* Sortirani studenti se ispisuju u izlaznu datoteku */
}
```

```

125     for (i = 0; i < br_studenata; i++)
        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;
}

```

### Rešenje 3.37

```

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

4  /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
    standardnog ulaza */
6  void ucitaj_matricu(int **a, int n, int m)
    {
8      printf("Unesite elemente matrice po vrstama:\n");
        int i, j;

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

18    /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
        kolona */

```

```
20 int zbir_vrste(int **a, int v, int m)
21 {
22     int i, zbir = 0;
23
24     for (i = 0; i < m; i++) {
25         zbir += a[v][i];
26     }
27     return zbir;
28 }
29
30 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
31    osnovu zbira koriscenjem selection sort algoritma */
32 void sortiraj_vrste(int **a, int n, int m)
33 {
34     int i, j, min;
35
36     for (i = 0; i < n - 1; i++) {
37         min = i;
38         for (j = i + 1; j < n; j++) {
39             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
40                 min = j;
41             }
42         }
43         if (min != i) {
44             int *tmp;
45             tmp = a[i];
46             a[i] = a[min];
47             a[min] = tmp;
48         }
49     }
50 }
51
52 /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
53    standardni izlaz */
54 void ispis_matricu(int **a, int n, int m)
55 {
56     int i, j;
57
58     for (i = 0; i < n; i++) {
59         for (j = 0; j < m; j++) {
60             printf("%d ", a[i][j]);
61         }
62         printf("\n");
63     }
64 }
65
66 /* Funkcija koja alokira memoriju za matricu dimenzija nxm */
67 int **alociraj_memoriju(int n, int m)
68 {
69     int i, j;
70     int **a;
```



```

72  a = (int **) malloc(n * sizeof(int *));
    if (a == NULL) {
74      fprintf(stderr, "Problem sa alokacijom memorije!\n");
        exit(EXIT_FAILURE);
76  }
    /* Za svaku vrstu ponaosob */
78  for (i = 0; i < n; i++) {
        /* Alocira se memorija */
80      a[i] = (int *) malloc(m * sizeof(int));
        /* Proverava se da li je doslo do greske prilikom alokacije */
82      if (a[i] == NULL) {
            /* Ako jeste, ispisuje se poruka */
84            fprintf(stderr, "Problem sa alokacijom memorije!\n");
            /* I oslobadja memorija zauzeta do ovog koraka */
86            for (j = 0; j < i; j++) {
                free(a[j]);
88            }
            free(a);
90            exit(EXIT_FAILURE);
        }
92  }

94  return a;
}

96  /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije nxm
    */
98  void oslobodi_memoriju(int **a, int n, int m)
    {
100     int i;
        for (i = 0; i < n; i++) {
102         free(a[i]);
        }
104     free(a);
    }

106  int main(int argc, char *argv[])
108  {
        int **a;
110     int n, m;

112     /* Unos dimenzija matrice */
        printf("Unesite dimenzije matrice: ");
114     scanf("%d %d", &n, &m);

116     /* Alokacija memorije */
        a = alociraj_memoriju(n, m);
118

        /* Ucitavanje elementa matrice */
120     ucitaj_matricu(a, n, m);

122     /* Poziv funkcije koja sortira vrste matrice prema zbiru */

```

```
124     sortiraj_vrstu(a, n, m);  
126     /* Ispis rezultujuće matrice */  
128     printf("Sortirana matrica je:\n");  
130     ispisi_matricu(a, n, m);  
132     /* Oslobađanje memorije */  
134     oslobodi_memoriju(a, n, m);  
136     return 0;  
138 }
```

## 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 `void dodaj_iza(Cvor * tekuci, Cvor * novi)` koja uvezuje u postojeću listu čvor `novi` iza čvora `tekuci`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene 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 kroz main programe iz zadatka 4.1 i koristiti test primere iz istog zadatka.*

[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. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje 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. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza.

Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

<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 program za rad sa binarnim pretraživačkim stablima.

- Definisati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog 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 `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.
- 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** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.

- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

Korišćenjem prethodnih funkcija, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

#### 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
Traži se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuće 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
Traži se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuće 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 pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku **Nedostaje ime ulazne datoteke!**. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

*Test 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.
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 zadatog 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: Skup funkcija napisan u prvom zadatku se može iskoristiti kao mala biblioteka za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka stabla.h predstavljati popis funkcija biblioteke, a datoteka stabla.c njihove*

implementacije. Programe koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.

#### 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 koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Prvo stablo: 1 7 8 9 2 2
Drugo stablo: 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:
Prvo stablo: 11 2 7 5
Drugo stablo: 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.

#### 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.

<i>Test 1</i>	<i>Test 2</i>
<b>Poziv:</b> ./a.out 2 15	<b>Poziv:</b> ./a.out 3 31
<b>ULAZ:</b> 10 5 15 3 2 4 30 12 14 13	<b>ULAZ:</b> 24 53 61 9 7 55 20 16
<b>IZLAZ:</b> 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	<b>IZLAZ:</b> 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

[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.

<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.

<i>Test 1</i>	<i>Test 2</i>
<pre> ULAZ:  10 5 15 2 11 16 1 13  IZLAZ:  7           </pre>	<pre> ULAZ:  16 30 40 24 10 18 45 22  IZLAZ:  6           </pre>

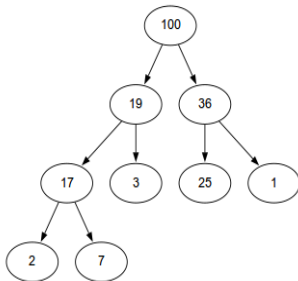
[Rešenje 4.25]

**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.

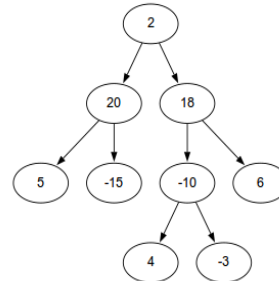
*Test 1*

```
|| 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.

- Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.
- Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- 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

```

1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
6  typedef struct cvor {
   int vrednost;
8  struct cvor *sledeci;
   } Cvor;

10
12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   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
   greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
   NULL ukoliko je lista prazna. */
26 Cvor *pronadji_poslednji(Cvor * glava);

28 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
30 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

32 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov cvor sa vrednoscu broj. */

```

```

34 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
36 /* Funkcija uvezuje cvor novi iza postojećeg cvora tekuci. */
void dodaj_iza(Cvor * tekuci, Cvor * novi);
38
/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
40 sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
   inace vraca 0. */
42 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
44
/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
46 NULL u slučaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);
48
/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   U pretrazi oslanja se na činjenicu da je lista koja se pretražuje
   neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
52 sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
   */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
54
/* Funkcija briše iz liste sve cvorove koji sadrže dati broj. Azurira
   pokazivac na glavu liste, koji može biti promenjen u slučaju da se
56 obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, int broj);
58
/* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
   oslanjajući se na činjenicu da je prosledjena lista sortirana
   neopadajuće. Azurira pokazivac na glavu liste, koji može biti
62 promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
64
/* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
   liste, razdvojene zapetama i uokvirene zagradama. */
66 void ispisi_listu(Cvor * glava);
68
70 #endif

```

```

#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"
4
Cvor *napravi_cvor(int broj)
6 {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    8     if (novi == NULL)
        return NULL;
10
    novi->vrednost = broj;
    12     novi->sledeci = NULL;
    return novi;

```



```
14 }
16 void oslobodi_listu(Cvor ** adresa_glave)
17 {
18     Cvor *pomocni = NULL;
19
20     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
21     while (*adresa_glave != NULL) {
22         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
23            osloboditi cvor koji predstavlja glavu liste */
24         pomocni = (*adresa_glave)->sledeci;
25         free(*adresa_glave);
26         /* Sledeci cvor je nova glava liste. */
27         *adresa_glave = pomocni;
28     }
29 }
30
31 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
32 {
33     /* Kreira se nov cvor i proverava se da li je bilo greske pri
34        alokaciji. */
35     Cvor *novi = napravi_cvor(broj);
36     if (novi == NULL)
37         return 1;
38
39     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste. */
40     novi->sledeci = *adresa_glave;
41     *adresa_glave = novi;
42
43     return 0;
44 }
45
46 Cvor *pronadji_poslednji(Cvor * glava)
47 {
48     /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
49     if (glava == NULL)
50         return NULL;
51
52     /* Sve dok glava pokazuje na cvor koji ima sledeceg, pokazivac
53        glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
54        ce pokazivati na cvor liste koji nema sledeceg, tj. na poslednji
55        cvor liste i vraca se vrednost pokazivaca glava.
56
57        Pokazivac glava je argument funkcije i njegove promene nece se
58        odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
59     while (glava->sledeci != NULL)
60         glava = glava->sledeci;
61
62     return glava;
63 }
64
65 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
```

```

66 {
67     Cvor *novi = napravi_cvor(broj);
68     if (novi == NULL)
69         return 1;
70
71     /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
72        ujedno i cela lista. Azurira se vrednost na koju pokazuje
73        adresa_glave i tako se azurira i pokazivacka promenljiva u
74        pozivajucoj funkciji. */
75     if (*adresa_glave == NULL) {
76         *adresa_glave = novi;
77         return 0;
78     }
79
80     /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
81        se dodaje na kraj liste kao sledbenik poslednjeg. */
82     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
83     poslednji->sledeci = novi;
84
85     return 0;
86 }
87
88 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
89 {
90     /* U praznoj listi nema takvog mesta i vraća se NULL. */
91     if (glava == NULL)
92         return NULL;
93
94     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
95        pokazivala na cvor ciji je sledeci ili ne postoji ili ima
96        vrednost vecu ili jednaku vrednosti novog cvora.
97
98        Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
99        provera da li se doslo do poslednjeg cvora liste pre nego sto se
100       proveru vrednost u sledecem cvoru, jer u slucaju poslednjeg,
101       sledeci ne postoji, pa ni njegova vrednost. */
102     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
103         glava = glava->sledeci;
104
105     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
106        poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
107        vecu od broj. */
108     return glava;
109 }
110
111 void dodaj_iza(Cvor * tekuci, Cvor * novi)
112 {
113     /* Novi cvor se dodaje iza tekuceg cvora. */
114     novi->sledeci = tekuci->sledeci;
115     tekuci->sledeci = novi;
116 }

```

```

118 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
119 {
120     /* U slucaju prazne liste glava nove liste je novi cvor. Ukoliko je
121        doslo do greske pri alokaciji memorije cvraa se 1. */
122     if (*adresa_glave == NULL) {
123         Cvor *novi = napravi_cvor(broj);
124         if (novi == NULL)
125             return 1;
126         *adresa_glave = novi;
127         return 0;
128     }
129
130     /* Lista nije prazna. */
131     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
132        treba dodati na pocetak liste. */
133     if ((*adresa_glave)->vrednost >= broj) {
134         return dodaj_na_pocetak_liste(adresa_glave, broj);
135     }
136
137     /* U slucaju da je glava liste cvor sa vrednoscju manjom od broj,
138        tada se pronalazi cvor liste iza koga treba uvezati nov cvor. */
139     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
140     Cvor *novi = napravi_cvor(broj);
141     if (novi == NULL)
142         return 1;
143
144     /* Uvezuje se novi cvor iza pomocnog. */
145     dodaj_iza(pomocni, novi);
146     return 0;
147 }
148
149 Cvor *pretrazi_listu(Cvor * glava, int broj)
150 {
151     for (; glava != NULL; glava = glava->sledeci)
152         if (glava->vrednost == broj)
153             return glava;
154
155     /* Nema trazenog broja u listi i vraca se NULL. */
156     return NULL;
157 }
158
159 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
160 {
161     /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
162     for (; glava != NULL && glava->vrednost <= broj;
163           glava = glava->sledeci)
164         if (glava->vrednost == broj)
165             return glava;
166
167     return NULL;
168 }

```

```

170 void obrisi_cvor(Cvor ** adresa_glave, int broj)
171 {
172     Cvor *tekuci = NULL;
173     Cvor *pomocni = NULL;
174
175     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
176        broju, i azurira se pokazivac na glavu liste. */
177     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
178     {
179         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
180            adresi adresa_glave. */
181         pomocni = (*adresa_glave)->sledeci;
182         free(*adresa_glave);
183         *adresa_glave = pomocni;
184     }
185
186     /* Ako je nakon toga lista ostala prazna, izlazi se iz funkcije. */
187     if (*adresa_glave == NULL)
188         return;
189
190     /* Od ovog trenutka, u svakoj iteraciji petlje tekuci pokazuje na
191        cvor cija vrednost je razlicita od trazenog broja. Isto vazi i
192        za sve cvorove levo od tekuceg. Poredi se vrednost sledeceg
193        cvora (ako postoji) sa trazenim brojem. Cvor se brise ako je
194        jednak, ili, ako je razlicit, prelazi se na sledeci cvor. Ovaj
195        postupak se ponavlja dok se ne dodje do poslednjeg cvora. */
196     tekuci = *adresa_glave;
197     while (tekuci->sledeci != NULL)
198     {
199         if (tekuci->sledeci->vrednost == broj) {
200             /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
201                prvo cuva u pomocni. */
202             pomocni = tekuci->sledeci;
203             /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
204                njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
205                sledeci od cvora koji se brise. */
206             tekuci->sledeci = pomocni->sledeci;
207             /* Sada treba osloboditi cvor sa vrednoscu broj. */
208             free(pomocni);
209         } else {
210             /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
211                pomera na sledeci. */
212             tekuci = tekuci->sledeci;
213         }
214     }
215     return;
216 }
217
218 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
219 {
220     Cvor *tekuci = NULL;
221     Cvor *pomocni = NULL;
222
223     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom

```

```
222     broju i azurira se pokazivac na glavu liste. */
while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
{
224     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
    adresi adresa_glave. */
    pomocni = (*adresa_glave)->sledeci;
226     free(*adresa_glave);
    *adresa_glave = pomocni;
228 }

230 /* Ako je nakon toga lista ostala prazna, funkcija se prekida. Isto
    se radi i ukoliko glava liste sadrzi vrednost koja je veca od
232     broja, jer kako je lista sortirana rastuce nema potrebe broj
    traziti u repu liste. */
234     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;
236

    /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
238     na cvor cija vrednost je manja od trazenog broja, kao i svim
    cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
240     je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
    ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
242     cija vrednost je veca od trazenog broja. */
    tekuci = *adresa_glave;
244     while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
        )
        if (tekuci->sledeci->vrednost == broj) {
246             pomocni = tekuci->sledeci;
            tekuci->sledeci = tekuci->sledeci->sledeci;
248             free(pomocni);
        } else {
250             /* Ne treba brisati sledeceg od tekuceg jer je manji od
                trazenog i tekuci se pomera na sledeci cvor. */
252             tekuci = tekuci->sledeci;
        }
254     return;
}

256 void ispisi_listu(Cvor * glava)
258 {
    /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
260     jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
    na glavu liste iz pozivajuce funkcije. */
262     putchar('[');
    for (; glava != NULL; glava = glava->sledeci) {
264         printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
266             printf(", ");
    }
268     printf("]\n");
270 }
```

```

#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

/* 1) Glavni program */
int main()
{
    /* Lista je prazna na pocetku. */
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
    int broj;

    /* Testiranje dodavanja novog broja na pocetak liste */
    printf("Unesite brojeve: (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
        /* 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. */
        if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
        }
        printf("\tLista: ");
        ispisi_listu(glava);
    }

    printf("\nUnesite broj koji se trazi: ");
    scanf("%d", &broj);

    trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
    else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

    oslobodi_listu(&glava);

    return 0;
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

/* 2) Glavni program */
int main()
{
    Cvor *glava = NULL;
    int broj;

```

```

12  /* Testiranje dodavanja novog broja na kraj liste. */
printf("Unesite brojeve: (za kraj CTRL+D)\n");
while (scanf("%d", &broj) > 0) {
14     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
        memorije za nov cvor. Memoriju alociranu za cvorove liste
16     treba osloboditi pre napustanja programa. */
    if (dodaj_na_kraj_liste(&glava, broj) == 1) {
18         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
        oslobodi_listu(&glava);
20         exit(EXIT_FAILURE);
    }
22     printf("\tLista: ");
    ispisi_listu(glava);
24 }

26 printf("\nUnesite broj koji se brise: ");
scanf("%d", &broj);
28

/* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
30 procitanom sa ulaza */
obrisi_cvor(&glava, broj);
32

printf("Lista nakon brisanja: ");
34 ispisi_listu(glava);

36 oslobodi_listu(&glava);

38 return 0;
}

```

```

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

5  /* 3) Glavni program */
int main()
7  {
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
    int broj;
11

    /* Testira se dodavanje u listu tako da ona bude neopadajuće
13     uredjena */
    printf("Unosite brojeve (za kraj CTRL+D)\n");
15     while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
            memorije za nov cvor. Memoriju alociranu za cvorove liste
17         treba osloboditi pre napustanja programa. */
        if (dodaj_sortirano(&glava, broj) == 1) {
19             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava);
21             exit(EXIT_FAILURE);
        }
    }
}

```

```

23     }
    printf("\tLista: ");
25     ispisi_listu(glava);
    }

27     printf("\nUnesite broj koji se trazi: ");
29     scanf("%d", &broj);

31     trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
33         printf("Broj %d se ne nalazi u listi!\n", broj);
    else
35         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

37     printf("\nUnesite broj koji se brise: ");
    scanf("%d", &broj);
39

    /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
41     procitanom sa ulaza */
    obrisi_cvor_sortirane_liste(&glava, broj);
43

    printf("Lista nakon brisanja: ");
45     ispisi_listu(glava);

47     oslobodi_listu(&glava);

49     return 0;
    }

```

## Rešenje 4.2

```

1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
6  typedef struct cvor {
    int vrednost;
8     struct cvor *sledeci;
    } Cvor;
10

12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   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 novi cvor na pocetak liste. Vraca 1 ukoliko je

```



```

    bilo greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
    pri alokaciji memorije, inace vraca 0. */
26 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

28 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
    ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
30 memorije, inace vraca 0. */
    int dodaj_sortirano(Cvor ** adresa_glave, int broj);
32

34 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
    NULL u slučaju da takav cvor ne postoji u listi. */
36 Cvor *pretrazi_listu(Cvor * glava, int broj);

38 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
40 neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
    */
42 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

44 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivac na glavu liste, koji može biti promenjen u slučaju da se
46 obrise stara glava liste. */
    void obrisi_cvor(Cvor ** adresa_glave, int broj);
48

50 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na cinjenicu da je prosledjena lista sortirana
    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
52 promenjen ukoliko se obrise stara glava liste. */
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
54

56 /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
    zapetama. */
    void ispisi_vrednosti(Cvor * glava);
58

60 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
    liste, razdvojene zapetama i uokvirene zagradama. */
    void ispisi_listu(Cvor * glava);
62

#endif

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

5 Cvor *napravi_cvor(int broj)
{
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

```

```

    if (novi == NULL)
        return NULL;

    novi->vrednost = broj;
    novi->sledeci = NULL;
    return novi;
}

void oslobodi_listu(Cvor ** adresa_glave)
{
    /* Lista je vec prazna */
    if (*adresa_glave == NULL)
        return;

    /* Ako lista nije prazna, treba osloboditi memoriju. Pre
       oslobadjanja memorije za glavu liste, treba osloboditi rep
       liste. */
    oslobodi_listu(&(*adresa_glave)->sledeci);
    /* Nakon oslobodjenog repa, oslobadja se glava liste, i azurira se
       glava u pozivajucoj funkciji tako da odgovara praznoj listi */
    free(*adresa_glave);
    *adresa_glave = NULL;
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
    /* Kreira se nov cvor i proverava se da li je bilo greske pri
       alokaciji */
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
        return 1;

    /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
    novi->sledeci = *adresa_glave;
    *adresa_glave = novi;
    return 0;
}

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
    if (*adresa_glave == NULL) {
        /* Glava liste je upravo novi cvor i ujedno i cela lista. */
        Cvor *novi = napravi_cvor(broj);
        /* Ukoliko je bilo greske pri alokaciji vraca se 1. */
        if (novi == NULL)
            return 1;

        /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
           azurira i pokazivacka promenljiva u pozivajucoj funkciji. */
        *adresa_glave = novi;
        return 0;
    }
}
```

```

61  /* Ako lista nije prazna, broj se dodaje u rep liste. */
62  /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
63     nov broj se dodaje u rep liste u rekurzivnom pozivu. Informacija
64     o uspesnosti alokacije u rekurzivnom pozivu funkcija prosledjuje
65     visem rekurzivnom pozivu koji tu informaciju vraca u rekurzivni
66     poziv iznad, sve dok se ne vrati u main. Tek je iz main funkcije
67     moguće pristupiti pravom pocetku liste i osloboditi je celu, ako
        ima potrebe. Ako je funkcija vratila 0, onda nije bilo greske.
        */
69  return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
70 }
71
72 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
73 {
74     /* U slucaju prazne liste, glava nove liste je upravo novi cvor. */
75     if (*adresa_glave == NULL) {
76         Cvor *novi = napravi_cvor(broj);
77         if (novi == NULL)
78             return 1;
79
80         *adresa_glave = novi;
81         return 0;
82     }
83
84     /* Lista nije prazna. Ako je broj manji ili jednak vrednosti u
85        glavi liste, onda se dodaje na pocetak liste i vraca se
86        informacija o uspesnosti alokacije. */
87     if ((*adresa_glave)->vrednost >= broj)
88         return dodaj_na_pocetak_liste(adresa_glave, broj);
89
90     /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
91        bude sortirana lista. */
92     return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
93 }
94
95 Cvor *pretrazi_listu(Cvor * glava, int broj)
96 {
97     /* U praznoj listi ga sigurno nema */
98     if (glava == NULL)
99         return NULL;
100
101     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava.
102        */
103     if (glava->vrednost == broj)
104         return glava;
105
106     /* Inace, pretraga se nastavlja u repu liste. */
107     return pretrazi_listu(glava->sledeci, broj);
108 }
109
110 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)

```

```

111 {
112     /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
113        vrednosti u glavi liste, jer je lista neopadajuće sortirana. */
114     if (glava == NULL || glava->vrednost > broj)
115         return NULL;
116
117     /* Ako glava liste sadrži traženi broj, vraća se pokazivač glava.
118        */
119     if (glava->vrednost == broj)
120         return glava;
121
122     /* Inače, pretraga se nastavlja u repu. */
123     return pretrazi_listu(glava->sledeci, broj);
124 }
125
126 void obrisi_cvor(Cvor ** adresa_glave, int broj)
127 {
128     /* U praznoj listi, nema cvorova za brisanje. */
129     if (*adresa_glave == NULL)
130         return;
131
132     /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj. */
133     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
134
135     /* Preostaje provera da li glavu liste treba obrisati. */
136     if ((*adresa_glave)->vrednost == broj) {
137         /* pomocni pokazuje na cvor koji treba da se obrise. */
138         Cvor *pomocni = *adresa_glave;
139         /* Azurira se pokazivač na glavu da pokazuje na sledeci u listi i
140            briše se cvor koji je bio glava liste. */
141         *adresa_glave = (*adresa_glave)->sledeci;
142         free(pomocni);
143     }
144 }
145
146 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
147 {
148     /* Ako je lista prazna ili glava liste sadrži vrednost koja je veća
149        od broja, kako je lista sortirana rastuće nema potrebe broj
150        tražiti u repu liste i zato se funkcija prekida. */
151     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
152         return;
153
154     /* Brisu se cvorovi iz repa koji imaju vrednost broj. */
155     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
156
157     /* Preostaje provera da li glavu liste treba obrisati. */
158     if ((*adresa_glave)->vrednost == broj) {
159         /* pomocni pokazuje na cvor koji treba da se obrise. */
160         Cvor *pomocni = *adresa_glave;
161         /* Azurira se pokazivač na glavu da pokazuje na sledeci u listi i
162            briše se cvor koji je bio glava liste. */

```

```

161     *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
163     }
    }
165 void ispisi_vrednosti(Cvor * glava)
167 {
    /* Prazna lista */
169     if (glava == NULL)
        return;

171     /* Ispisuje se vrednost u glavi liste. */
173     printf("%d", glava->vrednost);

175     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
        se poziva ista funkcija za ispis ostalih. */
177     if (glava->sledeci != NULL) {
        printf(", ");
179         ispisi_vrednosti(glava->sledeci);
    }
181 }

183 void ispisi_listu(Cvor * glava)
    {
185     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
187     na glavu liste iz pozivajuće funkcije. Ona ispisuje samo
        zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
189     rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
        elemente razdvojene zapetom i razmakom. */
191     putchar('[');
        ispisi_vrednosti(glava);
193     printf("]\n");
    }

```

### Rešenje 4.3

```

1  #include<stdio.h>
    #include<string.h>
3  #include<stdlib.h>
    #define MAX_DUZINA 20

5
    typedef struct _Cvor {
7      unsigned broj_pojavljivanja;
        char etiketa[20];
9      struct _Cvor *sledeci;
    } Cvor;

11
    /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
13     ili NULL ako alokacija nije uspesno izvršena. */
    Cvor *napravi_cvor(unsigned br, char *etiketa)

```

```

15 {
16     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
17     if (novi == NULL)
18         return NULL;
19
20     novi->broj_pojavljivanja = br;
21     strcpy(novi->etiketa, etiketa);
22     novi->sledeci = NULL;
23     return novi;
24 }
25
26 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste. */
27 void oslobodi_listu(Cvor ** adresa_glave)
28 {
29     Cvor *pomocni = NULL;
30
31     while (*adresa_glave != NULL) {
32         pomocni = (*adresa_glave)->sledeci;
33         free(*adresa_glave);
34         *adresa_glave = pomocni;
35     }
36 }
37
38 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
39 do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
40 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
41                             char *etiketa)
42 {
43     Cvor *novi = napravi_cvor(br, etiketa);
44     if (novi == NULL)
45         return 1;
46
47     novi->sledeci = *adresa_glave;
48     *adresa_glave = novi;
49     return 0;
50 }
51
52 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
53 NULL ako takav cvor ne postoji. */
54 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
55 {
56     Cvor *tekuci;
57     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
58         if (strcmp(tekuci->etiketa, etiketa) == 0)
59             return tekuci;
60     return NULL;
61 }
62
63 /* Funkcija ispisuje sadrzaj liste */
64 void ispisi_listu(Cvor * glava)
65 {
66     for (; glava != NULL; glava = glava->sledeci)

```

```

67     printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
68 }
69
70 /* Glavni program */
71 int main(int argc, char **argv)
72 {
73     if (argc != 2) {
74         fprintf(stderr,
75             "Greska! Program se poziva sa: ./a.out datoteka.html!\n")
76         ;
77         exit(EXIT_FAILURE);
78     }
79
80     /* Otvaramo datoteku za citanje */
81     FILE *in = NULL;
82     in = fopen(argv[1], "r");
83     if (in == NULL) {
84         fprintf(stderr,
85             "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
86         exit(EXIT_FAILURE);
87     }
88
89     char c;
90     int i = 0;
91     char procitana[MAX_DUZINA];
92     Cvor *glava = NULL;
93     Cvor *trazeni = NULL;
94
95     while ((c = fgetc(in)) != EOF) {
96
97         if (c == '<') {
98             /* Cita se zatvorena etiketa. */
99             if ((c = fgetc(in)) == '/') {
100                 i = 0;
101                 while ((c = fgetc(in)) != '>')
102                     procitana[i++] = c;
103             }
104             /* Cita se otvorena etiketa. */
105             else {
106                 i = 0;
107                 procitana[i++] = c;
108                 while ((c = fgetc(in)) != ' ' && c != '>')
109                     procitana[i++] = c;
110             }
111             procitana[i] = '\0';
112
113             /* Trazi se ucitana etiketa medju postojećim cvorovima liste.
114              Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
115              sa brojem pojavljivanja 1, inace uvecava se broj
116              pojavljivanja etikete. */
117             trazeni = pretrazi_listu(glava, procitana);
118             if (trazeni == NULL) {

```

```

119         if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
120             fprintf(stderr, "Neuspela alokacija za nov cvor\n");
121             oslobodi_listu(&glava);
122             exit(EXIT_FAILURE);
123         } else
124             trazenim->broj_pojavljivanja++;
125     }
126 }
127
128 fclose(in);
129
130 ispisi_listu(glava);
131 oslobodi_listu(&glava);
132
133 return 0;
134 }

```

#### Rešenje 4.4

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 #define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21
7
8 /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
9    studentu. */
10 typedef struct _Cvor {
11     char broj_indeksa[MAX_INDEKS];
12     char ime[MAX_IME_PREZIME];
13     char prezime[MAX_IME_PREZIME];
14     struct _Cvor *sledeci;
15 } Cvor;
16
17 /* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac na nov
18    cvor ili NULL ukoliko alokacija nije prosla. */
19 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
21     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
22     if (novi == NULL)
23         return NULL;
24     /* Inicijalizacija polja novog cvora */
25     strcpy(novi->broj_indeksa, broj_indeksa);
26     strcpy(novi->ime, ime);
27     strcpy(novi->prezime, prezime);
28     novi->sledeci = NULL;
29     return novi;
30 }

```



```
32 /* Funkcija oslobadja memoriju zauzetu za cvorove liste. */
void oslobodi_listu(Cvor ** adresa_glave)
34 {
    if (*adresa_glave == NULL)
36         return;
    /* Rep liste se oslobadja rekurzivnim pozivom. */
38 oslobodi_listu(&(*adresa_glave)->sledeci);
    /* Potom se oslobadja i glava liste. */
40 free(*adresa_glave);
    *adresa_glave = NULL;
42 }

44 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
    do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
46 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
    char *ime, char *prezime)
48 {
    Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
50 if (novi == NULL)
    return 1;
52
    novi->sledeci = *adresa_glave;
54 *adresa_glave = novi;
    return 0;
56 }

58 /* Funkcija ispisuje sadrzaj cvorova liste. */
void ispisi_listu(Cvor * glava)
60 {
    for (; glava != NULL; glava = glava->sledeci)
62         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
            glava->prezime);
64 }

66 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu, u
    suprotnom vraca NULL. */
68 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
    {
69         if (glava == NULL)
            return NULL;
70         if (!strcmp(glava->broj_indeksa, broj_indeksa))
            return glava;
72         return pretrazi_listu(glava->sledeci, broj_indeksa);
74     }

76 int main(int argc, char **argv)
78 {
    /* Argumenti komandne linije su neophodni jer se iz komandne linije
80     dobija ime datoteke sa informacijama o studentima. */
    if (argc != 2) {
82         fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
```

```

84     exit(EXIT_FAILURE);
85 }
86
87 /* Otvaranje datoteke za citanje */
88 FILE *in = NULL;
89 in = fopen(argv[1], "r");
90 if (in == NULL) {
91     fprintf(stderr,
92         "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
93     exit(EXIT_FAILURE);
94 }
95
96 /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
97    listu */
98 char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
99 char broj_indeksa[MAX_INDEKS];
100 Cvor *glava = NULL;
101 Cvor *trazeni = NULL;
102
103 /* Ucitavanje vrednosti u listu */
104 while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
105     if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
106         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
107         oslobodi_listu(&glava);
108         exit(EXIT_FAILURE);
109     }
110
111 /* Datoteka vise nije potrebna i zatvara se. */
112 fclose(in);
113
114 /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
115 while (scanf("%s", broj_indeksa) != EOF) {
116     trazeni = pretrazi_listu(glava, broj_indeksa);
117     if (trazeni == NULL)
118         printf("ne\n");
119     else
120         printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
121 }
122
123 /* Oslobadja se memorija zauzeta za cvorove liste. */
124 oslobodi_listu(&glava);
125
126 return 0;
127 }

```

## Rešenje 4.6

```

1 #include<stdio.h>
2 #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 tih listi. */
Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9  {
   /* Pokazivace 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;
15  Cvor *tekuci = NULL;

17  /* Ako su obe liste prazne i rezultat je prazna lista. */
   if (lista1 == NULL && lista2 == NULL)
19     return NULL;

21  /* Ako je prva lista prazna, rezultat je druga lista. */
   if (lista1 == NULL)
23     return lista2;

25  /* Ako je druga lista prazna, rezultat je prva lista. */
   if (lista2 == NULL)
27     return lista1;

29  /* Rezultujuca pokazuje na pocetak nove liste, tj. na cvor sa
   vrednoscu manjeg od brojeva sadrzanih u cvorovima na koje
31  pokazuju lista1 i lista2. */
   if (lista1->vrednost < lista2->vrednost) {
33     rezultujuca = lista1;
     lista1 = lista1->sledeci;
35  } else {
     rezultujuca = lista2;
37     lista2 = lista2->sledeci;
   }
39  tekuci = rezultujuca;

41  /* Kako rezultujuca pokazuje na pocetak nove liste i ne sme joj se
   menjati vrednost, koristi se pokazivac tekuci koji trenutno
43  sadrzi adresu promenljive rezultujuca. U svakoj iteraciji
   petlje, dobijace adekvatnog sledbenika tako da i nova lista bude
45  uredjena neopadajuće i pomerace se na adresu sledećeg. */
   while (lista1 != NULL && lista2 != NULL) {
47     if (lista1->vrednost < lista2->vrednost) {
         tekuci->sledeci = lista1;
49         lista1 = lista1->sledeci;
     } else {
         tekuci->sledeci = lista2;
51         lista2 = lista2->sledeci;
     }
53     tekuci = tekuci->sledeci;
55  }
```

```

57  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
    rezultujuću listu treba nadovezati ostatak druge liste. */
59  if (lista1 == NULL)
    tekuci->sledeci = lista2;
61  else
    tekuci->sledeci = lista1;
63
64  /* Preko adresa glavi polaznih listi vrednosti pokazivaca u
65  pozivajućoj funkciji se postavljaju na NULL, jer se svi cvorovi
66  prethodnih listi nalaze negde unutar rezultujuće liste. Do njih
67  se može doći prateći pokazivace iz glave rezultujuće liste, tako
    da stare pokazivace treba postaviti na NULL. */
69  *adresa_glave_1 = NULL;
    *adresa_glave_2 = NULL;
71
72  return rezultujuća;
73 }
74
75 /* Glavni program */
76 int main(int argc, char **argv)
77 {
78     /* Argumenti komandne linije su neophodni. */
79     if (argc != 3) {
80         fprintf(stderr,
81             "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
82         exit(EXIT_FAILURE);
83     }
84
85     /* Otvaramo datoteke sa elementima obe liste. */
86     FILE *in1 = NULL;
87     in1 = fopen(argv[1], "r");
88     if (in1 == NULL) {
89         fprintf(stderr,
90             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
91         exit(EXIT_FAILURE);
92     }
93
94     FILE *in2 = NULL;
95     in2 = fopen(argv[2], "r");
96     if (in2 == NULL) {
97         fprintf(stderr,
98             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
99         exit(EXIT_FAILURE);
100    }
101
102    int broj;
103    Cvor *lista1 = NULL;
104    Cvor *lista2 = NULL;
105    Cvor *rezultat = NULL;
106
107    /* Ucitavanje listi */
    while (fscanf(in1, "%d", &broj) != EOF)

```

```

109     dodaj_na_kraj_liste(&lista1, broj);
111     while (fscanf(in2, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&lista2, broj);
113
115     /* Pokazivac rezultat ce pokazivati na glavu liste koja se dobila
        objedinjavanjem listi */
        rezultat = objedini(&lista1, &lista2);
117
119     /* Ispis rezultujuce liste. */
        ispisi_listu(rezultat);
121
123     /* Lista rezultat dobijena prevezivanjem cvorova polaznih listi.
        Njenim oslobadjanjem bice oslobodjena sva zauzeta memorija. */
        oslobodi_listu(&rezultat);
125
127     fclose(in1);
        fclose(in2);
        return 0;
    }

```

### Rešenje 4.8

```

1  #include<stdio.h>
   #include<stdlib.h>
3
   typedef struct cvor {
6     char zagrada;
       struct cvor *sledeci;
7   } Cvor;
9
10  int main()
   {
11     Cvor *stek = NULL;
       FILE *ulaz = NULL;
13     char c;
       Cvor *pomocni = NULL;
15
       ulaz = fopen("izraz.txt", "r");
17     if (ulaz == NULL) {
         fprintf(stderr,
19             "Greska prilikom otvaranja datoteke izraz.txt!\n");
         exit(EXIT_FAILURE);
21     }
23
       while ((c = fgetc(ulaz)) != EOF) {
         /* Ako je učitana otvorena zagrada, stavlja se na stek. */
25         if (c == '(' || c == '{' || c == '[') {
           pomocni = (Cvor *) malloc(sizeof(Cvor));
27           if (pomocni == NULL) {
             fprintf(stderr, "Greska prilikom alokacije memorije!\n");

```

```

29     return 1;
30 }
31 pomocni->zagrada = c;
32 pomocni->sledeci = stek;
33 stek = pomocni;
34 }
35 /* Ako je učitana zatvorena zagrada, proverava se da li je stek
36 prazan i ako nije, da li se na vrhu steka nalazi odgovarajuća
37 otvorena zagrada. */
38 else {
39     if (c == '(' || c == '[' || c == '{') {
40         if (stek != NULL && ((stek->zagrada == '(' && c == ')')
41                               || (stek->zagrada == '[' && c == ']')
42                               || (stek->zagrada == '{' && c == '}')))
43         {
44             /* Sa vrha steka se uklanja otvorena zagrada */
45             pomocni = stek->sledeci;
46             free(stek);
47             stek = pomocni;
48         } else {
49             /* Zgrade u izrazu nisu ispravno uparene. */
50             break;
51         }
52     }
53 }
54 /* Pročitana je cela datoteka. Zatvaramo je. */
55 fclose(ulaz);
56
57 /* Ako je stek prazan i pročitana je cela datoteka, zgrade su
58 ispravno uparene, u suprotnom, nisu. */
59 if (stek == NULL && c == EOF)
60     printf("Zgrade su ispravno uparene.\n");
61 else {
62     printf("Zgrade nisu ispravno uparene.\n");
63     /* U slučaju neispravnog uparivanja treba osloboditi memoriju
64        koja je ostala zauzeta stekom. */
65     while (stek != NULL) {
66         pomocni = stek->sledeci;
67         free(stek);
68         stek = pomocni;
69     }
70 }
71
72 return 0;
73 }

```

## Rešenje 4.9

```

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

```

```

4  #include <string.h>
   #include <ctype.h>

6  #define MAX 100

8  #define OTVORENA 1
   #define ZATVORENA 2

10 #define VAN_ETIKETE 0
12 #define PROCITANO_MANJE 1
   #define U_ETIKETI 2

14 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
16    pokazivac na sledeci cvor. */
   typedef struct cvor {
18     char etiketa[MAX];
       struct cvor *sledeci;
20 } Cvor;

22 /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
   adresu ili NULL ako alokacija nije bila uspesna. */
24 Cvor *napravi_cvor(char *etiketa)
   {
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
28         return NULL;

30     /* Inicijalizacija polja u novom cvoru */
       if (strlen(etiketa) >= MAX) {
32         fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
           etiketa[MAX - 1] = '\0';
34     }
       strcpy(novi->etiketa, etiketa);
36     novi->sledeci = NULL;
       return novi;
38 }

40 /* Funkcija oslobadja memoriju zauzetu stekom. */
   void oslobodi_stek(Cvor ** adresa_vrha)
42 {
       Cvor *pomocni;
44     while (*adresa_vrha != NULL) {
           pomocni = *adresa_vrha;
46         *adresa_vrha = (*adresa_vrha)->sledeci;
           free(pomocni);
48     }
   }

50 /* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske pri
   alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0. */
52 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
54 {

```

```

    Cvor *novi = napravi_cvor(etiketa);
56  if (novi == NULL)
    {
    return 1;
58  }

    novi->sledeci = *adresa_vrha;
60  *adresa_vrha = novi;
    return 0;
62  }

/* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
64  pokazivac razlicit od NULL, tada u niz karaktera na koji on
66  pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
68  suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
    samim tim nije bilo moguće skinuti vrednost sa steka) ili 1 u
    suprotnom. */
70  int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
    {
72      Cvor *pomocni;

74      /* Pokušaj skidanja vrednost sa vrha praznog steka rezultuje
        greskom i vraca se 0. */
76      if (*adresa_vrha == NULL)
        return 0;

78      /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
        adresu kopira etiketa sa vrha steka. */
80      if (etiketa != NULL)
        strcpy(etiketa, (*adresa_vrha)->etiketa);

82      /* Element sa vrha steka se uklanja. */
        pomocni = *adresa_vrha;
84      *adresa_vrha = (*adresa_vrha)->sledeci;
        free(pomocni);
86      return 1;
88  }
90  }

/* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
92  steka. Ukoliko je stek prazan, vraca NULL. */
94  char *vrh_steka(Cvor * vrh)
    {
96      if (vrh == NULL)
        return NULL;
98      return vrh->etiketa;
    }

100  /* Funkcija prikazuje stek od vrha prema dnu. */
102  void prikazi_stek(Cvor * vrh)
    {
104      for (; vrh != NULL; vrh = vrh->sledeci)
        printf("< %s>\n", vrh->etiketa);
106  }

```



```

108 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
110 etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
112 Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
    procita etiketa. Vraca OTVORENA, ako je procitana otvorena
112 etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa. */
int uzmi_etiketu(FILE * f, char *etiketa)
114 {
    int c;
116 int i = 0;
    /* Stanje predstavlja informaciju dokle se stalo sa citanjem
118 etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
    nije zapoceto citanje. */
120 /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
    OTVORENA ili ZATVORENA. */
122 int stanje = VAN_ETIKETE;
    int tip;
124
    /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
126 to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
    samo malim slovima. */
128 while ((c = fgetc(f)) != EOF) {
    switch (stanje) {
130 case VAN_ETIKETE:
        if (c == '<')
132             stanje = PROCITANO_MANJE;
        break;
134 case PROCITANO_MANJE:
        if (c == '/') {
136             /* Cita se zatvorena etiketa. */
            tip = ZATVORENA;
138         } else {
            if (isalpha(c)) {
140                 /* Cita se otvorena etiketa */
                tip = OTVORENA;
142                 etiketa[i++] = tolower(c);
            }
144         }
        /* Od sada se cita etiketa i zato se menja stanje. */
146 stanje = U_ETIKETI;
        break;
148 case U_ETIKETI:
        if (isalpha(c) && i < MAX - 1) {
150             /* Ako je procitani karakter slovo i nije premasena
                dozvoljena duzina etikete, procitani karakter se smanjuje
                i smesta u etiketu. */
152             etiketa[i++] = tolower(c);
154         } else {
            /* Inace, staje se sa citanjem etikete. Korektno se zavrшава
156 niska koja sadrzi procitanu etiketu i vraca se njen tip.
            */
            etiketa[i] = '\0';

```

```

158         return tip;
159     }
160     break;
161 }
162 }
163 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
164    etiketa i vraca se EOF. */
165 return EOF;
166 }

167 int main(int argc, char **argv)
168 {
169     /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
170        nije procitana. */
171     Cvor *vrh = NULL;
172     char etiketa[MAX];
173     int tip;
174     int uparene = 1;
175     FILE *f = NULL;

176     /* Ime datoteke se preuzima iz komandne linije. */
177     if (argc < 2) {
178         fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
179         exit(0);
180     }

181     /* Datoteka se otvara za citanje. */
182     if ((f = fopen(argv[1], "r")) == NULL) {
183         fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
184             argv[1]);
185         exit(1);
186     }

187     /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
188     while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
189         /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
190            etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
191            potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
192            koje nemaju sadrzaj (npr link). Zbog jednostavnosti
193            pretpostavlja se da njih nema u HTML dokumentu. */
194         if (tip == OTVORENA) {
195             if (strcmp(etiketa, "br") != 0
196                 && strcmp(etiketa, "hr") != 0
197                 && strcmp(etiketa, "meta") != 0)
198                 if (potisni_na_stek(&vrh, etiketa) == 1) {
199                     fprintf(stderr, "Neuspela alokacija za nov cvor\n");
200                     oslobodi_stek(&vrh);
201                     exit(EXIT_FAILURE);
202                 }
203         }
204     }

205     /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
206        u pitanju zatvaranje etikete koja je poslednja otvorena, a jos

```

```

210         uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
211         je taj uslov ispunjen, skida se sa steka, jer je upravo
212         zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
213         nisu pravilno uparene. */
214     else if (tip == ZATVORENA) {
215         if (vrh_steka(vrh) != NULL
216             && strcmp(vrh_steka(vrh), etiketa) == 0)
217             skini_sa_steka(&vrh, NULL);
218         else {
219             printf("Etikete nisu pravilno uparene\n");
220             printf("(nadjena je etiketa </%s>", etiketa);
221             if (vrh_steka(vrh) != NULL)
222                 printf(", a poslednja otvorena je </%s>)\n", vrh_steka(vrh));
223             ;
224             else
225                 printf(" koja nije otvorena)\n");
226             uparene = 0;
227             break;
228         }
229     }
230 }
231 /* Završeno je citanje datoteke i zatvara se. */
232 fclose(f);
233
234 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
235    da bude prazan. Ukoliko nije, tada postoje etikete koje su
236    ostale otvorene. */
237 if (uparene) {
238     if (vrh_steka(vrh) == NULL)
239         printf("Etikete su pravilno uparene!\n");
240     else {
241         printf("Etikete nisu pravilno uparene\n");
242         printf("(etiketa </%s> nije zatvorena)\n", vrh_steka(vrh));
243         /* Oslobadja se memorija zauzeta stekom. */
244         oslobodi_stek(&vrh);
245     }
246 }
247 return 0;
248 }

```

### Rešenje 4.10

```

1  #ifndef _RED_H
2  #define _RED_H
3
4
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  #define MAX 1000
9  #define JMBG_DUZINA 14

```

```

11  /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
    opis njegovog zahteva. */
13  typedef struct {
14      char jmbg[JMBG_DUZINA];
15      char opis[MAX];
16  } Zahtev;

17  /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
    korisnika i pokazivac na sledeci cvor liste. */
19  typedef struct cvor {
20      Zahtev nalog;
21      struct cvor *sledeci;
22  } Cvor;

23
25  /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
    poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
    greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
    treba smestiti u nov cvor zbog smestanja manjeg podatka na
    sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
    bajtovima(B) u odnosu na strukturu Zahtev. */
29  Cvor *napravi_cvor(Zahtev * zahtev);

31
33  /* Funkcija prazni red, oslobadjajuci memoriju koji je red zauzeo. */
    void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);

35
37  /* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
    greske pri alokaciji memorije za nov cvor, inace vraca 0. */
    int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
        Zahtev * zahtev);

39
41  /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
    pokazivac razlicit od NULL, tada se u strukturu na koju on
    pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
    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
49  /* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev korisnika
    na pocetku reda. Ukoliko je red prazan, vraca NULL. */
    Zahtev *pocetak_reda(Cvor * pocetak);

51
53  /* Funkcija prikazuje sadrzaj reda. */
    void prikazi_red(Cvor * pocetak);

55 #endif

1  #include "red.h"

3  Cvor *napravi_cvor(Zahtev * zahtev)
    {
5      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

```

```
7     if (novi == NULL)
8         return NULL;
9
10    novi->nalog = *zahtev;
11    novi->sledeci = NULL;
12    return novi;
13 }
14
15 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
16 {
17     Cvor *pomocni = NULL;
18
19     while (*pocetak != NULL) {
20         pomocni = *pocetak;
21         *pocetak = (*pocetak)->sledeci;
22         free(pomocni);
23     }
24     *kraj = NULL;
25 }
26
27 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
28                Zahtev * zahtev)
29 {
30     Cvor *novi = napravi_cvor(zahtev);
31     if (novi == NULL)
32         return 1;
33     /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
34        kraj, to je podjednako efikasno kao dodavanje na pocetak liste
35        */
36     if (*adresa_kraja != NULL) {
37         (*adresa_kraja)->sledeci = novi;
38         *adresa_kraja = novi;
39     } else {
40         /* Ako je red bio ranije prazan */
41         *adresa_pocetka = novi;
42         *adresa_kraja = novi;
43     }
44     return 0;
45 }
46
47 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
48                  Zahtev * zahtev)
49 {
50     Cvor *pomocni = NULL;
51
52     if (*adresa_pocetka == NULL)
53         return 0;
54
55     if (zahtev != NULL)
56         *zahtev = (*adresa_pocetka)->nalog;
57
58     pomocni = *adresa_pocetka;
```

```

57  *adresa_pocetka = (*adresa_pocetka)->sledeci;
    free(pomocni);
59
    if (*adresa_pocetka == NULL)
61        *adresa_kraja = NULL;
63
    return 1;
}
65
Zahtev *pocetak_reda(Cvor * pocetak)
67 {
    if (pocetak == NULL)
69        return NULL;
71
    return &(amp;pocetak->nalog);
}
73
void prikazi_red(Cvor * pocetak)
75 {
    for (; pocetak != NULL; pocetak = pocetak->sledeci)
77        printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
79
    printf("\n");
}

```

```

#include <stdio.h>
2  #include <stdlib.h>
  #include <string.h>
4  #include "red.h"
6
#define VREME_ZA_PAUZU 5
8
int main(int argc, char **argv)
{
10     /* Red je prazan. */
    Cvor *pocetak = NULL, *kraj = NULL;
12     Zahtev nov_zahtev;
    Zahtev *sledeci = NULL;
14     char odgovor[3];
    int broj_usluzenih = 0;
16     FILE *izlaz = fopen("izvestaj.txt", "a");
18
    if (izlaz == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
20        exit(EXIT_FAILURE);
    }
22
    /* Sluzbenik evidentira korisnicke zahteve unosjenjem njihovog JMBG
24     broja i opisa potrebne usluge. */
    printf("Sluzbenik evidentira korisnicke zahteve:\n");
26
    /* Neophodan je poziv funkcije getchar da bi se i nov red nakon

```

```

28     JMBG broja procitao i da bi fgets nakon toga procitala ispravan
        red sa opisom zahteva. */
30 printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
while (scanf("%s", nov_zahtev.jmbg) != EOF) {
32     getchar();
    printf("\tOpis problema: ");
34     fgets(nov_zahtev.opis, MAX - 1, stdin);
    /* Ako je poslednji karakter nov red, eliminiše se. */
36     if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
        nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
38     if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
40         oslobodi_red(&pocetak, &kraj);
        exit(EXIT_FAILURE);
42     }

44     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
}

46 /* Datoteka vise nije potrebna i treba je zatvoriti. */
48 fclose(izlaz);

50 /* Dokle god ima korisnika u redu, treba ih uslužiti. */
while (1) {
52     sledeci = pocetak_reda(pocetak);
    /* Ako nema nikog vise u redu, prekida se petlja. */
54     if (sledeci == NULL)
        break;

56     printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
58     printf("i zahtevom: %s\n", sledeci->opis);

60     skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

62     broj_usluzenih++;

64     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
    scanf("%s", odgovor);

66     if (strcmp(odgovor, "Da") == 0)
68         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
    else
70         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
            nov_zahtev.opis);

72     if (broj_usluzenih == VREME_ZA_PAUZU) {
74         printf("\nDa li je kraj smene? [Da/Ne] ");
        scanf("%s", odgovor);

76         if (strcmp(odgovor, "Da") == 0)
78             break;
        else

```

```

80     broj_usluzenih = 0;
81 }
82 }

84 /*****
85  Usluzivanje korisnika moze da se izvrši i na sledeci nacin:
86  *****/
87 while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
88     printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
89           nov_zahtev.jmbg);
90     printf("sa zahtevom: %s\n", nov_zahtev.opis);
91     broj_usluzenih++;

92     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
93     scanf("%s", odgovor);
94     if (strcmp(odgovor, "Da") == 0)
95         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
96     else
97         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
98               nov_zahtev.jmbg, nov_zahtev.opis);

100     if (broj_usluzenih == VREME_ZA_PAUZU) {
101         printf("\nDa li je kraj smene? [Da/Ne] ");
102         scanf("%s", odgovor);
103         if (strcmp(odgovor, "Da") == 0)
104             break;
105         else
106             broj_usluzenih = 0;
107     }
108 }
109 }
110 *****/

112 /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
113    neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
114    koju zauzima red sa neobradjenim zahtevima korisnika. */
115 oslobodi_red(&pocetak, &kraj);
116
117 return 0;
118 }

```

### Rešenje 4.11

```

1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
   vrednost i pokazivace na sledeci i prethodni cvor liste. */
6  typedef struct cvor {
7      int vrednost;
8      struct cvor *sledeci;
9      struct cvor *prethodni;

```



```

10 } Cvor;

12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14 na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
    Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
    ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na
18 adresi adresa_kraja. */
    void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);

22 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
    bilo greske pri alokaciji memorije, inace vraca 0. */
24 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
    adresa_kraja, int broj);

26 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
    pri alokaciji memorije, inace vraca 0. */
28 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
    int broj);

30 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
    nov cvor sa vrednoscu broj. */
32 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

34 /* Funkcija uvezuje cvor novi iza postojeceg cvora tekuci. */
    void dodaj_iza(Cvor * tekuci, Cvor * novi);

38 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
40 inace vraca 0. */
    int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
    broj);

42 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
    NULL u slucaju da takav cvor ne postoji u listi. */
44 Cvor *pretrazi_listu(Cvor * glava, int broj);

46 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
50 neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
    sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
    */
52 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

54 /* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
    pokazivac glava se nalazi na adresi adresa_glave. */
56 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
    tekuci);

60

```

```

62  /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
    pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
    obrise stara glava. */
64  void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
        broj);
66
68  /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
    oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
    sortirana. Azurira pokazivac na glavu liste, koji moze biti
    promenjen ukoliko se obrise stara glava liste. */
70  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
        adresa_kraja, int broj);
72
74  /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
    liste, razdvojene zapetama i uokvirene zagradama. */
76  void ispisi_listu(Cvor * glava);
78  /* Funkcija prikazuje cvorove liste pocev od kraja ka glavi liste. */
    void ispisi_listu_unazad(Cvor * kraj);
80
    #endif

```

```

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

5  Cvor *napravi_cvor(int broj)
    {
7      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
        if (novi == NULL)
9          return NULL;

11     novi->vrednost = broj;
        novi->sledeci = NULL;
13     return novi;
    }

15  void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
17  {
        Cvor *pomocni = NULL;

19
21     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
        while (*adresa_glave != NULL) {
            /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
23             osloboditi memoriju cvora koji predstavlja glavu liste. */
            pomocni = (*adresa_glave)->sledeci;
25             free(*adresa_glave);
            /* Sledeci cvor je nova glava liste. */
27             *adresa_glave = pomocni;
        }
29     *adresa_kraja = NULL;
    }

```

```
31 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
33     adresa_kraja, int broj)
34 {
35     Cvor *novi = napravi_cvor(broj);
36     if (novi == NULL)
37         return 1;
38
39     /* Sledbenik novog cvora je glava stare liste */
40     novi->sledeci = *adresa_glave;
41     /* Ako stara lista nije bila prazna, onda prethodni od glave treba
42        da bude nov cvor. Inace, nov cvor je ujedno pocetni i krajnji.
43        */
44     if (*adresa_glave != NULL)
45         (*adresa_glave)->prethodni = novi;
46     else
47         *adresa_kraja = novi;
48     /* Novi cvor je nova glava liste. */
49     *adresa_glave = novi;
50
51     return 0;
52 }
53
54 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
55     int broj)
56 {
57     Cvor *novi = napravi_cvor(broj);
58     if (novi == NULL)
59         return 1;
60
61     /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
62        ujedno i cela lista. Azurira se vrednost na koju pokazuju
63        adresa_glave i adresa_kraja tako se azurira i pokazivacka
64        promenljiva u pozivajucoj funkciji. */
65     if (*adresa_glave == NULL) {
66         *adresa_glave = novi;
67         *adresa_kraja = novi;
68         return 0;
69     }
70
71     /* Kako lista nije prazna, novi cvor se dodaje na kraj liste kao
72        sledbenik poslednjeg i azurira se pokazivac na kraj u
73        pozivajucoj funkciji. */
74     (*adresa_kraja)->sledeci = novi;
75     novi->prethodni = (*adresa_kraja);
76     *adresa_kraja = novi;
77
78     return 0;
79 }
80
81 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
82 {
```

```
83  /* U praznoj listi nema takvog mesta i vraća se NULL. */
    if (glava == NULL)
        return NULL;
85
    /* Pokazivač glava se pomera na sledeći cvor sve dok ne bude
87     pokazivala na cvor čiji je sledeći ili ne postoji ili ima
        vrednost veću ili jednaku vrednosti novog cvora.
89
        Zbog izračunavanja izraza u C-u prvi deo konjukcije mora biti
91     provera da li se doslo do poslednjeg cvora liste pre nego što se
        proveri vrednost u sledećem cvoru, jer u slučaju poslednjeg,
93     sledeći ne postoji, pa ni njegova vrednost. */
    while (glava->sledeći != NULL && glava->sledeći->vrednost < broj)
95         glava = glava->sledeći;

97     /* Iz petlje se moglo izaci pomeranjem pokazivača glava do
        poslednjeg cvora ili, ranije, na cvoru čiji sledeći ima vrednost
99     veću od broj. */
    return glava;
101 }

103 void dodaj_iza(Cvor * tekuci, Cvor * novi)
{
105     novi->sledeći = tekuci->sledeći;
    novi->prethodni = tekuci;
107
    /* Ako tekuci ima sledećeg, onda se sledećem dodeljuje prethodnik,
109     a potom i tekuci dobija novog sledećeg postavljanjem pokazivača
        na ispravne adrese. */
111     if (tekuci->sledeći != NULL)
        tekuci->sledeći->prethodni = novi;
113     tekuci->sledeći = novi;
}

115
117 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
        broj)
{
119     /* Ako je lista prazna, nov cvor je i prvi i poslednji cvor liste.
        */
    if (*adresa_glave == NULL) {
121         Cvor *novi = napravi_cvor(broj);
        if (novi == NULL)
123             return 1;
        *adresa_glave = novi;
125         *adresa_kraja = novi;
        return 0;
127     }

129     /* Ukoliko je vrednost glave liste veća ili jednaka od nove
        vrednosti onda nov cvor treba staviti na početak liste. */
131     if ((*adresa_glave)->vrednost >= broj) {
        dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
    }
```

```

133     return 0;
134 }
135
136 Cvor *novi = napravi_cvor(broj);
137 if (novi == NULL)
138     return 1;
139
140 /* Pronazi se cvor iza koga treba uvezati nov cvor. */
141 Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
142 dodaj_iza(pomocni, novi);
143 /* Ako pomocni pokazuje na poslednji element onda je novi cvor
144    poslednji u listi. */
145 if (pomocni == *adresa_kraja)
146     *adresa_kraja = novi;
147
148 return 0;
149 }
150
151 Cvor *pretrazi_listu(Cvor * glava, int broj)
152 {
153     for (; glava != NULL; glava = glava->sledeci)
154         if (glava->vrednost == broj)
155             return glava;
156
157     /* Nema trazenog broja u listi i vraca se NULL. */
158     return NULL;
159 }
160
161 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
162 {
163     /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
164     for (; glava != NULL && glava->vrednost <= broj;
165           glava = glava->sledeci)
166         if (glava->vrednost == broj)
167             return glava;
168
169     /* Nema trazenog broja u listi i bice vrateno NULL. */
170     return NULL;
171 }
172
173 /* Kod dvostruko povezane liste brisanje cvora na koji pokazuje
174    tekuci moze se lako uraditi jer sadrzi pokazivace na svog
175    sledbenika i prethodnika u listi. */
176 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
177                   tekuci)
178 {
179     /* Ako je tekuci NULL pokazivac, nema sta da se brise. */
180     if (tekuci == NULL)
181         return;
182
183     /* Ako postoji prethodnik od tekuceg, onda se postavlja da njegov
184        sledeci bude sledeci od tekuceg. */

```

```

185     if (tekuci->prethodni != NULL)
186         tekuci->prethodni->sledeci = tekuci->sledeci;
187
188     /* Ako postoji sledbenik tekućeg, onda njegov prethodnik treba da
189        bude prethodnik tekućeg. */
190     if (tekuci->sledeci != NULL)
191         tekuci->sledeci->prethodni = tekuci->prethodni;
192
193     /* Ako je glava cvor koji se briše, nova glava liste biće sledbenik
194        stare glave. */
195     if (tekuci == *adresa_glave)
196         *adresa_glave = tekuci->sledeci;
197
198     /* Ako je cvor koji se briše, poslednji u listi, azurira se i
199        pokazivac na kraj liste u pozivajućoj funkciji. */
200     if (tekuci == *adresa_kraja)
201         *adresa_kraja = tekuci->prethodni;
202
203     /* Oslobadja se dinamički alocirani prostor za cvor tekuci. */
204     free(tekuci);
205 }
206
207 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
208 )
209 {
210     Cvor *tekuci = *adresa_glave;
211
212     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
213         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
214 }
215
216 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
217                                adresa_kraja, int broj)
218 {
219     Cvor *tekuci = *adresa_glave;
220
221     while ((tekuci =
222             pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
223         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
224 }
225
226 void ispisi_listu(Cvor * glava)
227 {
228     /* Funkciji se ne šalje adresa promenljive koja čuva glavu liste,
229        jer neće menjati listu, pa nema ni potrebe da azurira pokazivac
230        na glavu liste iz pozivajuće funkcije. */
231     putchar('[');
232     for (; glava != NULL; glava = glava->sledeci) {
233         printf("%d", glava->vrednost);
234         if (glava->sledeci != NULL)
235             printf(", ");
236     }
237 }

```

```

237     printf("]\n");
238 }
239
240 void ispisi_listu_unazad(Cvor * kraj)
241 {
242     putchar('[');
243     if (kraj == NULL) {
244         printf("]\n");
245         return;
246     }
247
248     for (; kraj != NULL; kraj = kraj->prethodni) {
249         printf("%d", kraj->vrednost);
250         if (kraj->prethodni != NULL)
251             printf(", ");
252     }
253     printf("]\n");
254 }

```

```

#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

/* 1) Glavni program */
int main()
{
    /* 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
       liste unazad bile efikasne poput dodavanja na pocetak liste i
       ispisivanja liste od pocetnog do poslednjeg cvora. */
    Cvor *glava = NULL;
    Cvor *kraj = NULL;
    Cvor *trazeni = NULL;
    int broj;

    /* Testiranje dodavanja novog broja na pocetak liste. */
    printf("Unesite brojeve: (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
        /* 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. */
        if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava, &kraj);
            exit(EXIT_FAILURE);
        }
        printf("\tLista: ");
        ispisi_listu(glava);
    }
}

```

```

34     printf("\nUnesite broj koji se trazi u listi: ");
    scanf("%d", &broj);

36     trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
38         printf("Broj %d se ne nalazi u listi!\n", broj);
    else
40         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

42     printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);

44     oslobodi_listu(&glava, &kraj);

46     return 0;
48 }

```

```

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

4
/* 2) Glavni program */
6 int main()
{
8     Cvor *glava = NULL;
    Cvor *kraj = NULL;
10     int broj;

12     /* Testiranje dodavanja novog broja na kraj liste. */
    printf("Unesite brojeve: (za kraj unesite 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, &kraj, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava, &kraj);
            exit(EXIT_FAILURE);
22         }
        printf("\tLista: ");
24         ispisi_listu(glava);
    }

26     printf("\nUnesite broj koji se brise iz liste: ");
28     scanf("%d", &broj);

30     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
        procitanom sa ulaza. */
32     obrisi_cvor(&glava, &kraj, broj);

34     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

```



```

36     printf("\nLista ispisana u nazad: ");
38     ispisi_listu_unazad(kraj);

40     oslobodi_listu(&glava, &kraj);

42     return 0;
}

```

```

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

5  /* 3) Glavni program */
   int main()
7  {
   Cvor *glava = NULL;
   Cvor *kraj = NULL;
   Cvor *trazeni = NULL;
11  int broj;

13  /* Testira se dodavanje u listu tako da ona bude neopadajuće
   uredjena */
15  printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
   while (scanf("%d", &broj) > 0) {
17      /* 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. */
19      if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
21          fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
           oslobodi_listu(&glava, &kraj);
23          exit(EXIT_FAILURE);
       }
25       printf("\tLista: ");
           ispisi_listu(glava);
27   }

29   printf("\nUnesite broj koji se trazi u listi: ");
   scanf("%d", &broj);

31   trazeni = pretrazi_listu(glava, broj);
33   if (trazeni == NULL)
       printf("Broj %d se ne nalazi u listi!\n", broj);
35   else
       printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

37   printf("\nUnesite broj koji se brise iz liste: ");
39   scanf("%d", &broj);

41   /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
       procitanom sa ulaza. */
43   obrisi_cvor_sortirane_liste(&glava, &kraj, broj);

```

```

45 printf("Lista nakon brisanja: ");
   ispisi_listu(glava);
47
   printf("\nLista ispisana u nazad: ");
49   ispisi_listu_unazad(kraj);
51   oslobodi_listu(&glava, &kraj);
53   return 0;
   }

```

#### Rešenje 4.14

```

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

/* a) Struktura kojom se predstavlja cvor binarnog pretraživackog
   stabla */
typedef struct cvor {
    int broj;
    struct cvor *levo;
    struct cvor *desno;
} Cvor;

/* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
Cvor *napravi_cvor(int broj)
{
    /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;

    /* Inicijalizuju se polja novog cvora. */
    novi->broj = broj;
    novi->levo = NULL;
    novi->desno = NULL;

    /* Vraca se adresa novog cvora. */
    return novi;
}

/* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
void prover_i_alokaciju(Cvor * novi_cvor)
{
    /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {

```

```

38      /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa
   */
40      fprintf(stderr, "Malloc greska za novi cvor!\n");
      exit(EXIT_FAILURE);
42  }
   }
44
   /* c) Funkcija koja dodaje zadati broj u stablo */
46 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
   {
48     /* Ako je stablo prazno */
      if (*adresa_korena == NULL) {
50
52         /* Kreira se novi cvor */
          Cvor *novi = napravi_cvor(broj);
          prover_i_alokaciju(novi);
54
56         /* I proglašava se korenom stabla */
          *adresa_korena = novi;
          return;
58     }

60     /* U suprotnom trazi se odgovarajuca pozicija za zadati broj: */

62     /* Ako je zadata vrednost manja od vrednosti korena */
      if (broj < (*adresa_korena)->broj)
64
66         /* Broj se dodaje u levo podstablo */
          dodaj_u_stablo(&(*adresa_korena)->levo, broj);

68     else
69         /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
70            dodaje u desno podstablo */
          dodaj_u_stablo(&(*adresa_korena)->desno, broj);
72 }

74 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
   Cvor *pretrazi_stablo(Cvor * koren, int broj)
76 {
78     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
      if (koren == NULL)
80         return NULL;

82     /* Ako je trazena vrednost sadrzana u korenu */
      if (koren->broj == broj) {
84
86         /* Prekidamo pretragu */
          return koren;
      }

88     /* Inace, ako je broj manji od vrednosti sadrzane u korenu */

```

```
90     if (broj < koren->broj)
91     {
92         /* Pretraga se nastavlja u levom podstablu */
93         return pretrazi_stablo(koren->levo, broj);
94     }
95     else
96     {
97         /* U suprotnom, pretraga se nastavlja u desnom podstablu */
98         return pretrazi_stablo(koren->desno, broj);
99     }
100 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost u stablu
    */
    Cvor *pronadji_najmanji(Cvor * koren)
102 {
103     /* Ako je stablo prazno, prekida se pretraga */
104     if (koren == NULL)
105     {
106         return NULL;
107     }
108     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
        levo od njega */
109
110     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
        najmanju vrednost */
111     if (koren->levo == NULL)
112     {
113         return koren;
114     }
115     /* Inace, pretragu treba nastaviti u levom podstablu */
116     return pronadji_najmanji(koren->levo);
117 }
118
119 /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u stablu
    */
    Cvor *pronadji_najveci(Cvor * koren)
122 {
123     /* Ako je stablo prazno, prekida se pretraga */
124     if (koren == NULL)
125     {
126         return NULL;
127     }
128     /* Vrednosti koje su veće od vrednosti u korenu stabla nalaze se
        desno od njega */
129
130     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
        najveću vrednost */
131     if (koren->desno == NULL)
132     {
133         return koren;
134     }
135     /* Inace, pretragu treba nastaviti u desnom podstablu */
136     return pronadji_najveci(koren->desno);
137 }
138
139 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
```

```
140 void obrisi_element(Cvor ** adresa_korena, int broj)
141 {
142     Cvor *pomocni_cvor = NULL;
143
144     /* Ako je stablo prazno, brisanje nije primenljivo */
145     if (*adresa_korena == NULL)
146         return;
147
148     /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
149        stabla, ona se eventualno nalazi u levom podstablu, pa treba
150        rekursivno primeniti postupak na levo podstablo. Koren ovako
151        modifikovanog stabla je nepromenjen. */
152     if (broj < (*adresa_korena)->broj) {
153         obrisi_element(&(*adresa_korena)->levo, broj);
154         return;
155     }
156
157     /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
158        stabla, ona se eventualno nalazi u desnom podstablu pa treba
159        rekursivno primeniti postupak na desno podstablo. Koren ovako
160        modifikovanog stabla je nepromenjen. */
161     if ((*adresa_korena)->broj < broj) {
162         obrisi_element(&(*adresa_korena)->desno, broj);
163         return;
164     }
165
166     /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
167        jednaka broju koji se brise (tj. slucaj kada treba obrisati
168        koren) */
169
170     /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
171        prazno stablo (vraca se NULL) */
172     if ((*adresa_korena)->levo == NULL
173         && (*adresa_korena)->desno == NULL) {
174         free(*adresa_korena);
175         *adresa_korena = NULL;
176         return;
177     }
178
179     /* Ako koren ima samo levog sina, tada se brisanje vrši tako što se
180        brise koren, a novi koren postaje levi sin */
181     if ((*adresa_korena)->levo != NULL
182         && (*adresa_korena)->desno == NULL) {
183         pomocni_cvor = (*adresa_korena)->levo;
184         free(*adresa_korena);
185         *adresa_korena = pomocni_cvor;
186         return;
187     }
188
189     /* Ako koren ima samo desnog sina, tada se brisanje vrši tako što
190        se brise koren, a novi koren postaje desni sin */
191     if ((*adresa_korena)->desno != NULL
```

```

192     && (*adresa_korena)->levo == NULL) {
193         pomocni_cvor = (*adresa_korena)->desno;
194         free(*adresa_korena);
195         *adresa_korena = pomocni_cvor;
196         return;
197     }
198
199     /* Slučaj kada koren ima oba sina - najpre se potrazi sledbenik
200        korena (u smislu poretka) u stablu. To je upravo po vrednosti
201        najmanji cvor u desnom podstablu. On se može pronaći npr.
202        funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
203        vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
204        broj koji se briše). Zatim se prosto rekurzivno pozove funkcija
205        za brisanje na desno podstablo. S obzirom da u njemu treba
206        obrisati najmanji element, a on zasigurno ima najviše jednog
207        potomka, jasno je da će njegovo brisanje biti obavljeno na jedan
208        od jednostavnijih načina koji su gore opisani. */
209     pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
210     (*adresa_korena)->broj = pomocni_cvor->broj;
211     pomocni_cvor->broj = broj;
212     obrisi_element(&(*adresa_korena)->desno, broj);
213 }
214
215 /* h) Funkcija ispisuje stablo u infiksnoj notaciji (Levo postablo -
216     Koren - Desno podstablo) */
217 void ispisi_stablo_infiksno(Cvor * koren)
218 {
219     /* Ako stablo nije prazno */
220     if (koren != NULL) {
221
222         /* Prvo se ispisuju svi cvorovi levo od korena */
223         ispisi_stablo_infiksno(koren->levo);
224
225         /* Zatim se ispisuje vrednost u korenu */
226         printf("%d ", koren->broj);
227
228         /* Na kraju se ispisuju cvorovi desno od korena */
229         ispisi_stablo_infiksno(koren->desno);
230     }
231 }
232
233 /* i) Funkcija ispisuje stablo u prefiksnoj notaciji (Koren - Levo
234     podstablo - Desno podstablo) */
235 void ispisi_stablo_prefiksno(Cvor * koren)
236 {
237     /* Ako stablo nije prazno */
238     if (koren != NULL) {
239
240         /* Prvo se ispisuje vrednost u korenu */
241         printf("%d ", koren->broj);
242
243         /* Zatim se ispisuju svi cvorovi levo od korena */

```

```

244     ispisi_stablo_prefiksno(koren->levo);

246     /* Na kraju se ispisuju svi cvorovi desno od korena */
    ispisi_stablo_prefiksno(koren->desno);
248 }
}

250 /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo podstablo
252 - Desno postablo - Koren) */
void ispisi_stablo_postfiksno(Cvor * koren)
254 {

256     /* Ako stablo nije prazno */
    if (koren != NULL) {

258         /* Prvo se ispisuju svi cvorovi levo od korena */
        ispisi_stablo_postfiksno(koren->levo);

260         /* Zatim se ispisuju svi cvorovi desno od korena */
        ispisi_stablo_postfiksno(koren->desno);

262         /* Na kraju se ispisuje vrednost u korenu */
        printf("%d ", koren->broj);
264     }
}

266 }

268 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
void oslobodi_stablo(Cvor ** adresa_korena)
270 {
272     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*adresa_korena == NULL)
274         return;

276     /* Inace ... */
    /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);

278     /* Oslobadja se memorija zauzeta desnim podstablom */
    oslobodi_stablo(&(*adresa_korena)->desno);

280     /* Oslobadja se memorija zauzeta korenom */
    free(*adresa_korena);

282     /* Proglasava se stablo praznim */
    *adresa_korena = NULL;
284 }

286

288

290 int main()
292 {
    Cvor *koren;
    int n;
294     Cvor *trazeni_cvor;

```

```

296  /* Proglasava se stablo praznim */
298  koren = NULL;

300  /* Dodaju se vrednosti u stablo */
301  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
302  while (scanf("%d", &n) != EOF) {
303      dodaj_u_stablo(&koren, n);
304  }

306  /* Generisu se trazeni ispisi: */
307  printf("\nInfiksni ispisi: ");
308  ispisi_stablo_infiksno(koren);
309  printf("\nPrefiksni ispisi: ");
310  ispisi_stablo_prefiksno(koren);
311  printf("\nPostfiksni ispisi: ");
312  ispisi_stablo_postfiksno(koren);

314  /* Demonstrira se rad funkcije za pretragu */
315  printf("\nTrazi se broj: ");
316  scanf("%d", &n);
317  trazeni_cvor = pretrazi_stablo(koren, n);
318  if (trazeni_cvor == NULL)
319      printf("Broj se ne nalazi u stablu!\n");
320
321  else
322      printf("Broj se nalazi u stablu!\n");

324  /* Demonstrira se rad funkcije za brisanje */
325  printf("Brise se broj: ");
326  scanf("%d", &n);
327  obrisi_element(&koren, n);
328  printf("Rezultujuce stablo: ");
329  ispisi_stablo_infiksno(koren);
330  printf("\n");

332  /* Oslobadja se memorija zauzeta stablom */
333  oslobodi_stablo(&koren);
334  return 0;
}

```

### 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

```



```

10     pojavljivanja i redom pokazivace na levo i desno podstablo */
11 typedef struct cvor {
12     char *rec;
13     int brojac;
14     struct cvor *levo;
15     struct cvor *desno;
16 } Cvor;
17
18 /* Funkcija koja kreira novi cvora stabla */
19 Cvor *napravi_cvor(char *rec)
20 {
21     /* Alocira se memorija za novi cvor i proverava se uspesnost
22        alokacije. */
23     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
24     if (novi_cvor == NULL)
25         return NULL;
26
27     /* Alocira se memorija za zadatu rec: potrebno je rezervisati
28        memoriju za svaki karakter reci ukljucujuci i terminirajucu
29        nulu */
30     novi_cvor->rec =
31         (char *) malloc((strlen(rec) + 1) * sizeof(char));
32     if (novi_cvor->rec == NULL) {
33         free(novi_cvor);
34         return NULL;
35     }
36
37     /* Inicijalizuju se polja u novom cvoru */
38     strcpy(novi_cvor->rec, rec);
39     novi_cvor->brojac = 1;
40     novi_cvor->levo = NULL;
41     novi_cvor->desno = NULL;
42
43     /* Vraca se adresa novog cvora */
44     return novi_cvor;
45 }
46
47 /* Funkcija koja proverava uspesnost kreiranja novog cvora
48    stabla */
49 void prover_i_alokaciju(Cvor * novi_cvor)
50 {
51     /* Ukoliko je cvor neuspesno kreiran */
52     if (novi_cvor == NULL) {
53         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
54            programa */
55         fprintf(stderr, "Malloc greska za novi cvor!\n");
56         exit(EXIT_FAILURE);
57     }
58 }
59
60 /* Funkcija koja dodaje novu rec u stablo. */
61 void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)

```

```

62 {
63     /* Ako je stablo prazno */
64     if (*adresa_korena == NULL) {
65         /* Kreira se cvor koji sadrzi zadatu rec */
66         Cvor *novi = napravi_cvor(rec);
67         prover_i_alokaciju(novi);
68
69         /* I proglašava se korenom stabla */
70         *adresa_korena = novi;
71         return;
72     }
73
74     /* U suprotnom se trazi odgovarajuca pozicija za novu rec */
75
76     /* Ako je rec leksikografski manja od reci u korenu ubacuje se
77     u levo podstablo */
78     if (strcmp(rec, (*adresa_korena)->rec) < 0)
79         dodaj_u_stablo(&(*adresa_korena)->levo, rec);
80
81     else
82         /* Ako je rec leksikografski veca od reci u korenu ubacuje
83         se u desno podstablo */
84         dodaj_u_stablo(&(*adresa_korena)->desno, rec);
85
86     else
87         /* Ako je rec jednaka reci u korenu, uvecava se njen broj
88         pojavljivanja */
89         (*adresa_korena)->brojac++;
90 }
91
92 /* Funkcija koja oslobadja memoriju zauzetu stablom */
93 void oslobodi_stablo(Cvor ** adresa_korena)
94 {
95     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
96     if (*adresa_korena == NULL)
97         return;
98
99     /* Inace ... */
100    /* Oslobadja se memorija zauzeta levim podstablom */
101    oslobodi_stablo(&(*adresa_korena)->levo);
102
103    /* Oslobadja se memorija zauzeta desnim podstablom */
104    oslobodi_stablo(&(*adresa_korena)->desno);
105
106    /* Oslobadja se memorija zauzeta korenom */
107    free((*adresa_korena)->rec);
108    free(*adresa_korena);
109
110    /* Stablo se proglašava praznim */
111    *adresa_korena = NULL;
112 }

```

```

114 /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec
      (rec sa najvećim brojem pojavljivanja) */
116 Cvor *nadj_i_najfrekventniju_rec(Cvor * koren)
      {
118     Cvor *max, *max_levo, *max_desno;

120     /* Ako je stablo prazno, prekida se sa pretragom */
      if (koren == NULL)
122         return NULL;

124     /* Pronalazi se najfrekventnija rec u levom podstablu */
      max_levo = nadj_i_najfrekventniju_rec(koren->levo);
126
128     /* Pronalazi se najfrekventnija rec u desnom podstablu */
      max_desno = nadj_i_najfrekventniju_rec(koren->desno);

130     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
      podstabla, korena i desnog podstabla */
132     max = koren;
      if (max_levo != NULL && max_levo->brojac > max->brojac)
134         max = max_levo;
      if (max_desno != NULL && max_desno->brojac > max->brojac)
136         max = max_desno;

138     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
      return max;
140 }

142 /* Funkcija koja ispisuje reci iz stabla u leksikografskom
      poretku pracene brojem pojavljivanja */
144 void prikazi_stablo(Cvor * koren)
      {
146     /* Ako je stablo prazno, završava se sa ispisom */
      if (koren == NULL)
148         return;

150     /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz
      levog podstabla */
152     prikazi_stablo(koren->levo);

154     /* Zatim rec iz korena */
      printf("%s: %d\n", koren->rec, koren->brojac);
156
158     /* I nastavlja se sa ispisom reci iz desnog podstabla */
      prikazi_stablo(koren->desno);
160 }

162 /* Funkcija ucitava sledeću rec iz zadate datoteke f i upisuje
      je u niz rec. Maksimalna dužina reci je određena argumentom
      max. Funkcija vraća EOF ako u datoteci nema više reci ili 0 u
164     suprotnom. Rec je niz malih ili velikih slova. */

```

```

166 int procitaj_rec(FILE * f, char rec[], int max)
167 {
168     /* Karakter koji se cita */
169     int c;
170
171     /* Indeks pozicije na koju se smesta procitani karakter */
172     int i = 0;
173
174     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se
175        god nije stiglo do kraja datoteke... */
176     while (i < max - 1 && (c = fgetc(f)) != EOF) {
177         /* Proverava se da li je procitani karakter slovo */
178         if (isalpha(c))
179             /* Ako jeste, smesta se u niz - pritom se vrsi konverzija
180                u mala slova jer program treba da bude neosetljiv na
181                razliku izmedju malih i velikih slova */
182             rec[i++] = tolower(c);
183
184         else
185             /* Ako nije, proverava se da li je procitano barem jedno
186                slovo nove reci */
187             /* Ako jeste, prekida se sa citanjem */
188             if (i > 0)
189                 break;
190
191         /* U suprotnom se ide na sledecu iteraciju */
192     }
193
194     /* Dodaje se na rec terminirajuca nula */
195     rec[i] = '\0';
196
197     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
198     return i > 0 ? 0 : EOF;
199 }
200
201 int main(int argc, char **argv)
202 {
203     Cvor *koren = NULL, *max;
204     FILE *f;
205     char rec[MAX];
206
207     /* Provera da li je navedeno ime datoteke prilikom pokretanja
208        programa */
209     if (argc < 2) {
210         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
211         exit(EXIT_FAILURE);
212     }
213
214     /* Priprema datoteke za citanje */
215     if ((f = fopen(argv[1], "r")) == NULL) {
216         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
217                 argv[1]);

```

```

218     exit(EXIT_FAILURE);
    }

220     /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
       pretrage. */
222     while (procitaj_rec(f, rec, MAX) != EOF)
        dodaj_u_stablo(&koren, rec);

224

226     /* Posto je citanjem reci zavrsono, zatvara se datoteka */
    fclose(f);

228     /* Prikazuju se sve reci iz teksta i brojevi njihovih
       pojavljivanja. */
230     prikazi_stablo(koren);

232     /* Pronalazi se najfrekventnija rec */
    max = najdi_najfrekventniju_rec(koren);

234

236     /* Ako takve reci nema... */
    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 dinamicki alociran prostor za stablo */
    oslobodi_stablo(&koren);

248
    return 0;
250 }

```

### Rešenje 4.16

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <ctype.h>
5
   #define MAX_IME_DATOTEKE 50
7  #define MAX_CIFARA 13
   #define MAX_IME_I_PREZIME 100
9
   /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
11    broj telefona i redom pokazivace na levo i desno podstablo */
   typedef struct cvor {
13     char ime_i_prezime[MAX_IME_I_PREZIME];
     char telefon[MAX_CIFARA];

```

```

15     struct cvor *levo;
16     struct cvor *desno;
17 } Cvor;

19 /* Funkcija koja kreira novi cvora stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
21 {
22     /* Alocira se memorija za novi cvor i proverava se uspesnost
23        alokacije. */
24     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
25     if (novi_cvor == NULL)
26         return NULL;
27
28     /* Inicijalizuju se polja novog cvora */
29     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
30     strcpy(novi_cvor->telefon, telefon);
31     novi_cvor->levo = NULL;
32     novi_cvor->desno = NULL;
33
34     /* Vraca se adresa novog cvora */
35     return novi_cvor;
36 }
37
38 /* Funkcija koja proverava uspesnost kreiranja novog cvora
39    stabla */
40 void proveri_alokaciju(Cvor * novi_cvor)
41 {
42     /* Ukoliko je cvor neuspesno kreiran */
43     if (novi_cvor == NULL) {
44         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
45            programa */
46         fprintf(stderr, "Malloc greska za novi cvor!\n");
47         exit(EXIT_FAILURE);
48     }
49 }
50
51 /* Funkcija koja dodaje novu osobu i njen broj telefona u
52    stablo. */
53 void dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
54                    char *telefon)
55 {
56     /* Ako je stablo prazno */
57     if (*adresa_korena == NULL) {
58         /* Kreira se novi cvor */
59         Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
60         proveri_alokaciju(novi);
61
62         /* I proglašava se korenom stabla */
63         *adresa_korena = novi;
64         return;
65     }

```

```

67      /* U suprotnom trazi se odgovarajuca pozicija za novi unos.
69      Kako pretragu treba vrsiti po imenu i prezimenu, stablo
        treba da bude pretrazivacko po ovom polju */

71
73      /* Ako je zadato ime i prezime leksikografski manje od imena i
        prezimena sadrzanog u korenu, podaci se dodaju u levo
        podstablo */
75      if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
77              < 0)
        dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
79                      telefon);

81      else
        /* Ako je zadato ime i prezime leksikografski vece od imena
83        i prezimena sadrzanog u korenu, podaci se dodaju u desno
        podstablo */
85      if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
        dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
87                      telefon);

89      /* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
91      {
93      /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
        if (*adresa_korena == NULL)
            return;

95
        /* Inace ... */
97      /* Oslobadja se memorija zauzeta levim podstablom */
        oslobodi_stablo(&(*adresa_korena)->levo);

99
        /* Oslobadja se memorija zauzeta desnim podstablom */
101      oslobodi_stablo(&(*adresa_korena)->desno);

103
        /* Oslobadja se memorija zauzeta korenom */
        free(*adresa_korena);

105
        /* Stablo se proglašava praznim */
107      *adresa_korena = NULL;
    }

109
    /* Funkcija koja ispisuje imenik u leksikografskom poretku */
111    /* Napomena: ova funkcija nije trazena u zadatku ali se moze
        koristiti za proveru da li je stablo lepo kreirano ili ne */
113    void prikazi_stablo(Cvor * koren)
    {
115        /* Ako je stablo prazno, završava se sa ispisom */
        if (koren == NULL)
117            return;

```

```
119  /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz
    120      levog podstabla */
121  prikazi_stablo(koren->levo);

122  /* Zatim se ispisuju podaci iz korena */
123  printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
124
125  /* I nastavlja se sa ispisom podataka iz desnog podstabla */
126  prikazi_stablo(koren->desno);
127  }

128
129  /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
    130      ime i prezime i broj telefona u odgovarajuce nizove.
    131      Maksimalna duzina imena i prezimena odredjena je konstantom
    132      MAX_IME_PREZIME, a maksimalna duzina broja telefona
    133      konstantom MAX_CIFARA. Funkcija vraca EOF ako nema vise
    134      kontakata ili 0 u suprotnom. */
135  int procitaj_kontakt(FILE * f, char *ime_i_prezime,
    136                      char *telefon)
137  {
138
139      /* Karakter koji se cita */
140      int c;
141
142      /* Indeks pozicije na koju se smesta procitani karakter */
143      int i = 0;
144
145      /* Linije datoteke koje se obradjuju su formata Ime Prezime
    146          BrojTelefona */
147
148      /* Preskacu se eventualne praznine sa pocetka linije datoteke */
149      while ((c = fgetc(f)) != EOF && isspace(c));
150
151      /* Prvo procitano slovo upisuje se u ime i prezime */
152      if (!feof(f))
153          ime_i_prezime[i++] = c;
154
155      /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
    156          cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
    157          Pritom treba voditi racuna da li ima dovoljno mesta za
    158          smestanje procitanog karaktera i da se slucajno ne dodje do
    159          kraja datoteke */
160      while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
161          if (!isdigit(c))
162              ime_i_prezime[i++] = c;
163
164          else if (i > 0)
165              break;
166      }
167
168      /* Upisuje se terminirajuca nula na mesto poslednjeg
    169          procitanog blanko karaktera */
170      ime_i_prezime[--i] = '\0';
```



```
171  /* I pocinje se sa citanjem broja telefona */
173  i = 0;

175  /* Upisuje se cifra koja je vec procitana */
    telefon[i++] = c;

177
    /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
179     karaktera cije prisustvo nije dozvoljeno u broju telefona */
    while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
181         if (c == '/' || c == '-' || isdigit(c))
            telefon[i++] = c;
183         else
            break;

185
    /* Upisuje se terminirajuca nula */
187    telefon[i] = '\0';

189
    /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
    return !feof(f) ? 0 : EOF;
191 }

193 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
    prezimenom */
195 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
    {
197     /* Ako je imenik prazan, zavrшава se sa pretragom */
        if (koren == NULL)
199             return NULL;

201     /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
        zavrшава sa pretragom */
203     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
        return koren;

205
        /* Ako je zadato ime i prezime leksikografski manje od
207         vrednosti u korenu pretraga se nastavlja levo */
        if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
209             return pretrazi_imenik(koren->levo, ime_i_prezime);

211     else
        /* u suprotnom, pretraga se nastavlja desno */
213         return pretrazi_imenik(koren->desno, ime_i_prezime);
    }

215
217 int main(int argc, char **argv)
    {
        char ime_datoteke[MAX_IME_DATOTEKE];
219        Cvor *koren = NULL;
        Cvor *trazeni;
221        FILE *f;
        char ime_i_prezime[MAX_IME_I_PREZIME];
```

```

223 char telefon[MAX_CIFARA];
224 char c;
225 int i;

227 /* Ucitava se ime datoteke i vrsi se njena priprema za citanje
   */
229 printf("Unesite ime datoteke: ");
230 scanf("%s", ime_datoteke);
231 if ((f = fopen(ime_datoteke, "r")) == NULL) {
232     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
233             ime_datoteke);
234     exit(EXIT_FAILURE);
235 }

237 /* Podaci se citaju iz datoteke i smestaju u binarno stablo
   pretrage. */
239 while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
240     dodaj_u_stablo(&koren, ime_i_prezime, telefon);

241 /* Zatvara se datoteka */
242 fclose(f);

244 /* Omogucava se pretraga imenika */
245 while (1) {
246     /* Ucitavaja se ime i prezime */
247     printf("Unesite ime i prezime: ");
248     i = 0;
249     while ((c = getchar()) != '\n')
250         ime_i_prezime[i++] = c;
251     ime_i_prezime[i] = '\0';

253     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja
       se funkcionalnost */
255     if (strcmp(ime_i_prezime, "KRAJ") == 0)
256         break;

259     /* Inace se ispisuje rezultat pretrage */
260     trazeni = pretrazi_imenik(koren, ime_i_prezime);
261     if (trazeni == NULL)
262         printf("Broj nije u imeniku!\n");
263     else
264         printf("Broj je: %s \n", trazeni->telefon);
265 }

267 /* Oslobadja se memorija zauzeta imenikom */
268 oslobodi_stablo(&koren);

269 return 0;
271 }

```

Rešenje 4.17

```

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

5  #define MAX 51

7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
   studenta, ukupan uspeh, uspeh iz matematike, uspeh iz
9  maternjeg jezika i redom pokazivace na levo i desno podstablo
   */
11 typedef struct cvor_stabla {
    char ime[MAX];
13   char prezime[MAX];
    double uspeh;
15   double matematika;
    double jezik;
17   struct cvor_stabla *levo;
    struct cvor_stabla *desno;
19 } Cvor;

21 /* Funkcija kojom se kreira cvor stabla */
Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
23                  double matematika, double jezik)
{
25   /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
27   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   if (novi == NULL)
29       return NULL;

31   /* Inicijalizuju se polja strukture */
   strcpy(novi->ime, ime);
33   strcpy(novi->prezime, prezime);
   novi->uspeh = uspeh;
35   novi->matematika = matematika;
   novi->jezik = jezik;
37   novi->levo = NULL;
   novi->desno = NULL;

39   /* Vraca se adresa kreiranog cvora */
41   return novi;
}

43 /* Funkcija kojom se proverava uspesnost alociranja memorije */
45 void prover_i_alokaciju(Cvor * novi_cvor)
{
47   /* Ukoliko je cvor neuspesno kreiran */
   if (novi_cvor == NULL) {
49       /* Ispisuje se odgovarajuca poruka i prekida se izvršavanje
          programa */
51       fprintf(stderr, "Malloc greska za novi cvor!\n");

```

```
        exit(EXIT_FAILURE);
53     }
54 }
55
56 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
57 void oslobodi_stablo(Cvor ** koren)
58 {
59     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
60     if (*koren == NULL)
61         return;
62
63     /* Inace ... */
64     /* Oslobadja se memorija zauzeta levim podstablom */
65     oslobodi_stablo(&(*koren)->levo);
66
67     /* Oslobadja se memorija zauzeta desnim podstablom */
68     oslobodi_stablo(&(*koren)->desno);
69
70     /* Oslobadja se memorija zauzeta korenom */
71     free(*koren);
72
73     /* Stablo se proglašava praznim */
74     *koren = NULL;
75 }
76
77 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
78 void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
79                    double uspeh, double matematika,
80                    double jezik)
81 {
82     /* Ako je stablo prazno */
83     if (*koren == NULL) {
84         /* Kreira se novi cvor */
85         Cvor *novi =
86             napravi_cvor(ime, prezime, uspeh, matematika, jezik);
87         prover_i_alokaciju(novi);
88
89         /* I proglašava se korenom stabla */
90         *koren = novi;
91
92         return;
93     }
94
95     /* Inace, dodaje se cvor u stablo tako da bude sortirano po
96     ukupnom broju poena */
97     if (uspeh + matematika + jezik >
98         (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
99         dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
100                      matematika, jezik);
101     else
102         dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
103                      matematika, jezik);
104 }
```

```

105 }
106
107 /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost
108 argumenta polozili jednaka 0 ispisuju se informacije o
109 ucenicima koji nisu polozili prijemni, a ako je vrednost
110 argumenta razlicita od nule, ispisuju se informacije o
111 ucenicima koji su polozili prijemni */
112 void stampaj(Cvor * koren, int polozili)
113 {
114     /* Stablo je prazno - prekida se sa ispisom */
115     if (koren == NULL)
116         return;
117
118     /* Stampaju se informacije iz levog podstabla */
119     stampaj(koren->levo, polozili);
120
121     /* Stampaju se informacije iz korenog cvora */
122     if (polozili && koren->matematika + koren->jezik >= 10)
123         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
124             koren->prezime, koren->uspeh, koren->matematika,
125             koren->jezik, koren->uspeh + koren->matematika + koren->jezik);
126     else if (!polozili && koren->matematika + koren->jezik < 10)
127         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
128             koren->prezime, koren->uspeh, koren->matematika,
129             koren->jezik, koren->uspeh + koren->matematika + koren->jezik);
130
131     /* Stampaju se informacije iz desnog podstabla */
132     stampaj(koren->desno, polozili);
133 }
134
135 /* Funkcija koja odredjuje koliko studenata nije polozilo
136 prijemni ispit */
137 int nisu_polozili(Cvor * koren)
138 {
139     /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
140     if (koren == NULL)
141         return 0;
142
143     /* Pretraga se vrši i u levom i u desnom podstablu - ako uslov
144 za polaganje nije ispunjen za koreni cvor, broj studenata
145 se uvecava za 1 */
146     if (koren->matematika + koren->jezik < 10)
147         return 1 + nisu_polozili(koren->levo) +
148             nisu_polozili(koren->desno);
149
150     return nisu_polozili(koren->levo) +
151         nisu_polozili(koren->desno);
152 }
153
154 int main(int argc, char **argv)

```

```

157 {
    FILE *in;
    Cvor *koren;
159 char ime[MAX], prezime[MAX];
    double uspeh, matematika, jezik;
161
    /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
163 in = fopen("prijemni.txt", "r");
    if (in == NULL) {
165         fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
167         exit(EXIT_FAILURE);
    }
169
    /* Citanje podataka i dodavanje u stablo */
171 koren = NULL;
    while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
173             &matematika, &jezik) != EOF) {
        dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
175                     jezik);
    }
177
    /* Zatvaranje datoteke */
179 fclose(in);

181 /* Stampaju se prvo podaci o ucenicima koji su polozili
    prijemni */
183 stampaj(koren, 1);

185 /* Linij se iscrtava samo ako postoje ucenici koji nisu
    polozili prijemni */
187 if (nisu_polozili(koren) != 0)
    printf("-----\n");
189
    /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
191 stampaj(koren, 0);

193 /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
195
    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
11 typedef struct cvor_stabla {
12     char ime[MAX_NISKA];
13     char prezime[MAX_NISKA];
14     int dan;
15     int mesec;
16     struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
18 } Cvor;
19
20 /* Funkcija koja kreira novi cvor */
21 Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
22 {
23     /* Alocira se memorija */
24     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
25     if (novi == NULL)
26         return NULL;
27
28     /* Inicijalizuju se polja strukture */
29     strcpy(novi->ime, ime);
30     strcpy(novi->prezime, prezime);
31     novi->dan = dan;
32     novi->mesec = mesec;
33     novi->levo = NULL;
34     novi->desno = NULL;
35
36     /* Vraca se adresa novog cvora */
37     return novi;
38
39 /* Funkcija koja proverava uspesnost alokacije */
40 void prover_i_alokaciju(Cvor * novi_cvor)
41 {
42     /* Ako memorija nije uspesno alocirana */
43     if (novi_cvor == NULL) {
44         /* Ispisuje se poruka i prekida se sa izvršavanjem programa */
45         fprintf(stderr, "Malloc greska za novi cvor!\n");
46         exit(EXIT_FAILURE);
47     }
48 }
49
50 /* Funkcija koja oslobadja memoriju zauzetu stablom */
51 void oslobodi_stablo(Cvor ** koren)
52 {
53     /* Stablo je prazno */
54     if (*koren == NULL)
55         return;
56
57     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
58     if ((*koren)->levo)
```

```

59     oslobodi_stablo(&(*koren)->levo);

61     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
62     if ((*koren)->desno)
63         oslobodi_stablo(&(*koren)->desno);

65     /* Oslobadja se memorija zauzeta korenom */
66     free(*koren);

67     /* Proglasava se stablo praznim */
68     *koren = NULL;
69 }

71 /* Funkcija koja dodaje novi cvor u stablo - stablo treba da bude
72    uredjeno po datumu - prvo po mesecu, a zatim po danu */
73 void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
74                    int dan, int mesec)
75 {
76     /* Ako je stablo prazno */
77     if (*koren == NULL) {
78
79         /* Kreira se novi cvor */
80         Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
81         prover_i_alokaciju(novi_cvor);

82         /* I proglasava se korenom */
83         *koren = novi_cvor;

84         return;
85     }

86     /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
87        meseca */
88     if (mesec < (*koren)->mesec)
89         dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
90     else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
91         dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
92     else
93         dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec);
94 }

95 /* Funkcija vrši pretragu stabla i vraća cvor sa traženim datumom.
96    */
97 Cvor *pretraži(Cvor * koren, int dan, int mesec)
98 {
99     /* Stablo je prazno, obustavlja se pretraga */
100    if (koren == NULL)
101        return NULL;

102    /* Ako je traženi datum u korenu */
103    if (koren->dan == dan && koren->mesec == mesec)
104        return koren;
105 }

```



```
111  /* Ako je mesec trazenog datuma manji od meseca sadrzanog u korenu
113     ili ako su meseci isti ali je dan trazenog datuma manji od
115     aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
        svakako proverava da li leva grana postoji - ako ne postoji
        treba vratiti prvi sledeci, a to je bas vrednost uocenog korena
    */
117  if (mesec < koren->mesec
119      || (mesec == koren->mesec && dan < koren->dan)) {
        if (koren->levo == NULL)
121            return koren;
        else
123            return pretrazi(koren->levo, dan, mesec);
    }

125  /* Inace se nastavlja pretraga u desnom delu */
    return pretrazi(koren->desno, dan, mesec);
}

127  /* Funkcija koja pronalazi najmanji datum u stablu */
129  Cvor *pronadji_najmanji_datum(Cvor * koren)
    {
131        /* Stablo je prazno, obustavlja se pretraga */
        if (koren == NULL)
133            return NULL;

135        /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
            sadrzi najmanji datum */
137        if (koren->levo == NULL)
            return koren;
        else
139            /* Inace, trazimo manji datum u levom podstablu */
            return pronadji_najmanji_datum(koren->levo);
    }

143  /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
        */
145  void datum_u_nisku(int dan, int mesec, char datum[])
    {
147        if (dan < 10) {
            datum[0] = '0';
149            datum[1] = dan + '0';
        } else {
151            datum[0] = dan / 10 + '0';
            datum[1] = dan % 10 + '0';
153        }
        datum[2] = '.';

155        if (mesec < 10) {
            datum[3] = '0';
157            datum[4] = mesec + '0';
159        } else {
```

```
161     datum[3] = mesec / 10 + '0';
162     datum[4] = mesec % 10 + '0';
163 }
164 datum[5] = '.';
165 datum[6] = '\\0';
166 }
167 int main(int argc, char **argv)
168 {
169     FILE *in;
170     Cvor *koren;
171     Cvor *slavljenik;
172     char ime[MAX_NISKA], prezime[MAX_NISKA];
173     int dan, mesec;
174     char datum[7];
175
176     /* Provera da li je zadato ime ulazne datoteke */
177     if (argc < 2) {
178         /* Ako nije, ispisuje se poruka i prekida se sa izvršavanjem
179            programa */
180         fprintf(stderr, "Nedostaje ime ulazne datoteke!\\n");
181         exit(EXIT_FAILURE);
182     }
183
184     /* Inace, priprema se datoteka za citanje */
185     in = fopen(argv[1], "r");
186     if (in == NULL) {
187         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\\n",
188             argv[1]);
189         exit(EXIT_FAILURE);
190     }
191
192     /* I stablo se popunjava podacima */
193     koren = NULL;
194     while (fscanf
195         (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
196         dodaj_u_stablo(&koren, ime, prezime, dan, mesec);
197
198     /* Datoteka se zatvara */
199     fclose(in);
200
201     /* Omogucuje se pretraga podataka */
202     while (1) {
203
204         /* Ucitava se novi datum */
205         printf("Unesite datum: ");
206         if (scanf("%d.%d.", &dan, &mesec) == EOF)
207             break;
208
209         /* Pretrazuje se stablo */
210         slavljenik = pretrazi(koren, dan, mesec);
211     }
```

```

213     /* Ispisuju se pronadjeni podaci */
215     /* Ako slavljenik nije pronadjen, to moze znaci da: */
216     /* 1. drvo je prazno */
217     if (slavljenik == NULL && koren == NULL) {
218         printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
219         continue;
220     }
221     /* 2. posle datuma koji je unesen, nema podataka u stablu - u
222     ovom slucaju se pretraga vrši pocevši od naredne godine i
223     ispisuje se najmanji datum */
224     if (slavljenik == NULL) {
225         slavljenik = pronadji_najmanji_datum(koren);
226         datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
227         printf("Slavljenik: %s %s %s\n", slavljenik->ime,
228             slavljenik->prezime, datum);
229         continue;
230     }
231     /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
232     /* 1. Pronadjeni su tacni podaci */
233     if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
234         printf("Slavljenik: %s %s\n", slavljenik->ime,
235             slavljenik->prezime);
236         continue;
237     }
238     /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
239     datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
240     printf("Slavljenik: %s %s %s\n", slavljenik->ime,
241         slavljenik->prezime, datum);
242 }
243
244 /* Oslobadja se memorija zauzeta stablom */
245 oslobodi_stablo(&koren);
246
247 return 0;
248 }
249

```

### Rešenje 4.19

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura kojom se predstavlja cvor binarnog pretrazivackog stabla
5  */
6  typedef struct cvor {
7      int broj;
8      struct cvor *levo, *desno;
9  } Cvor;
10

```

```

12  /* b) Funkcija koja alokira memoriju za novi cvor stabla,
    inicijalizuje polja strukture i vraca pokazivac na novi cvor */
    Cvor *napravi_cvor(int broj);

14

16  /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
    void prover_i_alokaciju(Cvor * novi_cvor);

18

20  /* Funkcija koja dodaje zadati broj u stablo */
    void dodaj_u_stablo(Cvor ** adresa_korena, int broj);

22

24  /* Funkcija koja proverava da li se zadati broj nalazi u stablu */
    Cvor *pretrazi_stablo(Cvor * koren, int broj);

26

28  /* Funkcija koj pronalazi cvor koji sadrzi najmanju vrednost u stablu
    */
    Cvor *pronadji_najmanji(Cvor * koren);

30

32  /* Funkcija koja pronalazi cvor koji sadrzi najveću vrednost u stablu
    */
    Cvor *pronadji_najveci(Cvor * koren);

34

36  /* Funkcija koja brise cvor stabla koji sadrzi zadati broj. */
    void obrisi_element(Cvor ** adresa_korena, int broj);

38

40  /* Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo podstablo
    - Koren - Desno podstablo) */
    void prikazi_stablo(Cvor * koren);

42  /* Funkcija koja oslobadja memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** adresa_korena);

44  #endif

```

```

#include <stdio.h>
#include <stdlib.h>
#include "stabla.h"

Cvor *napravi_cvor(int broj)
{
    /* Alokira se memorija za novi cvor i proverava se uspesnost
    alokacije. */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;
    /* Inicijalizuju se polja novog cvora. */
    novi->broj = broj;
    novi->levo = NULL;
    novi->desno = NULL;
    /* Vraca se adresa novog cvora. */
    return novi;
}

```

```

20 void prover_i_alokaciju(Cvor * novi_cvor)
21 {
22     /* Ukoliko je cvor neuspješno kreiran */
23     if (novi_cvor == NULL) {
24         /* Ispisuje se odgovarajuća poruka i prekida izvršavanje programa */
25         fprintf(stderr, "Malloc greska za novi cvor!\n");
26         exit(EXIT_FAILURE);
27     }
28 }
29
30 void dodaj_u_stablo(Cvor ** koren, int broj)
31 {
32     /* Ako je stablo prazno */
33     if (*koren == NULL) {
34         /* Kreira se novi cvor */
35         Cvor *novi = napravi_cvor(broj);
36         prover_i_alokaciju(novi);
37         /* I proglašava se korenom stabla */
38         *koren = novi;
39         return;
40     }
41     /* U suprotnom se traži odgovarajuća pozicija za zadati broj: */
42
43     /* Ako je zadata vrednost manja od vrednosti korena */
44     if (broj < (*koren)->broj)
45         /* Broj se dodaje u levo podstablo */
46         dodaj_u_stablo(&(*koren)->levo, broj);
47     else
48         /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
49         dodaje u desno podstablo */
50         dodaj_u_stablo(&(*koren)->desno, broj);
51 }
52
53 Cvor *pretrazi_stablo(Cvor * koren, int broj)
54 {
55     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
56     if (koren == NULL)
57         return NULL;
58     /* Ako je trazena vrednost sadržana u korenu */
59     if (koren->broj == broj) {
60         /* Prekida se pretraga */
61         return koren;
62     }
63     /* Inace, ako je broj manji od vrednosti sadržane u korenu */
64     if (broj < koren->broj)
65         /* Pretraga se nastavlja u levom podstablu */
66         return pretrazi_stablo(koren->levo, broj);
67     else
68         /* U suprotnom, pretraga se nastavlja u desnom podstablu */
69         return pretrazi_stablo(koren->desno, broj);
70 }

```

```
72 Cvor *pronadji_najmanji(Cvor * koren)
73 {
74     /* Ako je stablo prazno, prekida se pretraga */
75     if (koren == NULL)
76         return NULL;
77
78     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
79     levo od njega */
80
81     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
82     najmanju vrednost */
83     if (koren->levo == NULL)
84         return koren;
85
86     /* Inace, pretragu treba nastaviti u levom podstablu */
87     return pronadji_najmanji(koren->levo);
88 }
89
90 Cvor *pronadji_najveci(Cvor * koren)
91 {
92     /* Ako je stablo prazno, prekida se pretraga */
93     if (koren == NULL)
94         return NULL;
95
96     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
97     desno od njega */
98
99     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
100     najveću vrednost */
101     if (koren->desno == NULL)
102         return koren;
103
104     /* Inace, pretragu treba nastaviti u desnom podstablu */
105     return pronadji_najveci(koren->desno);
106 }
107
108 void obrisi_element(Cvor ** adresa_korena, int broj)
109 {
110     Cvor *pomocni_cvor = NULL;
111
112     /* Ako je stablo prazno, brisanje nije primenljivo */
113     if (*adresa_korena == NULL)
114         return;
115
116     /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
117     stabla, ona se eventualno nalazi u levom podstablu, pa treba
118     rekursivno primeniti postupak na levo podstablo. Koren ovako
119     modifikovanog stabla je nepromenjen. */
120     if (broj < (*adresa_korena)->broj) {
121         obrisi_element(&(*adresa_korena)->levo, broj);
122         return;
123     }
```

```
124 }
126 /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
128 stabla, ona se eventualno nalazi u desnom podstablu pa treba
130 rekursivno primeniti postupak na desno podstablo. Koren ovako
132 modifikovanog stabla je nepromenjen. */
134 if ((*adresa_korena)->broj < broj) {
136     obrisi_element(&(*adresa_korena)->desno, broj);
138     return;
140 }
142 /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
144 jednaka broju koji se brise (tj. slucaj kada treba obrisati
146 koren) */
148 /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
150 prazno stablo (vraca se NULL) */
152 if ((*adresa_korena)->levo == NULL
154     && (*adresa_korena)->desno == NULL) {
156     free(*adresa_korena);
158     *adresa_korena = NULL;
160     return;
162 }
164 /* Ako koren ima samo levog sina, tada se brisanje vrši tako sto se
166 brise koren, a novi koren postaje levi sin */
168 if ((*adresa_korena)->levo != NULL
170     && (*adresa_korena)->desno == NULL) {
172     pomocni_cvor = (*adresa_korena)->levo;
174     free(*adresa_korena);
176     *adresa_korena = pomocni_cvor;
178     return;
180 }
182 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako sto
184 se brise koren, a novi koren postaje desni sin */
186 if ((*adresa_korena)->desno != NULL
188     && (*adresa_korena)->levo == NULL) {
190     pomocni_cvor = (*adresa_korena)->desno;
192     free(*adresa_korena);
194     *adresa_korena = pomocni_cvor;
196     return;
198 }
200 /* Slucaj kada koren ima oba sina - najpre se potrazi sledbenik
202 korena (u smislu poretka) u stablu. To je upravo po vrednosti
204 najmanji cvor u desnom podstablu. On se moze pronaci npr.
206 funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
208 vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
210 broj koji se brise). Zatim se prosto rekursivno pozove funkcija
212 za brisanje na desno podstablo. S obzirom da u njemu treba
214 obrisati najmanji element, a on zasigurno ima najvise jednog
```

```

176     potomka, jasno je da ce njegovo brisanje biti obavljeno na jedan
177     od jednostavnijih nacina koji su gore opisani. */
178     pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
179     (*adresa_korena)->broj = pomocni_cvor->broj;
180     pomocni_cvor->broj = broj;
181     obrisi_element(&(*adresa_korena)->desno, broj);
182 }

183 void prikazi_stablo(Cvor * koren)
184 {
185     /* Ako je stablo prazno, prekida se ispis */
186     if (koren == NULL)
187         return;

188     /* Prvo se ispisuju svi cvorovi levo od korena */
189     prikazi_stablo(koren->levo);

190     /* Zatim se ispisuje vrednost u korenu */
191     printf("%d ", koren->broj);

192     /* Na kraju se ispisuju svi cvorovi desno od korena */
193     prikazi_stablo(koren->desno);
194 }

195 void oslobodi_stablo(Cvor ** koren)
196 {
197     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
198     if (*koren == NULL)
199         return;
200     /* Inace ... */
201     /* Oslobadja se memorija zauzeta levim podstablom */
202     if ((*koren)->levo)
203         oslobodi_stablo(&(*koren)->levo);
204     /* Oslobadja se memorija zauzeta desnim podstablom */
205     if ((*koren)->desno)
206         oslobodi_stablo(&(*koren)->desno);
207     /* Oslobadja se memorija zauzeta korenom */
208     free(*koren);
209     /* Proglasava se stablo praznim */
210     *koren = NULL;
211 }

```

```

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

3
4 /* Ukljucuje se biblioteka za rad sa stablima - pogledati uvodni
5    zadatak ove glave */
6 #include "stabla.h"

7
8
9 /* Funkcija koja proverava da li su dva stabla koja sadrze cele
10    brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,

```



```
12     odnosno 0 ako nisu */
13 int identitet(Cvor * koren1, Cvor * koren2)
14 {
15     /* Ako su oba stabla prazna, jednaka su */
16     if (koren1 == NULL && koren2 == NULL)
17         return 1;
18
19     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
20     if (koren1 == NULL || koren2 == NULL)
21         return 0;
22
23     /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
24        vrednosti, moze se zakljuciti da se razlikuju */
25     if (koren1->broj != koren2->broj)
26         return 0;
27
28     /* Inace, proverava se da li vazi i jednakost levih i desnih
29        podstabala */
30     return (identitet(koren1->levo, koren2->levo)
31             && identitet(koren1->desno, koren2->desno));
32 }
33
34 int main()
35 {
36     int broj;
37     Cvor *koren1, *koren2;
38
39     /* Ucitavaju se elementi prvog stabla */
40     koren1 = NULL;
41     printf("Prvo stablo: ");
42     scanf("%d", &broj);
43     while (broj != 0) {
44         dodaj_u_stablo(&koren1, broj);
45         scanf("%d", &broj);
46     }
47
48     /* Ucitavaju se elementi drugog stabla */
49     koren2 = NULL;
50     printf("Drugo stablo: ");
51     scanf("%d", &broj);
52     while (broj != 0) {
53         dodaj_u_stablo(&koren2, broj);
54         scanf("%d", &broj);
55     }
56
57     /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
58        njen rezultat. */
59     if (identitet(koren1, koren2))
60         printf("Stabla jesu identicna.\n");
61     else
62         printf("Stabla nisu identicna.\n");
```

```

64  /* Oslobadja se memorija zauzeta stablima */
    oslobodi_stablo(&koren1);
    oslobodi_stablo(&koren2);
66
    return 0;
68 }

```

### Rešenje 4.20

```

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

/* Uklucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija kreira novo stablo identicno stablu koje je dato korenom.
 */
void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
{
    /* Izlaz iz rekurzije */
    if (koren == NULL) {
        *duplikat = NULL;
        return;
    }

    /* Duplira se koren stabla i postavlja da bude koren novog stabla
     */
    *duplikat = napravi_cvor(koren->broj);
    prover_i_alokaciju(*duplikat);

    /* Rekurzivno se duplira levo podstablo i njegova adresa se cuva u
     pokazivacu na levo podstablo korena duplikata. */
    kopiraj_stablo(koren->levo, &(*duplikat)->levo);

    /* Rekurzivno se duplira desno podstablo i njegova adresa se cuva u
     pokazivacu na desno podstablo korena duplikata. */
    kopiraj_stablo(koren->desno, &(*duplikat)->desno);
}

/* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
   dobija modifikacijom prvog stabla */
void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
{
    /* Ako drugo stablo nije prazno */
    if (koren2 != NULL) {
        /* Dodaje se njegov koren u prvo stablo */
        dodaj_u_stablo(adresa_korena1, koren2->broj);

        /* Rekurzivno se racuna unija levog i desnog podstabla drugog
         stabla sa prvim stablom */
        kreiraj_uniju(adresa_korena1, koren2->levo);
    }
}

```

```

42     kreiraj_uniju(adresa_korena1, koren2->desno);
43 }
44 }

46 /* Funkcija izracunava presek dva stabla - rezultujuce stablo se
     dobija modifikacijom prvog stabla */
48 void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
49 {
50     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
51     if (*adresa_korena1 == NULL)
52         return;

54     /* Inace... */
55     /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.
56        iz levog i desnog podstabla prvog stabla brisu se svi oni
57        elementi koji ne postoje u drugom stablu */
58     kreiraj_presek(&(*adresa_korena1)->levo, koren2);
59     kreiraj_presek(&(*adresa_korena1)->desno, koren2);

60     /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on
61        uklanja iz prvog stabla */
62     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
63         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
64 }

66 /* Funkcija izracunava razliku dva stabla - rezultujuce stablo se
     dobija modifikacijom prvog stabla */
68 void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
69 {
70     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
71     if (*adresa_korena1 == NULL)
72         return;

74     /* Inace... */
75     /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
76        tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
77        elementi koji postoje i u drugom stablu */
78     kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
79     kreiraj_razliku(&(*adresa_korena1)->desno, koren2);

82     /* Ako se koren prvog stabla nalazi i u drugom stablu tada se isti
83        uklanja iz prvog stabla */
84     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
85         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
86 }

88 int main()
89 {
90     Cvor *koren1;
91     Cvor *koren2;
92     Cvor *pomocni = NULL;
93     int n;

```

```
94      /* Ucitavaju se elementi prvog stabla */
96      koren1 = NULL;
97      printf("Prvo stablo: ");
98      while (scanf("%d", &n) != EOF) {
99          dodaj_u_stablo(&koren1, n);
100      }

102      /* Ucitavaju se elementi drugog stabla */
103      koren2 = NULL;
104      printf("Drugo stablo: ");
105      while (scanf("%d", &n) != EOF) {
106          dodaj_u_stablo(&koren2, n);
107      }

108      /* Kreira se unija stabala: prvo se napravi kopija prvog stabla
109      kako bi se isto moglo iskoristiti i za preostale operacije */
110      kopiraj_stablo(koren1, &pomocni);
111      kreiraj_uniju(&pomocni, koren2);
112      printf("Unija: ");
113      prikazi_stablo(pomocni);
114      putchar('\n');

116      /* Oslobadja se stablo sa rezultatom operacije */
117      oslobodi_stablo(&pomocni);

120      /* Kreira se presek stabala: prvo se napravi kopija prvog stabla
121      kako bi se isto moglo iskoristiti i za preostale operacije */
122      kopiraj_stablo(koren1, &pomocni);
123      kreiraj_presek(&pomocni, koren2);
124      printf("Presek: ");
125      prikazi_stablo(pomocni);
126      putchar('\n');

128      /* Oslobadja se stablo sa rezultatom operacije */
129      oslobodi_stablo(&pomocni);

130      /* Kreira se razlika stabala: prvo se napravi kopija prvog stabla
131      kako bi se isto moglo iskoristiti i za preostale operacije; */
132      kopiraj_stablo(koren1, &pomocni);
133      kreiraj_razliku(&pomocni, koren2);
134      printf("Razlika: ");
135      prikazi_stablo(pomocni);
136      putchar('\n');

138      /* Oslobadja se stablo sa rezultatom operacije */
139      oslobodi_stablo(&pomocni);

142      /* Oslobadjaju se i polazna stabla */
143      oslobodi_stablo(&koren2);
144      oslobodi_stablo(&koren1);
```

```

146     return 0;
    }

```

### Rešenje 4.21

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
6
7  #define MAX 50
8
9  /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
10     cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11     su smestene u niz. */
12  int kreiraj_niz(Cvor * koren, int a[])
13  {
14     int r, s;
15
16     /* Stablo je prazno - u niz je smesteno 0 elemenata */
17     if (koren == NULL)
18         return 0;
19
20     /* Dodaju se u niz elementi iz levog podstabla */
21     r = kreiraj_niz(koren->levo, a);
22
23     /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
24        u niz i na osnovu nje se moze odrediti indeks novog elementa */
25
26     /* Smesta se vrednost iz korena */
27     a[r] = koren->broj;
28
29     /* Dodaju se elementi iz desnog podstabla */
30     s = kreiraj_niz(koren->desno, a + r + 1);
31
32     /* Racuna se indeks na koji treba smestiti naredni element */
33     return r + s + 1;
34 }
35
36 /* Funkcija sortira niz tako sto najpre elemente niza smesti u
37    stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva u
38    desno.
39
40    Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
41    sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
42    sortiranje vrši u pomocnoj dinamicnoj strukturi, a ne razmenom
43    elemenata niza. */
44 void sortiraj(int a[], int n)
45 {
46     int i;

```

```

47  Cvor *koren;

49  /* Kreira se stablo smestanjem elemenata iz niza u stablo */
   koren = NULL;
51  for (i = 0; i < n; i++)
       dodaj_u_stablo(&koren, a[i]);

53

55  /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
   a */
   kreiraj_niz(koren, a);

57

59  /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
   */
   oslobodi_stablo(&koren);
}

61
63  int main()
64  {
65     int a[MAX];
66     int n, i;

67     /* Ucitavaju se dimenzija i elementi niza */
68     printf("n: ");
69     scanf("%d", &n);
70     if (n < 0 || n > MAX) {
71         printf("Greska: pogresna dimenzija niza!\n");
72         return 0;
73     }

74
75     printf("a: ");
76     for (i = 0; i < n; i++)
77         scanf("%d", &a[i]);

78
79     /* Poziva se funkcija za sortiranje */
80     sortiraj(a, n);

81
82     /* Ispisuje se rezultat */
83     for (i = 0; i < n; i++)
84         printf("%d ", a[i]);
85     printf("\n");

86
87     return 0;
88 }

```

### Rešenje 4.22

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
   #include "stabla.h"

```

```
6
/* a) Funkcija koja izracunava broj cvorova stabla */
8 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
        zato sto treba racunati i koren */
16     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 */
24     if (koren == NULL)
        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;

32     /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
        */
34     return broj_listova(koren->levo) + broj_listova(koren->desno);
36 }

38 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
void pozitivni_listovi(Cvor * koren)
40 {
    /* Slucaj kada je stablo prazno */
42     if (koren == NULL)
        return;

44     /* Ako je cvor list i sadrzi pozitivnu vrednost */
    if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
46     /* Stampa se */
        printf("%d ", koren->broj);

48     /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
    pozitivni_listovi(koren->levo);
    pozitivni_listovi(koren->desno);
52 }

54
/* d) Funkcija koja izracunava zbir cvorova stabla */
56 int zbir_svih_cvorova(Cvor * koren)
{
```

```
58  /* Ako je stablo prazno, zbir cvorova je 0 */
    if (koren == NULL)
60      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 vece od korena se
    nalaze u desnom podstablu */
76
    /* Ako desnog podstabla nema */
78  if (koren->desno == NULL)
    /* Najveća vrednost je koren */
80    return koren;
82
    /* Inace, najveća vrednost se trazi desno */
84  return najveci_element(koren->desno);
}

86
/* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
{
90  /* Dubina praznog stabla je 0 */
    if (koren == NULL)
92      return 0;

94  /* Izracunava se dubina levog podstabla */
    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 */
int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
108 {
    /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazeneog
```



```
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
121     /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
122        */
123     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
124         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
125 }
126
127 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
128 void ispis_nivo(Cvor * koren, int i)
129 {
130     /* Ideja je slicna ideji iz prethodne funkcije */
131
132     /* Nema vise cvorova, nema spustanja kroz stablo */
133     if (koren == NULL)
134         return;
135
136     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
137     if (i == 0) {
138         printf("%d ", koren->broj);
139         return;
140     }
141     /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
142        desnom podstablu */
143     ispis_nivo(koren->levo, i - 1);
144     ispis_nivo(koren->desno, i - 1);
145 }
146
147 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
148    stabla */
149 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
150 {
151     /* Ako je stablo prazno, obustavlja se pretraga */
152     if (koren == NULL)
153         return NULL;
154
155     /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
156     if (i == 0)
157         return koren;
158
159     /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
160     Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
```

```

162  /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
    Cvor *b = najveći_element_na_itom_nivou(koren->desno, i - 1);

164  /* Trazi se i vraća maksimum izracunatih vrednosti */
    if (a == NULL && b == NULL)
166      return NULL;
    if (a == NULL)
168      return b;
    if (b == NULL)
170      return a;
    return a->broj > b->broj ? a : b;
172 }

174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
    int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
    /* Ako je stablo prazno, zbir je nula */
178     if (koren == NULL)
        return 0;

180     /* Ako se stiglo do traženog nivoa, vraća se vrednost */
182     if (i == 0)
        return koren->broj;

184     /* Inace, spustanje se nastavlja za jedan nivo nize i traže se sume
186     iz levog i desnog podstabla */
    return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
188         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
190 }

192 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
    manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
    {
196     /* Ako je stablo prazno, zbir je nula */
    if (koren == NULL)
198         return 0;

200     /* Ako je vrednost u korenu manja od tražene vrednosti, zbog
    prirode pretraživackog stabla treba obići i levo i desno
202     podstablo */
    if (koren->broj <= x)
204         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
            zbir_manjih_od_x(koren->desno, x);

206     /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
    medju njima jedino može biti onih koje zadovoljavaju uslov */
208     return zbir_manjih_od_x(koren->levo, x);
210 }

212 int main(int argc, char **argv)

```

```

214 {
215     /* Analiza argumenata komandne linije */
216     if (argc != 3) {
217         fprintf(stderr,
218             "Greska! Program se poziva sa: ./a.out nivo
219             broj_zapretragu\n");
220         exit(EXIT_FAILURE);
221     }
222     int i = atoi(argv[1]);
223     int x = atoi(argv[2]);
224
225     /* Kreira se stablo */
226     Cvor *koren = NULL;
227     int broj;
228     while (scanf("%d", &broj) != EOF)
229         dodaj_u_stablo(&koren, broj);
230
231     /* ispisuju se rezultati rada funkcija */
232     printf("Broj cvorova: %d\n", broj_cvorova(koren));
233     printf("Broj listova: %d\n", broj_listova(koren));
234     printf("Pozitivni listovi: ");
235     pozitivni_listovi(koren);
236     printf("\n");
237     printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
238     if (najveci_element(koren) == NULL)
239         printf("Najveci element: ne postoji\n");
240     else
241         printf("Najveci element: %d\n", najveci_element(koren)->broj);
242
243     printf("Dubina stabla: %d\n", dubina_stabla(koren));
244
245     printf("Broj cvorova na %d. nivou: %d\n", i,
246         broj_cvorova_na_itom_nivou(koren, i));
247     printf("Elementi na %d. nivou: ", i);
248     ispis_nivo(koren, i);
249     printf("\n");
250     if (najveci_element_na_itom_nivou(koren, i) == NULL)
251         printf("Nema elemenata na %d. nivou!\n", i);
252     else
253         printf("Maksimalni element na %d. nivou: %d\n", i,
254             najveci_element_na_itom_nivou(koren, i)->broj);
255
256     printf("Zbir elemenata na %d. nivou: %d\n", i,
257         zbir_cvorova_na_itom_nivou(koren, i));
258     printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
259         zbir_manjih_od_x(koren, x));
260
261     /* Oslobadja se memorija zauzeta stablom */
262     oslobodi_stablo(&koren);
263
264     return 0;
265 }

```

## Rešenje 4.23

```

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

4  /* Uključuje se biblioteka za rad sa stablima */
   #include "stabla.h"

6  /* Funkcija koja izracunava dubinu stabla */
8  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)
32         return;

34     /* Ako se stiglo do trazene nivoa - ispisuje se vrednost */
       if (i == 0) {
36         printf("%d ", koren->broj);
           return;
38     }

       /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
       podstablu */
40     ispisi_nivo(koren->levo, i - 1);
       ispisi_nivo(koren->desno, i - 1);
42 }

44

46 /* Funkcija koja ispisuje stablo po nivoima */
   void ispisi_stablo_po_nivoima(Cvor * koren)
   {
48     int i;

50     /* Prvo se izracunava dubina stabla */

```

```

52     int dubina;
    dubina = dubina_stabla(koren);

54     /* Ispisuje se nivo po nivo stabla */
    for (i = 0; i < dubina; i++) {
56         printf("%d. nivo: ", i);
        ispisi_nivo(koren, i);
58         printf("\n");
    }
60 }

62 int main(int argc, char **argv)
{
64     Cvor *koren;
    int broj;

66     /* Citaju se vrednosti sa ulaza i dodaju se u stablo */
    koren = NULL;
    while (scanf("%d", &broj) != EOF) {
68         dodaj_u_stablo(&koren, broj);
70     }

72     /* Ispisuje se stablo po nivoima */
    ispisi_stablo_po_nivoima(koren);

74     /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);

76     return 0;
78 }
80 }

```

### Rešenje 4.25

```

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

3

   /* Ukljucuje 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
20         jer se racuna i koren */
21      return dubina_levo >
22             dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
23  }
24
25  /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
26     stablo */
27  int avl(Cvor * koren)
28  {
29      int dubina_levo, dubina_desno;
30
31      /* Ako je stablo prazno, zaustavlja se brojanje */
32      if (koren == NULL) {
33          return 0;
34      }
35
36      /* Izracunava se dubina levog podstabla korena */
37      dubina_levo = dubina_stabla(koren->levo);
38
39      /* Izracunava se dubina desnog podstabla korena */
40      dubina_desno = dubina_stabla(koren->desno);
41
42      /* Ako je uslov za AVL stablo ispunjen */
43      if (abs(dubina_desno - dubina_levo) <= 1) {
44          /* Racuna se broj AVL cvorova u levom i desnom podstablu i
45             uvecava za jedan iz razloga sto koren ispunjava uslov */
46          return 1 + avl(koren->levo) + avl(koren->desno);
47      } else {
48          /* Inace, racuna se samo broj AVL cvorova u podstablama */
49          return avl(koren->levo) + avl(koren->desno);
50      }
51  }
52
53  int main(int argc, char **argv)
54  {
55      Cvor *koren;
56      int broj;
57
58      /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo */
59      koren = NULL;
60      while (scanf("%d", &broj) != EOF) {
61          dodaj_u_stablo(&koren, broj);
62      }
63
64      /* Racuna se i ispisuje broj AVL cvorova */
65      printf("%d\n", avl(koren));
66
67      /* Oslobadja se memorija zauzeta stablom */
68      oslobodi_stablo(&koren);
69  }
```

```

71     return 0;
    }

```

### Rešenje 4.26

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija proverava da li je zadato binarno stablo celih
8     pozitivnih brojeva hip. Ideja koja ce biti implementirana u
9     osnovi ima pronalazenje maksimalne vrednosti levog i
10    maksimalne vrednosti desnog podstabla - ako je vrednost u
11    korenu veca od izracunatih vrednosti, uoceni fragment stabla
12    zadovoljava uslov za hip. Zato ce funkcija vratiti maksimalne
13    vrednosti iz uocenog podstabala ili vrednost -1 ukoliko se
14    zakljuci da stablo nije hip. */
15    int heap(Cvor * koren)
16    {
17        int max_levo, max_desno;
18
19        /* Prazno sablo je hip - kao rezultat se vraca 0 kao najmanji
20           pozitivan broj */
21        if (koren == NULL) {
22            return 0;
23        }
24        /* Ukoliko je stablo list... */
25        if (koren->levo == NULL && koren->desno == NULL) {
26            /* Vraca se njegova vrednost */
27            return koren->broj;
28        }
29        /* Inace... */
30
31        /* Proverava se svojstvo za levo podstablo. */
32        max_levo = heap(koren->levo);
33
34        /* Proverava se svojstvo za desno podstablo. */
35        max_desno = heap(koren->desno);
36
37        /* Ako levo ili desno podstablo uocenog cvora nije hip, onda
38           nije ni celo stablo. */
39        if (max_levo == -1 || max_desno == -1) {
40            return -1;
41        }
42
43        /* U suprotnom proverava se da li svojstvo vazi za uoceni
44           cvor. */
45        if (koren->broj > max_levo && koren->broj > max_desno) {
46            /* Ako vazi, vraca se vrednost korena */

```

```
48     return koren->broj;
50 }
52 /* U suprotnom zaključuje se da stablo nije hip */
53 return -1;
54 }
55
56 int main(int argc, char **argv)
57 {
58     Cvor *koren;
59     int hip_indikator;
60
61     /* Kreira se stablo koje sadrži brojeve 100 19 36 17 3 25 1 2
62        7 */
63     koren = NULL;
64     koren = napravi_cvor(100);
65     koren->levo = napravi_cvor(19);
66     koren->levo->levo = napravi_cvor(17);
67     koren->levo->levo->levo = napravi_cvor(2);
68     koren->levo->levo->desno = napravi_cvor(7);
69     koren->levo->desno = napravi_cvor(3);
70     koren->desno = napravi_cvor(36);
71     koren->desno->levo = napravi_cvor(25);
72     koren->desno->desno = napravi_cvor(1);
73
74     /* Poziva se funkcija kojom se proverava da li je stablo hip */
75     hip_indikator = heap(koren);
76
77     /* Ispisuje se rezultat */
78     if (hip_indikator == -1) {
79         printf("Zadato stablo nije hip!\n");
80     } else {
81         printf("Zadato stablo je hip!\n");
82     }
83
84     /* Oslobadja se memorija zauzeta stablom. */
85     oslobodi_stablo(&koren);
86
87     return 0;
88 }
```



## Glava 5

# Ispitni rokovi

### 5.1 Programiranje 2, praktični deo ispita, jun 2015.

**Zadatak 5.1** Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost -1 i prekinuti izvršavanje programa.

#### *Test 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.

Test 1	Test 2	Test 3
<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>	<pre> ULAZ: 2 3 0 0 -5 1 2 -4 IZLAZ: 2 </pre>	<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>Test 1</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>Test 2</p> <pre> Poziv: ./a.out IZLAZ: (na stderr) -1 </pre>
<p>Test 3</p> <pre> Poziv: ./a.out ulaz.txt foo DATOTEKA ULAZ.TXT NE POSTOJI IZLAZ: (na stderr) -1 </pre>	<p>Test 4</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 brojeba. Napisati funkciju `int f3(Cvor *koren, int n)` koja u datom stablu prebrojava čvorove na  $n$ -tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou 0. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

```
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
```

<i>Test 4</i>	<i>Test 5</i>
ULAZ: 3	ULAZ: -1 4 5 1 7
IZLAZ: 0	IZLAZ: 0

[Rešenje 5.6]

### 5.3 Programiranje 2, praktični deo ispita, septembar 2015.

[illegible]

<p><i>Test 1</i></p> <pre>    ULAZ:      6 1    IZLAZ:      3 </pre>	<p><i>Test 2</i></p> <pre>    ULAZ:      15 3    IZLAZ:    3758096385 </pre>	<p><i>Test 3</i></p> <pre>    ULAZ:      31 100    IZLAZ:    4026531841 </pre>
<p><i>Test 4</i></p> <pre>    ULAZ:      4 0    IZLAZ:      4 </pre>	<p><i>Test 5</i></p> <pre>    ULAZ:      0 5    IZLAZ:      0 </pre>	

[Rešenje 5.7]

**Zadatak 5.8** Napisati funkciju `void 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. 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.

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
5
6 void greska()
7 {
8     printf("-1\n");
9     exit(EXIT_FAILURE);
10 }
11
12 int main(int argc, char *argv[])
13 {
14     FILE *ulaz;
15     char **linije;
16     int i, j, n;
17
18     /* Proverava argumenata komandne linije. */
19     if (argc != 2) {
20         greska();
21     }
22
23     /* Otvaranje datoteke cije ime je navedeno kao argument komandne
24        linije neposredno nakon imena programa koji se poziva. */
25     ulaz = fopen(argv[1], "r");
26     if (ulaz == NULL) {
27         greska();
28     }
29
30     /* Ucitavanje broja linija. */
31     fscanf(ulaz, "%d", &n);
32
33     /* Alociranje memorije na osnovu ucitanog broja linija. */
34     linije = (char **) malloc(n * sizeof(char *));
35     if (linije == NULL) {
36         greska();
37     }
38     for (i = 0; i < n; i++) {
39         linije[i] = malloc(MAX * sizeof(char));
40         if (linije[i] == NULL) {
41             for (j = 0; j < i; j++) {
42                 free(linije[j]);
43             }
44             free(linije);
45             greska();
46         }
```

```

    }
48 }

50 /* Ucitavanje svih n linija iz datoteke. */
51 for (i = 0; i < n; i++) {
52     fscanf(ulaz, "%s", linije[i]);
53 }
54
55 /* Ispisivanje u odgovarajucem poretку ucitane linije koje
56    zadovoljavaju kriterijum. */
57 for (i = n - 1; i >= 0; i--) {
58     if (isupper(linije[i][0])) {
59         printf("%s\n", linije[i]);
60     }
61 }
62
63 /* Oslobadjanje memorije koja je dinamicki alocirana. */
64 for (i = 0; i < n; i++) {
65     free(linije[i]);
66 }
67
68 free(linije);
69
70 /* Zatvaranje datoteku. */
71 fclose(ulaz);
72
73 return 0;
74 }

```

## Rešenje 5.2

```

1 /* stabla.h */
2
3 #ifndef __STABLA_H__
4 #define __STABLA_H__ 1
5
6 /* Struktura kojom se predstavlja Cvor stabla */
7 typedef struct dcvor {
8     int broj;
9     struct dcvor *levo, *desno;
10 } Cvor;
11
12 /* Funkcija alokira prostor za novi Cvor stabla, inicijalizuje polja
13    strukture i vraća pokazivac na nov Cvor */
14 Cvor *napravi_cvor(int b);
15
16 /* Funkcija oslobadja dinamicki alocirani prostor za stablo Nakon
17    oslobadjanja se u pozivajucoj funkciji koren postavlja NULL, jer
18    je stablo prazno */
19 void oslobodi_stablo(Cvor ** adresa_korena);
20

```



```

22  /* Funkcija proverava da li je novi Cvor ispravno alokiran, i nakon
    toga prekida program */
    void prover_i_alokaciju(Cvor * novi);
24
    /* Funkcija dodaje nov Cvor u stablo i azurira vrednost korena stabla
    u pozivajucoj funkciji. */
26  void dodaj_u_stablo(Cvor ** adresa_korena, int broj);
28
    #endif

```

```

1  /* stabla.c */
3  #include <stdio.h>
    #include <stdlib.h>
5  #include "stabla.h"
7  Cvor *napravi_cvor(int b)
    {
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
        if (novi == NULL)
11         return NULL;
13
        /* Inicijalizacija polja novog Cvora */
        novi->broj = b;
15         novi->levo = NULL;
        novi->desno = NULL;
17
        return novi;
19     }
21
    void oslobodi_stablo(Cvor ** adresa_korena)
    {
23         /* Prazno stablo i nema sta da se oslobadja */
        if (*adresa_korena == NULL)
25         return;
27
        /* Rekurzivno se oslobadja najpre levo, a onda i desno podstablo */
        if ((*adresa_korena)->levo)
29         oslobodi_stablo(&(*adresa_korena)->levo);
        if ((*adresa_korena)->desno)
31         oslobodi_stablo(&(*adresa_korena)->desno);
33
        free(*adresa_korena);
        *adresa_korena = NULL;
35     }
37
    void prover_i_alokaciju(Cvor * novi)
    {
39         if (novi == NULL) {
            fprintf(stderr, "Malloc greska za nov cvor!\n");
41         exit(EXIT_FAILURE);
        }
    }

```

```

43 }
45 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
46 {
47     /* Postojeće stablo je prazno */
48     if (*adresa_korena == NULL) {
49         Cvor *novi = napravi_cvor(broj);
50         prover_i_alokaciju(novi);
51         /* Kreirani Cvor novi će biti od sada koren stabla */
52         *adresa_korena = novi;
53         return;
54     }
55
56     /* Brojevi se smestaju u uredjeno binarno stablo, pa ako je broj
57        koji se ubacuje manji od broja koji je u korenu onda se dodaje u
58        levo podstablo. */
59     if (broj < (*adresa_korena)->broj)
60         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
61     /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
62        dodaje se nov Cvor desno od korena. */
63     else
64         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
65 }

```

```

1  /* main.c */
2
3  #include <stdio.h>
4  #include "stabla.h"
5
6
7  int sumirajN(Cvor * koren, int n)
8  {
9      if (koren == NULL)
10         return 0;
11
12     if (n == 0)
13         return koren->broj;
14
15     return sumirajN(koren->levo, n - 1) + sumirajN(koren->desno, n - 1)
16         ;
17 }
18
19 int main()
20 {
21     Cvor *koren = NULL;
22     int n;
23     int nivo;
24
25     scanf("%d", &nivo);
26
27     while (1) {
28         scanf("%d", &n);
29         /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */

```

```

29     if (n == 0)
        break;

31     /* Ako nije, dodaje se procitani broj u stablo. */
    dodaj_u_stablo(&koren, n);
33 }

35 /* Ispisuje se rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren, nivo));
37
39 /* Oslobadja se memorija */
    oslobodi_stablo(&koren);

41 return 0;
}

```

### Rešenje 5.3

```

#include <stdio.h>
2 #define MAX 50

4 int main()
{
6     int m[MAX][MAX];
    int v, k;
8     int i, j;
    int max_broj_negativnih, max_indeks_kolone;
10    int broj_negativnih;

12    /* Ucitavanje dimenzije matrice */
    scanf("%d", &v);
14    if (v < 0 || v > MAX) {
        fprintf(stderr, "-1\n");
16        return 0;
    }

18    scanf("%d", &k);
20    if (k < 0 || k > MAX) {
        fprintf(stderr, "-1\n");
22        return 0;
    }

24    /* Ucitavanje elemenata matrice */
26    for (i = 0; i < v; i++) {
        for (j = 0; j < k; j++) {
28            scanf("%d", &m[i][j]);
        }
30    }

32    /* Pronalazenje kolone koja sadrzi najveći broj negativnih
        elemenata */

```

```

34     max_indeks_kolone = 0;

36     max_broj_negativnih = 0;
    for (i = 0; i < v; i++) {
38         if (m[i][0] < 0) {
            max_broj_negativnih++;
40         }

42     }

44     for (j = 0; j < k; j++) {
        broj_negativnih = 0;
46         for (i = 0; i < v; i++) {
            if (m[i][j] < 0) {
48                 broj_negativnih++;
            }
50             if (broj_negativnih > max_broj_negativnih) {
                max_indeks_kolone = j;
52             }
        }
54     }

56     /* Ispisivanje trazenog rezultata */
    printf("%d\n", max_indeks_kolone);

60     return 0;
}

```

### 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     if (argc != 3) {
        fprintf(stderr, "-1\n");
14         exit(EXIT_FAILURE);
    }

16     /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18         linije */
    if ((f = fopen(argv[1], "r")) == NULL) {
20         fprintf(stderr, "-1\n");
    }

```

```

    exit(EXIT_FAILURE);
22 }

24 while (fgets(linija, MAX, f) != NULL) {
    p = linija;
26     while (1) {
        p = strstr(p, argv[2]);
28         if (p == NULL)
            break;
30         brojac++;
        p = p + strlen(argv[2]);
32     }
    }

34 fclose(f);

36 printf("%d\n", brojac);

38 return 0;
40 }

```

### Rešenje 5.5

```

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

5  /* Struktura trougao */
   typedef struct _trougao {
7      double xa, ya, xb, yb, xc, yc;
   } trougao;

9

11 /* Funkcija racuna duzinu duzi */
   double duzina(double x1, double y1, double x2, double y2)
   {
13     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
   }

15

17 /* Funkcija racuna povrsinu trougla */
   double povrsina(trougao t)
   {
19     double a = duzina(t.xb, t.yb, t.xc, t.yc);
       double b = duzina(t.xa, t.ya, t.xc, t.yc);
21     double c = duzina(t.xa, t.ya, t.xb, t.yb);
       double s = (a + b + c) / 2;
23     return sqrt(s * (s - a) * (s - b) * (s - c));
   }

25

27 /* Funkcija racuna poredi dva trougla, napisana tako da se moze
   proslediti funkciji qsort */
   int poredi(const void *a, const void *b)

```

```
29 {
30     trougao x = *(trougao *) a;
31     trougao y = *(trougao *) b;
32     double xp = povrsina(x);
33     double yp = povrsina(y);
34     if (xp < yp)
35         return 1;
36     if (xp > yp)
37         return -1;
38     return 0;
39 }
40
41 int main()
42 {
43     FILE *f;
44     int n, i;
45     trougao *niz;
46
47     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
48         fprintf(stderr, "-1\n");
49         exit(EXIT_FAILURE);
50     }
51
52     if (fscanf(f, "%d", &n) != 1) {
53         fprintf(stderr, "-1\n");
54         exit(EXIT_FAILURE);
55     }
56
57     if ((niz = malloc(n * sizeof(trougao))) == NULL) {
58         fprintf(stderr, "-1\n");
59         exit(EXIT_FAILURE);
60     }
61
62     for (i = 0; i < n; i++) {
63         if (fscanf(f, "%lf%lf%lf%lf%lf%lf",
64                 &niz[i].xa, &niz[i].ya,
65                 &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
66         {
67             fprintf(stderr, "-1\n");
68             exit(EXIT_FAILURE);
69         }
70     }
71
72     qsort(niz, n, sizeof(trougao), &poredi);
73
74     for (i = 0; i < n; i++)
75         printf("%g %g %g %g %g %g\n",
76             niz[i].xa, niz[i].ya,
77             niz[i].xb, niz[i].yb, niz[i].xc, niz[i].yc);
78
79     free(niz);
80     fclose(f);
81 }
```

```

81     return 0;
    }

```

## Rešenje 5.6

```

/* stabla.h */
2
#ifdef __STABLA_H__
4 #define __STABLA_H__ 1

6 /* Struktura koja predstavlja cvor stabla, sadrzi vrednost koja se
   cuva i pokazivace na levo i desno podstablo. */
8 typedef struct cvor {
    int vrednost;
10    struct cvor *levi;
    struct cvor *desni;
12 } Cvor;

14 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira dinamicki,
   funkcijom malloc(). U slucaju greske program se prekida i ispisuje
   se poruka o gresci. U slucaju uspeha inicijalizuje se vrednost
16 datim brojem, a pokazivaci na podstabla se inicijalizuju na NULL.
   Funkcija vraća adresu novokreiranog cvora */
18 Cvor *napravi_cvor(int broj);

20 /* Funkcija dodaje novi cvor u stablo sa datim korenom. Ukoliko broj
   vec postoji u stablu, ne radi nista. Cvor se kreira funkcijom
22 napravi_cvor(). */
24 void dodaj_u_stablo(Cvor ** koren, int broj);

26 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje elemente u
   rastućem poretku) */
28 void prikazi_stablo(Cvor * koren);

30 /* Funkcija učitava stablo sa standardnog ulaza do kraja ulaza i
   vraća pokazican na njegov koren */
32 Cvor *ucitaj_stablo();

34 /* Funkcija oslobadja prostor koji je alociran za cvorove stabla. */
void oslobodi_stablo(Cvor ** koren);
36 #endif

1 /* stabla.c */
3 #include <stdio.h>
#include <stdlib.h>
5 #include "stabla.h"

```

```
7 Cvor *napravi_cvor(int broj)
{
9     /* Dinamicki kreiramo cvor */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

11
    /* U slucaju greske ... */
13     if (novi == NULL) {
        fprintf(stderr, "-1\n");
15         exit(1);
    }

17
    /* Inicijalizacija */
19     novi->vrednost = broj;
    novi->levi = NULL;
21     novi->desni = NULL;

23     /* Vracamo adresu novog cvora */
    return novi;
25 }

27 void dodaj_u_stablo(Cvor ** koren, int broj)
{
29     /* Izlaz iz rekurzije: ako je stablo bilo prazno, novi koren je
        upravo novi cvor */
    if (*koren == NULL) {
31         *koren = napravi_cvor(broj);
33         return;
    }

35
    /* Ako je stablo neprazno, i koren sadrzi manju vrednost od datog
        broja, broj se umece u desno podstablo, rekurzivnim pozivom */
37     if ((*koren)->vrednost < broj)
        dodaj_u_stablo(&(*koren)->desni, broj);
    /* Ako je stablo neprazno, i koren sadrzi vecu vrednost od datog
        broja, broj se umece u levo podstablo, rekurzivnim pozivom */
41     else if ((*koren)->vrednost > broj)
        dodaj_u_stablo(&(*koren)->levi, broj);
43 }

45
47 void prikazi_stablo(Cvor * koren)
{
    /* Izlaz iz rekurzije */
49     if (koren == NULL)
        return;

51
    prikazi_stablo(koren->levi);
    printf("%d ", koren->vrednost);
    prikazi_stablo(koren->desni);
55 }

57 Cvor *ucitaj_stablo()
{
```



```
59   Cvor *koren = NULL;
    int x;
61   while (scanf("%d", &x) == 1)
        dodaj_u_stablo(&koren, x);
63   return koren;
    }

65   void oslobodi_stablo(Cvor ** koren)
67   {
        /* Izlaz iz rekurzije */
69         if (*koren == NULL)
                return;

71         oslobodi_stablo(&(*koren)->levi);
73         oslobodi_stablo(&(*koren)->desni);
        free(*koren);

75         *koren = NULL;
77   }

1   /* main.c */

3   #include <stdio.h>
    #include "stabla.h"

5

7   int f3(Cvor * koren, int n)
    {
9       if (koren == NULL || n < 0)
                return 0;
        if (n == 0) {
11            if (koren->levi == NULL && koren->desni != NULL)
                    return 1;
13            if (koren->levi != NULL && koren->desni == NULL)
                    return 1;
15            return 0;
        }
17        return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
    }

19

21   int main()
    {
23         Cvor *koren;
        int n;

25         scanf("%d", &n);
        koren = ucitaj_stablo();

27         printf("%d\n", f3(koren, n));

29         oslobodi_stablo(&koren);

31         return 0;
```

```
33 }
```

### Rešenje 5.7

```

#include <stdio.h>

2 unsigned int Rotiraj(unsigned int x, unsigned int n)
4 {
    int i;
    unsigned int maska = 1;

    /* Formiranje maske sa n jedinica na kraju 000...00001111 */
    for (i = 1; i < n; i++)
10     maska = (maska << 1) | 1;

12     return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
    }

14
16 int main()
17 {
    unsigned int x, n;

18     scanf("%u%u", &x, &n);

20     printf("%u\n", Rotiraj(x, n));

22     return 0;
24 }
```

### Rešenje 5.8

```

/* liste.h */

2
3 #ifndef __LISTE_H__
4 #define __LISTE_H__ 1

6 /* Struktura koja predstavlja cvor liste */
7 typedef struct cvor {
8     double vrednost;
9     struct cvor *sledeci;
10 } Cvor;

12 /* Pomocna funkcija koja kreira cvor. */
13 Cvor *napravi_cvor(double broj);

14
15 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
16    ciji se pocetni cvor nalazi na adresi adresa_glave. */
17 void oslobodi_listu(Cvor ** adresa_glave);
18
```

```

20  /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
    zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
22  adresa_glave. */
    void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi);
24
    /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL kao je lista prazna */
26  Cvor *pronadji_poslednji(Cvor * glava);
28
    /* Funkcija dodaje novi cvor na kraj liste. */
30  void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);
32
    /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
    void ispisi_listu(Cvor * glava);
34
    /* Funkcija koja dopunjuje listu na nacin opisan u zadatku */
36  void dopuni_listu(Cvor ** adresa_glave);
38  #endif

```

```

/* liste.c */
2
#include <stdio.h>
4  #include <stdlib.h>
    #include "liste.h"
6
/* Pomocna funkcija koja kreira cvor. */
8  Cvor *napravi_cvor(double broj)
    {
10     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
        if (novi == NULL)
12         return NULL;

        /* inicijalizacija polja u novom cvoru */
14     novi->vrednost = broj;
        novi->sledeci = NULL;
16

18     return novi;
    }
20

/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
    ciji se pocetni cvor nalazi na adresi adresa_glave. */
22  void oslobodi_listu(Cvor ** adresa_glave)
24  {
        Cvor *pomocni = NULL;
26

        while (*adresa_glave != NULL) {
28             pomocni = (*adresa_glave)->sledeci;
                free(*adresa_glave);
30             *adresa_glave = pomocni;

```

```
    }
32 }

34 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
   ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
36 zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
   adresa_glave. */
38 void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
   {
40     /* Ukoliko je novi NULL */
   if (novi == NULL) {
42         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
         oslobodi_listu(adresa_glave);
44         exit(EXIT_FAILURE);
   }
46 }

48 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
50 Cvor *pronadji_poslednji(Cvor * glava)
   {
52     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
       funkcija vraca NULL. */
54     if (glava == NULL)
       return NULL;

56     while (glava->sledeci != NULL)
58         glava = glava->sledeci;

60     return glava;
   }

62 /* Funkcija dodaje novi cvor na kraj liste. */
64 void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
   {
66     Cvor *novi = napravi_cvor(broj);
     prover_i_alokaciju(adresa_glave, novi);

68     if (*adresa_glave == NULL) {
70         *adresa_glave = novi;
       return;
72     }

74     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
     poslednji->sledeci = novi;
76 }

78 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
   */
   void ispisi_listu(Cvor * glava)
   {
80     for (; glava != NULL; glava = glava->sledeci)
```

```

82     printf("%.21f ", glava->vrednost);
84     putchar('\n');
86 }
88 /* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka */
89 void dopuni_listu(Cvor ** adresa_glave)
90 {
91     Cvor *tekuci;
92     Cvor *novi;
93     double aritmeticka_sredina;
94     if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
95         return;
96
97     tekuci = *adresa_glave;
98     while (tekuci->sledeci != NULL) {
99         aritmeticka_sredina =
100             ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
101         novi = napravi_cvor(aritmeticka_sredina);
102         prover_i_alokaciju(adresa_glave, novi);
103
104         novi->sledeci = tekuci->sledeci;
105         tekuci->sledeci = novi;
106         tekuci = tekuci->sledeci;
107     }
108
109     return;
110 }

```

```

/* main.c */
2
#include <stdio.h>
4 #include "liste.h"
6
int main()
{
8     Cvor *glava = NULL;
9     double broj;
10
11     /* Ucitavanje se vrshi do kraja ulaza. Elementi se dodaju na kraj
12        liste! */
13     while (scanf("%lf", &broj) > 0)
14         dodaj_na_kraj_liste(&glava, broj);
15
16     dopuni_listu(&glava);
17
18     ispisi_listu(glava);
19
20     oslobodi_listu(&glava);
21
22     return 0;

```

```
}

```

### Rešenje 5.9

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Funkcija proverava da li je magican kvadrat koji joj se
   prosledjuje kao argument. Ukoliko jeste magican funkcija vraca 1,
   inace 0. */
6  int magicni_kvadrat(int **M, int n)
8  {
9      int i, j;
10     int zbir = 0, zbir_pom;
11
12     for (j = 0; j < n; j++)
13         zbir += M[0][j];
14
15     for (i = 1; i < n; i++) {
16         zbir_pom = 0;
17         for (j = 0; j < n; j++)
18             zbir_pom += M[i][j];
19         if (zbir_pom != zbir)
20             return 0;
21     }
22
23     for (j = 0; j < n; j++) {
24         zbir_pom = 0;
25         for (i = 0; i < n; i++)
26             zbir_pom += M[i][j];
27         if (zbir_pom != zbir)
28             return 0;
29     }
30     return 1;
31 }
32
33 int main()
34 {
35     int n, i, j;
36     int **matrica = NULL;
37     int zbir = 0;
38
39     scanf("%d", &n);
40
41     if (n <= 0) {
42         printf("-1\n");
43         exit(EXIT_FAILURE);
44     }
45
46     matrica = (int **) malloc(n * sizeof(int *));
47     if (matrica == NULL) {

```

```
48     printf("-1\n");
49     exit(EXIT_FAILURE);
50 }
51
52 for (i = 0; i < n; i++) {
53     matrica[i] = (int *) malloc(n * sizeof(int));
54
55     if (matrica[i] == NULL) {
56         fprintf(stderr, "-1\n");
57         for (j = 0; j < i; j++)
58             free(matrica[j]);
59
60         free(matrica);
61         exit(EXIT_FAILURE);
62     }
63 }
64
65 for (i = 0; i < n; i++)
66     for (j = 0; j < n; j++)
67         scanf("%d", &matrica[i][j]);
68
69 if (magicni_kvadrat(matrica, n)) {
70     for (i = 0; i < n; i++)
71         zbir += matrica[0][i];
72     printf("%d\n", zbir);
73 } else
74     printf("-\n");
75
76 for (j = 0; j < n; j++)
77     free(matrica[j]);
78
79 free(matrica);
80
81 return 0;
82 }
```