

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

*Andželka Zečević*, asistent na Matematičkom fakultetu u Beogradu

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu

Studentski trg 16, 11000 Beograd

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

Recenzenti:

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

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

Obrada teksta, crteži i korice: *autori*

Štampa:

Tiraž:

СИР Каталогизација у публикацији

Народна библиотека Србије, Београд

©2015. Milena Vujošević Janičić, Jelena Graovac, Ana Spasić, Mirko Spasić, Andželka 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 alociranoj 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 rezitorijum 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 savladavanje koncepata koji se obrađuju u okviru kursa.

...

*Autori*

# Sadržaj

<b>1 Uvodni zadaci</b>	<b>2</b>
1.1 Podela koda po datotekama . . . . .	2
1.2 Algoritmi za rad sa bitovima . . . . .	6
1.3 Rekurzija . . . . .	12
1.4 Rešenja . . . . .	21
<b>2 Pokazivači</b>	<b>69</b>
2.1 Pokazivačka aritmetika . . . . .	69
2.2 Višedimenzionalni nizovi . . . . .	73
2.3 Dinamička alokacija memorije . . . . .	77
2.4 Pokazivači na funkcije . . . . .	84
2.5 Rešenja . . . . .	86
<b>3 Algoritmi pretrage i sortiranja</b>	<b>127</b>
3.1 Algoritmi pretrage . . . . .	127
3.2 Algoritmi sortiranja . . . . .	132
3.3 Bibliotečke funkcije pretrage i sortiranja . . . . .	143
3.4 Rešenja . . . . .	148
<b>4 Dinamičke strukture podataka</b>	<b>220</b>
4.1 Liste . . . . .	220
4.2 Stabla . . . . .	231
4.3 Rešenja . . . . .	240
<b>5 Ispitni rokovi</b>	<b>335</b>
5.1 Programiranje 2, praktični deo ispita, jun 2015. . . . .	335
5.2 Programiranje 2, praktični deo ispita, jul 2015. . . . .	337
5.3 Programiranje 2, praktični deo ispita, septembar 2015. . . . .	339
5.4 Rešenja . . . . .	341

# 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 imaginarni_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`.

## 1.1 Podela koda po datotekama

- (i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja oduzima dva kompleksna broja  $z_1$  i  $z_2$ .
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja množi dva kompleksna broja  $z_1$  i  $z_2$ .
- (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  $z_1$  koji se unosi sa standardnog ulaza ispisuje njegov realni deo, imaginarni deo i moduo. Zatim, za naredni kompleksan broj  $z_2$  koji se unosi sa standardnog ulaza ispisuje njegov konjugovano-kompleksan broj i argument. Na kraju program ispisuje zbir, razliku i proizvod brojeva  $z_1$  i  $z_2$ .

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite realan i imaginarni deo kompleksnog broja: 1 -3  
(1.00 - 3.00 i)  
Unesite realan i imaginarni 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  
imaginarni_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 imaginarni 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. Ispisivanje polinoma počinje od najvišeg stepena ka najnižem. Ispisuju se samo oni koeficijenti koji su različiti od nule.
- (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 dатој тачки `x` koristeći Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja sabira dva polinoma `p` и `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja množi dva polinoma `p` и `q`.
- (g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja računa izvod polinoma `p`.
- (h) Napisati funkciju `Polinom n_izvod(const Polinom * p, int n)` koja računa `n`-ti izvod polinoma `p`.

Napisati program koji testira prethodno napisane funkcije. Najpre se polinomi `p` и `q` unose sa standardnog ulaza i ispisuju na standardni izlaz u odgovarajućem obliku. Zatim se računa i ispisuje 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 тачки коју unosi korisnik. Na kraju se sa standardnog ulaza unosi број `n`, i ispisuje `n`-ti izvod polinoma `r`.

### Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite polinom p (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):  
|| 3 1.2 3.5 2.1 4.2  
|| Unesite polinom q (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):  
|| 2 2.1 0 -3.9  
|| Zbir polinoma je: 1.20x^3+5.60x^2+2.10x+0.30  
|| Prozvod 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) Definisati 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 reciprocna_vrednost(const Razlomak * r)` koja izračunava recipročnu vrednost razlomka `r`.
- (h) Napisati funkciju `Razlomak skrati(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`.

## 1.2 Algoritmi za rad sa bitovima

- (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 biblioteku stampanje\_bitova za rad sa bitovima koja sadrži funkciju stampaj\_bitove koja štampa bitove u binarnom zapisu neoznačenog celog broja x. Napisati program koji testira funkciju stampaj\_bitove za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

### Test 1

```
ULAZ:  
0x7F  
IZLAZ:  
000000000000000000000000000000001111111
```

### Test 2

```
ULAZ:  
0x80  
IZLAZ:  
0000000000000000000000000000000010000000
```

### Test 3

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

### Test 4

```
ULAZ:  
0xABCD123  
IZLAZ:  
1010101110011011110000100100011
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkcije prebroj\_bitove\_1 i prebroj\_bitove\_2 koje broje bitove sa vrednošću 1 u binarnom zapisu celog broja x. Prebrojavanje bitova ostvariti na dva načina:

## 1.2 Algoritmi za rad sa bitovima

- (a) formiranjem odgovarajuće maske i njenim pomeranjem
- (b) formiranjem odgovarajuće maske i pomeranjem promenljive  $x$ .

Napisati program koji za broj koji unosi u heksadekadnom formatu sa standardnog ulaza računa broj bitova sa vrednošću 1 korišćenjem funkcije `prebroj_bitove_1` ili funkcije `prebroj_bitove_2`. Od korisnika sa standardnog ulaza tražiti da izabere koju od ove funkcije treba koristiti u zavisnosti da li unese 1 ili 2. Ukoliko korisnik ne unese odgovarajuće vrednosti za redni broj funkcije prekinuti izvršavanje programa i ispisati odgovarajuću poruku na standardni izlaz za greške.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: 0x7F  
Unesite redni broj funkcije: 1  
Broj jedinica u zapisu je  
funkcija prebroj_bitove_1: 7
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: 0xABCD123  
Unesite redni broj funkcije: 2  
Broj jedinica u zapisu je  
funkcija prebroj_bitove_2: 17
```

*Primer 3*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: 0x00FF00FF  
Unesite redni broj funkcije: 2  
Broj jedinica u zapisu je  
funkcija prebroj_bitove_2: 16
```

*Primer 4*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: 0x00FF00FF  
Unesite redni broj funkcije: 3  
IZLAZ ZA GREŠKU:  
Neodgovarajuci redni broj funkcije!
```

[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 heksadekadnom formatu sa standardnog ulaza.

*Test 1*

```
ULAZ:  
0x7F  
IZLAZ:  
Najveci:  
11111100000000000000000000000000  
Najmanji:  
0000000000000000000000000000000111111
```

*Test 2*

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

Test 3	Test 4
ULAZ:	ULAZ:
0x00FF00FF	0xFFFFFFFF
IZLAZ:	IZLAZ:
Najveci:	Najveci:
11111111111111110000000000000000	11111111111111111111111111111111
Najmanji:	Najmanji:
00000000000000001111111111111111	11111111111111111111111111111111

[Rešenje 1.7]

**Zadatak 1.8** Napisati funkcije za rad sa bitovima. NAPOMENA: *Pozicije se broje počev od pozicije bita najmanje težine, pri čemu je bit najmanje težine na poziciji nula.*

- (a) Napisati funkciju **postavi\_0** koja određuje broj koji se dobija kada se  $n$  bitova datog broja  $x$ , počevši od pozicije  $p$ , postave na 0.
- (b) Napisati funkciju **postavi\_1** koja određuje broj koji se dobija kada se  $n$  bitova datog broja  $x$ , počevši od pozicije  $p$ , postave na 1.
- (c) Napisati funkciju **vrati\_bitove** koja određuje broj u kome se  $n$  bitova najmanje težine poklapa sa  $n$  bitova broja  $x$  počevši od pozicije  $p$ .
- (d) Napisati funkciju **postavi\_1\_n\_bitova** koja vraća broj koji se dobija upisivanjem poslednjih  $n$  bitova najmanje težine broja  $y$  u broj  $x$ , počevši od pozicije  $p$ .
- (e) Napisati funkciju **invertuj** koja vraća broj koji se dobija invertovanjem  $n$  bitova broja  $x$  počevši od pozicije  $p$ .

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve  $x$ ,  $n$ ,  $p$ ,  $y$  koji se unose sa standardnog ulaza. Program treba nakon učitavanja odgovarajućih vrednosti ispiše najpre binarne reprezenatacije brojeva  $x$  i  $y$ , a potom i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija.

*Primer 1*

```

|| INTERAKCIJA SA PROGRAMOM:
Unesite neoznacen ceo broj x: 235
Unesite neoznacen ceo broj n: 9
Unesite neoznacen ceo broj p: 24
Unesite neoznacen ceo broj y: 127
x = 235 = 0000000000000000000000000000000011101011
postavi_0( 235, 9, 24) = 0000000000000000000000000000000011101011

x = 235 = 0000000000000000000000000000000011101011
postavi_1( 235, 9, 24) = 0000000111111100000000011101011

x = 235 = 0000000000000000000000000000000011101011
vrati_bitove( 235, 9, 24) = 000000000000000000000000000000000000000000000000000000000000000

x = 235 = 0000000000000000000000000000000011101011
y = 127 = 000000000000000000000000000000001111111
postavi_1_n_bitove( 235, 9, 24, 127) = 0000000001111110000000011101011

x = 235 = 0000000000000000000000000000000011101011
invertuj( 235, 9, 24) = 00000000111111100000000011101011

```

*Primer 2*

```

|| INTERAKCIJA SA PROGRAMOM:
Unesite neoznacen ceo broj x: 2882398951
Unesite neoznacen ceo broj n: 5
Unesite neoznacen ceo broj p: 10
Unesite neoznacen ceo broj y: 35156526
x = 2882398951 = 10101011110011011110101011100111
postavi_0(2882398951, 5, 10) = 10101011110011011110100000100111

x = 2882398951 = 10101011110011011110101011100111
postavi_1(2882398951, 5, 10) = 1010101111001101111011111100111

x = 2882398951 = 10101011110011011110101011100111
vrati_bitove(2882398951, 5, 10) = 000000000000000000000000000000001011

x = 2882398951 = 10101011110011011110101011100111
y = 35156526 = 00000010000110000111001000101110
postavi_1_n_bitove(2882398951, 5, 10, 35156526) = 10101011110011011110101110100111

x = 2882398951 = 10101011110011011110101011100111
invertuj(2882398951, 5, 10) = 10101011110011011110110100100111

```

[Rešenje 1.8]

**Zadatak 1.9** Pod rotiranjem uлево подразумева се помјеравање свих битова за једну позицију улево, с тим што се бит са позиције највеће тежине помјера на позицију најмање тежине. Аналогно, ротирање удесно подразумева помјеравање свих битова за једну позицију удесно, с тим што се бит са позиције најмање тежине помјера на позицију највише тежине.

- (a) Написати функцију `rotiraj_ulevo` која одређује број који се добија ротира-

## 1.2 Algoritmi za rad sa bitovima

njem k puta uлево datog celog broja x.

- (b) Napisati funkciju `rotiraj_udesno` koja određuje broj koji se dobija rotiranjem k puta udesno datog celog neoznačenog broja x.
- (c) Napisati funkciju `rotiraj_udesno_oznaceni` 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 heksadekasmom formatu sa standardnog ulaza.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite neoznacen ceo broj x: B10011A7  
Unesite neoznacen ceo broj k: 5  
x = 10110001000000000001000110100111  
rotiraj_ulevo(2969571751, 5)      = 0010000000000100011010011110110  
rotiraj_udesno(2969571751, 5)     = 00111101100010000000000010001101  
rotiraj_udesno_oznaceni(2969571751, 5) = 00111101100010000000000010001101
```

[Rešenje 1.9]

**Zadatak 1.10** Napisati funkciju `ogledalo` 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 heksadekasmom formatu, tako što najpre ispisuje binarnu reprezentaciju unetog broja, a potom i binarnu reprezentaciju broja dobijenog nakon poziva funkcije `ogledalo` za uneti broj.

### Test 1

```
ULAZ:  
255  
IZLAZ:  
000000000000000000000000000000001001010101  
101010100100000000000000000000000000000000
```

### Test 2

```
ULAZ:  
-15  
IZLAZ:  
1111111111111111111111111111111101011  
110101111111111111111111111111111111111111
```

[Rešenje 1.10]

**Zadatak 1.11** Napisati funkciju `int broj_01(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.*

Test 1

ULAZ:	11
IZLAZ:	1

Test 2

ULAZ:	1024
IZLAZ:	0

Test 3

ULAZ:	2147377146
IZLAZ:	22

[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. Smatratи da je krajnji desni bit binarne reprezentacije 0-ti bit. Pri rešavanju nije dozvoljeno koristiti ni pomoćni niz ni aritmetičke operatore +, -, /, \*, %.

Primer 1

POZIV:	./a.out 1 2
--------	-------------

INTERAKCIJA SA PROGRAMOM:	
ULAZ:	11
IZLAZ:	13

Primer 2

POZIV:	./a.out 1 2
--------	-------------

INTERAKCIJA SA PROGRAMOM:	
ULAZ:	1024
IZLAZ:	1024

Primer 2

POZIV:	./a.out 12 12
--------	---------------

INTERAKCIJA SA PROGRAMOM:	
ULAZ:	12345
IZLAZ:	12345

**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 zamenuju 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>
ULAZ: 11 IZLAZ: 0000000B	ULAZ: 1024 IZLAZ: 00000400	ULAZ: 12345 IZLAZ: 00003039

[Rešenje 1.14]

**Zadatak 1.15** Napisati funkciju koja za dva neoznačena broja  $x$  i  $y$  invertuje u podatku  $x$  one bitove koji se poklapaju sa odgovarajućim bitovima u broju  $y$ . Ostali bitovi ostaju nepromjenjeni. 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 IZLAZ: 4294967285	ULAZ: 3251 0 IZLAZ: 4294967295	ULAZ: 12541 1024 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 IZLAZ: 0	ULAZ: 3245 IZLAZ: 2	ULAZ: 100328 IZLAZ: 1

## 1.3 Rekurzija

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

- (a) tako da rešenje bude linearne složenosti,
- (b) tako da rešenje bude logaritamske složenosti.

Napisati program koji testira napisane funkciju tako što se sa standardnog ulaza najpre unosi redni broj funkcije koja se primenjuje, a zatim i ceo broj  $x$  i prirodan

broj  $k$ , a potom se na standarni izlaz ispisuje rezultat primene odgovarajuće funkcije na unete brojeve. Ukoliko se za redni broj funkcije unese broj različit od 1 i 2 ispisati odgovarajuću poruku i prekinuti izvršavanje programa.

*Primer 1*

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

*Primer 2*

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

[Rešenje 1.17]

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

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

Napisati program koji testira napisane funkcije 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. **NAPOMENA:** *Gornje ograničenje 12 je postavljeno zbog ograničenja za tip podataka int i činjenice da niz faktorijela brzo raste.*

*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** Napisati funkciju koja računa  $n$ -ti element u nizu Fibonačijevih brojeva. Elementi niza Fibonačijevih brojeva  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n - 1) + F(n - 2).$$

Napisati program koji testira napisane funkciju tako što se sa standardnog ulaza unosi prirodan broj  $n$  i na standardni izlaz se ispisuje rezultat primene napisane funkcije na prirodan broj  $n$ .

*Primer 1*

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

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite koji clan niza se racuna: 8  
|| F(8) = 21
```

**Zadatak 1.21** Elementi niza  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

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

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

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

Napisati program koji testira napisane funkciju tako što se sa standardnog ulaza najpre unosi redni broj funkcije koja se primenjuje, a zatim i vrednosti koeficijenata  $a$  i  $b$  i prirodan broj  $n$ . Na standardni izlaz se ispisuje rezultat primene odabrane funkcije na unete vrednosti koeficijenata  $a$  i  $b$  i prirodan broj  $n$ . NAPOMENA: *Niz  $F$  definisan na ovaj način predstavlja uopštenje Fibonačijevih brojeva.*

*Primer 1*

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

*Primer 2*

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

[Rešenje 1.21]

**Zadatak 1.22** 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 1.22]

**Zadatak 1.23** Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije (1 ili 2), zatim dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojnog niza, a potom i elemente niza. Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim nizom, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa.

*Primer 1*

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

*Primer 2*

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

[Rešenje 1.23]

**Zadatak 1.24** 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 1.24]

**Zadatak 1.25** Napisati rekurzivnu funkciju 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 1.25]

**Zadatak 1.26** Napisati rekurzivnu funkciju 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 1.26]

**Zadatak 1.27** Napisati rekurzivnu funkciju 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 1.27]

**Zadatak 1.28** 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 1.28]

**Zadatak 1.29** 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:
	000000000000000000000000000000000000001010

*Test 2*

	ULAZ:
	0
	IZLAZ:
	00

**Zadatak 1.30** 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 1.30]

**Zadatak 1.31** 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 1.31]

**Zadatak 1.32** 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. Prepostaviti 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 1.32]

\* **Zadatak 1.33** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za proizvoljan prirodan broj  $n$  ( $n \leq 15$ ) unet sa standardnog ulaza.

Test 1

ULAZ:
2
IZLAZ:
1 2
2 1

Test 2

ULAZ:
3
IZLAZ:
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Test 3

ULAZ:
-5
Duzina
permutacije
mora biti
broj iz
intervala
[0, 15]!

[Rešenje 1.33]

\* **Zadatak 1.34** Paskalov trougao sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbiru dve vrednosti koje su u susedna dva 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 elemenata 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.

$$\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 1 & & 1 & & \\
 & 1 & & 2 & & 1 & \\
 1 & & 3 & & 3 & & 1 \\
 1 & 4 & 6 & 4 & 1 & & \\
 1 & 5 & 10 & 10 & 5 & 1 & 
 \end{array}$$

- (a) Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta  $\binom{n}{k}$  koristeći osobine Paskalovog trougla.
- (b) Napisati rekurzivnu funkciju koja izračunava  $d_n$  kao sumu elemenata  $n$ -te hipotenuze Paskalovog trougla.

Napisati program koji za unetu veličinu Paskalovog trougla i redni broj hipotenuze najpre iscrtava Paskalov trougao, a zatim štampa sumu elemenata hipotenuze.

*Test 1*

ULAZ:	5 3
IZLAZ:	<pre>       1      1 1     1 2 1    1 3 3 1   1 4 6 4 1  1 5 10 10 5 1  8 </pre>

*Test 2*

ULAZ:	6 5
IZLAZ:	<pre>       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 </pre>

[Rešenje 1.34]

**Zadatak 1.35** 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:	<pre> a a a b b a b b </pre>

*Test 2*

ULAZ:	3
IZLAZ:	<pre> a a a a a b a b a a b b b a a b a b b b a b b b </pre>

**Zadatak 1.36 Hanojske kule:** Data su tri vertikalna štapa, na jednom se nalazi  $n$  diskova poluprečnika  $1, 2, 3, \dots$  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 1.37 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 <stdlib.h>
3 #include <math.h>
4
5 /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
6   imaginaran deo kompleksnog broja */
7 typedef struct {
8   float real;
9   float imag;
10 } KompleksanBroj;
11
12 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
13   kompleksnog broja i smesta ih u strukturu cija je adresa argument
14   funkcije */
15 void ucitaj_kompleksan_broj(KompleksanBroj * z)
16 {
17   /* Ucitavanje vrednosti sa standardnog ulaza */
18   printf("Unesite realan i imaginaran deo kompleksnog broja: ");
19   scanf("%f", &z->real);
20   scanf("%f", &z->imag);
21 }
22
23 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
24   obliku (x + i y). Ovoj funkciji se argument prenosi po vrednosti
25   jer se u samoj funkciji ne menja njegova vrednost */
26 void ispisi_kompleksan_broj(KompleksanBroj z)
27 {
28   /* Zapocinje se sa ispisom */
29 }
```

```

    printf("(");

30   /* Razlikuju se dva slučaja: 1) realni deo kompleksnog broja
32     razlicit od nule: tada se realni deo ispisuje na standardni
34     izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
36     imaginarni deo pozitivan ili negativan, a potom i apsolutna
38     vrednost imaginarnog dela kompleksnog broja 2) realni deo
      kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
      s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
      decimalnih mesta */

40   if (z.real != 0) {
        printf("%.2f", z.real);

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

54   /* Zavrsava se sa ispisom */
    printf(")");

56 }

58 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
60 float realan_deo(KompleksanBroj z)
{
62   return z.real;
}

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

70 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
72 float moduo(KompleksanBroj z)
{
74   return sqrt(z.real * z.real + z.imag * z.imag);
}

76 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
78   odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z)
{

```

```

82  /* Konjugovano kompleksan broj z se dobija tako sto se promeni znak
83   imaginarnom delu kompleksnog broja */
84
84  KompleksanBroj z1 = z;
85
86  z1.imag *= -1;
87
88  return z1;
89 }
90
91 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
92   argumenata funkcije */
93 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
94 {
95  /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
96   broj ciji je realan deo zbir realnih delova kompleksnih brojeva
97   z1 i z2, a imaginarni deo zbir imaginarnih delova kompleksnih
98   brojeva z1 i z2 */
99
100 KompleksanBroj z = z1;
101
102 z.real += z2.real;
103 z.imag += z2.imag;
104
105 return z;
106 }
107
108 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
109   argumenata funkcije */
110 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
111 {
112  /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
113   broj ciji je realan deo razlika realnih delova kompleksnih
114   brojeva z1 i z2, a imaginarni deo razlika imaginarnih delova
115   kompleksnih brojeva z1 i z2 */
116
116 KompleksanBroj z = z1;
117
118 z.real -= z2.real;
119 z.imag -= z2.imag;
120
121 return z;
122 }
123
124 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
125   argumenata funkcije */
126 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
127 {
128  /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
129   broj ciji se realan i imaginarni deo racunaju po formuli za
130   mnozenje kompleksnih brojeva z1 i z2 */
131
132 }

```

```
134     KompleksanBroj z;
135
136     z.real = z1.real * z2.real - z1.imag * z2.imag;
137     z.imag = z1.real * z2.imag + z1.imag * z2.real;
138
139     return z;
140 }
141
142 /* Funkcija vraca argument zadatog kompleksnog broja */
143 float argument(KompleksanBroj z)
144 {
145     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
146      iz biblioteke math.h */
147
148     return atan2(z.imag, z.real);
149 }
150
151 int main()
152 {
153     char c;
154
155     /* Deklaracija 3 promenljive tipa KompleksanBroj */
156     KompleksanBroj z1, z2, z;
157
158     /* Ucitavanje prvog kompleksnog broja, a potom i njegovo
159      ispisivanje na standardni izlaz */
160     ucitaj_kompleksan_broj(&z1);
161     ispisi_kompleksan_broj(z1);
162     printf("\n");
163
164     /* Ucitavanje drugog kompleksnog broja, a potom njegovo ispisivanje
165      na standardni izlaz */
166     ucitaj_kompleksan_broj(&z2);
167     ispisi_kompleksan_broj(z2);
168     printf("\n");
169
170     /* Ucitavanje i provera znaka na osnovu koga korisnik bira
171      aritmeticku operaciju koja ce se izvrsiti nad kompleksnim
172      brojevima */
173     getchar();
174     printf("Unesite znak (+,-,*): ");
175     scanf("%c", &c);
176     if (c != '+' && c != '-' && c != '*') {
177         printf("Greska: nedozvoljena vrednost operatora!\n");
178         exit(EXIT_FAILURE);
179     }
180
181     /* Analizira se uneti operator */
182     if (c == '+') {
183         /* Racuna se zbir */
184         z = saberi(z1, z2);
185     } else if (c == '-') {
```

```

186     /* Racuna se razlika */
187     z = oduzmi(z1, z2);
188 } else {
189     /* Racuna se proizvod */
190     z = mnozi(z1, z2);
191 }
192
193 /* Ispisuje se rezultat */
194 printf("\n");
195 ispisi_kompleksan_broj(z1);
196 printf(" %c ", c);
197 ispisi_kompleksan_broj(z2);
198 printf(" = ");
199 ispisi_kompleksan_broj(z);
200 printf("\n");
201
202 /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
203   broja */
204 printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo: %.f\n",
205       realan_deo(z), imaginaran_deo(z), moduo(z));
206
207 /* Izracunava se i ispisuje konjugovano kompleksan broj drugog
208   kompleksnog broja */
209 printf("Njegov konjugovano kompleksan broj: ");
210 ispisi_kompleksan_broj(konjugovan(z));
211 printf("\n");
212
213 /* Testira se funkcija koja racuna argument kompleksnog broja */
214 printf("Argument kompleksnog broja: %f\n", argument(z));
215
216 exit(EXIT_SUCCESS);
}

```

## Rešenje 1.2

Datoteka 1.1: *complex.h*

```

1  /* Zaglavljje complex.h sadrzi definiciju tipa KompleksanBroj i
2   deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavljje
3   nikada ne treba da sadrzi definicije funkcija. Da bi neki program
4   mogao da koristi ove brojeve i funkcije iz ove biblioteke,
5   neophodno je da ukljuci ovo zaglavljje. */
6
7  /* Ovim preprocesorskim direktivama se zaključava zaglavljje i
8   onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
9   posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
10  narednoj preprocesorskoj define direktivi. */
11 #ifndef _COMPLEX_H
12 #define _COMPLEX_H
13

```

```

15  /* Zaglavljа standardne biblioteke koje sadrže deklaracije funkcija
   koje se koriste u definicijama funkcija navedenim u complex.c */
16  #include <stdio.h>
17  #include <math.h>

18  /* Struktura KompleksanBroj */
19  typedef struct {
20      float real;
21      float imag;
22  } KompleksanBroj;

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

24
25  /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
   kompleksnog broja i smesta ih u strukturu cija je adresa argument
   funkcije */
26
27  void ucitaj_kompleksan_broj(KompleksanBroj * z);

28
29  /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
   obliku (x + i y) */
30
31  void ispisi_kompleksan_broj(KompleksanBroj z);

32
33  /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
34  float realan_deo(KompleksanBroj z);

35
36  /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
37  float imaginaran_deo(KompleksanBroj z);

38
39  /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
40  float modulo(KompleksanBroj z);

41
42  /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
   odgovara kompleksnom broju argumentu */
43
44  KompleksanBroj konjugovan(KompleksanBroj z);

45
46  /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
   argumenata funkcije */
47
48  KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);

49
50  /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
   argumenata funkcije */
51
52  KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);

53
54  /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
   argumenata funkcije */
55
56  KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);

57
58  /* Funkcija vraca argument zadatog kompleksnog broja */
59  float argument(KompleksanBroj z);

60
61  /* Kraj zakljucanog dela */
62
63
64
65

```

```
#endif
```

Datoteka 1.2: *complex.c*

```

2  /* Uključuje se zaglavlje za rad sa kompleksnim brojevima, jer je
   neophodno da bude poznata definicija tipa KompleksanBroj. Takođe,
   time su uključena zaglavla standardne biblioteke koja su navedena
4   u complex.h */
5   #include "complex.h"
6

8 void ucitaj_kompleksan_broj(KompleksanBroj * z)
9 {
10    /* Ucitavanje vrednosti sa standardnog ulaza */
11    printf("Unesite realan i imaginarni deo kompleksnog broja: ");
12    scanf("%f", &z->real);
13    scanf("%f", &z->imag);
14 }

16 void ispisi_kompleksan_broj(KompleksanBroj z)
17 {
18    /* Zapocinje se sa ispisom */
19    printf("(");
20
21    /* Razlikuju se dva slučaja: 1) realni deo kompleksnog broja
22       razlicit od nule: tada se realni deo ispisuje na standardni
23       izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
24       imaginarni deo pozitivan ili negativan, a potom i apsolutna
25       vrednost imaginarnog dela kompleksnog broja 2) realni deo
26       kompleksnog broja je nula: tada se samo ispisuje imaginarni deo,
27       s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
28       decimalnih mesta */

29    if (z.real != 0) {
30        printf("%.2f", z.real);
31
32        if (z.imag > 0)
33            printf(" + %.2f i", z.imag);
34        else if (z.imag < 0)
35            printf(" - %.2f i", -z.imag);
36    } else {
37        if (z.imag == 0)
38            printf("0");
39        else
40            printf("%.2f i", z.imag);
41    }
42
43    /* Zavrsava se sa ispisom */
44    printf(")");
45}
46 }
```

```

48 float realan_deo(KompleksanBroj z)
49 {
50     /* Vraca se vrednost realnog dela kompleksnog broja */
51     return z.real;
52 }

54 float imaginaran_deo(KompleksanBroj z)
55 {
56     /* Vraca se vrednost imaginarnog dela kompleksnog broja */
57     return z.imag;
58 }

60 float moduo(KompleksanBroj z)
61 {
62     /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
63     return sqrt(z.real * z.real + z.imag * z.imag);
64 }

66 KompleksanBroj konjugovan(KompleksanBroj z)
67 {
68     /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
69      se promeni znak imaginarnom delu kompleksnog broja */
70     KompleksanBroj z1 = z;
71     z1.imag *= -1;
72     return z1;
73 }

74 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
75 {
76     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
77      broj ciji je realan deo zbir realnih delova kompleksnih brojeva
78      z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
79      brojeva z1 i z2 */
80     KompleksanBroj z = z1;
81
82     z.real += z2.real;
83     z.imag += z2.imag;
84
85     return z;
86 }
87

88 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
89 {
90     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
91      broj ciji je realan deo razlika realnih delova kompleksnih
92      brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
93      kompleksnih brojeva z1 i z2 */
94     KompleksanBroj z = z1;
95     z.real -= z2.real;
96     z.imag -= z2.imag;
97     return z;
98 }

```

```

100 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
101 {
102     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
103      broj ciji se realan i imaginarni deo racunaju po formuli za
104      mnozenje kompleksnih brojeva z1 i z2 */
105
106     KompleksanBroj z;
107
108     z.real = z1.real * z2.real - z1.imag * z2.imag;
109     z.imag = z1.real * z2.imag + z1.imag * z2.real;
110
111     return z;
112 }
113
114 float argument(KompleksanBroj z)
115 {
116     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
117      iz biblioteke math.h */
118     return atan2(z.imag, z.real);
119 }
```

Datoteka 1.3: *main.c*

```

1 ****
2 Ovaj program koristi korektno definisaniu biblioteku kompleksnih
3 brojeva. U zaglavlju complex.h nalazi se definicija kompleksnog broja
4 i popis deklaracija podrzanih funkcija, a u complex.c se nalaze
5 njihove definicije.
6
7 Kompilacija programa se najjednostavnije postize naredbom
8 gcc -Wall -lm -o complex complex.c main.c
9
10 Kompilacija se moze uraditi i na sledeci nacin:
11 gcc -Wall -c -o complex.o complex.c
12 gcc -Wall -c -o main.o main.c
13 gcc -lm -o complex complex.o main.o
14
15 Napomena: Prethodne komande se koriste kada se sva tri navedena
16 dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
17 complex.c complex.h) nalazi u direktorijumu sa imenom header_dir
18 prevodjenje se vrsti dodavanjem opcije opcije -I header_dir
19 gcc -I header_dir -Wall -lm -o complex complex.c main.c
20 ****
21
22
23 #include <stdio.h>
24 /* Uključuje se zaglavljne neophodno za rad sa kompleksnim brojevima
25 */
26 #include "complex.h"
27
28 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
```

```

    polarni oblik */
29 int main()
{
31     KompleksanBroj z;
33     /* Ucitavamo kompleksan broj */
34     ucitaj_kompleksan_broj(&z);
35
36     /* Ispisujemo njegov polarni oblik */
37     printf("Polarni oblik kompleksnog broja je %.2f * e^i * %.2f\n",
38           moduo(z), argument(z));
39
40     return 0;
41 }
```

**Rešenje 1.3**Datoteka 1.4: *polinom.h*

```

#ifndef _POLINOM_H
#define _POLINOM_H

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

/* Maksimalni stepen polinoma */
#define MAKS_STEPEN 20

/* Polinomi se predstavljaju strukturom koja cuva koeficijente
   (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
typedef struct {
    double koef[MAKS_STEPEN + 1];
    int stepen;
} Polinom;

/* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
   obliku. Polinom se prenosi po adresi da bi se ustedela memorija:
   ne kopira se cela struktura, vec se samo prenosi adresa na kojoj
   se nalazi polinom koji ispisujemo */
void ispisi(const Polinom * p);

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

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

/* Funkcija koja sabira dva polinoma */
```

```

32 Polinom saberi(const Polinom * p, const Polinom * q);
34 /* Funkcija koja mnozi dva polinoma p i q */
35 Polinom pomnozi(const Polinom * p, const Polinom * q);
36 /* Funkcija koja racuna izvod polinoma p */
37 Polinom izvod(const Polinom * p);
38 /* Funkcija koja racuna n-ti izvod polinoma p */
39 Polinom n_izvod(const Polinom * p, int n);
42 #endif

```

Datoteka 1.5: *polinom.c*

```

#include <stdio.h>
2 #include <stdlib.h>
# include "polinom.h"
4
void ispisi(const Polinom * p)
{
    int nulaPolinom = 1;
    int i;
    /* Ispisivanje polinoma pocinje od najviseg stepena ka najnizem da
     bi polinom bio ispisana na prirodan nacin. Ispisuju se samo oni
     koeficijenti koji su razliciti od nule. Ispred pozitivnih
     koeficijenata je potrebno ispisati znak + (osim u slucaju
     koeficijenta uz najvisi stepen). */
14   for (i = p->stepen; i >= 0; i--) {
16
        if (p->koef[i]) {
            /* Polinom nije nula polinom, cim je neki od koeficijenata
             razlicit od nule */
            nulaPolinom = 0;
20           if (p->koef[i] >= 0 && i != p->stepen)
                putchar('+');
22           if (i > 1)
                printf("%.2fx^%d", p->koef[i], i);
24           else if (i == 1)
                printf("%.2fx", p->koef[i]);
26           else
                printf("%.2f", p->koef[i]);
28       }
29   }
30   /* U slucaju nula polinoma indikator ce imati vrednost 1 i tada se
31   ispisuje nula. */
32   if(nulaPolinom)
33       printf("0");
34       putchar('\n');
35 }
36 Polinom ucitaj()

```

```

38 {
40     int i;
41     Polinom p;
42
43     /* Ucitava se stepena polinoma */
44     scanf("%d", &p.stepen);
45
46     /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
47      dozvoljenog opsega */
48     while (p.stepen > MAKS_STEPEN || p.stepen < 0) {
49         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
50         scanf("%d", &p.stepen);
51     }
52
53     /* Unose se koeficijenti polinoma */
54     for (i = p.stepen; i >= 0; i--)
55         scanf("%lf", &p.koef[i]);
56
57     /* Vraca se procitani polinom */
58     return p;
59 }
60
61 double izracunaj(const Polinom * p, double x)
62 {
63     /* Rezultat se na pocetku inicializuje na nulu, a potom se u
64      svakoj iteraciji mnozi sa x, a potom i uvecava za
65      vrednost odgovarajuceg koeficijenta */
66
67     /* Primer: Hornerov algoritam za polinom  $x^4+2x^3+3x^2+2x+1$ :
68       $x^4+2x^3+3x^2+2x+1 = (((x+2)*x + 3)*x + 2)*x + 1$  */
69
70     double rezultat = 0;
71     int i = p->stepen;
72     for (; i >= 0; i--)
73         rezultat = rezultat * x + p->koef[i];
74     return rezultat;
75 }
76
77 Polinom saberi(const Polinom * p, const Polinom * q)
78 {
79     Polinom rez;
80     int i;
81
82     /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
83      stepenom */
84     rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
85
86     /* Racunaju se svi koeficijenti rezultujuceg polinoma tako sto se
87      sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
88      sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
      od stepena nekog od polaznih polinoma podrazumeva se da je
      koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog

```

```

90     polinoma */
91     for (i = 0; i <= rez.stepen; i++)
92         rez.koef[i] =
93             (i > p->stepen ? 0 : p->koef[i]) +
94             (i > q->stepen ? 0 : q->koef[i]);
95
96     /* Vraca se dobijeni polinom */
97     return rez;
98 }
99
100 Polinom pomnozi(const Polinom * p, const Polinom * q)
101 {
102     int i, j;
103     Polinom r;
104
105     /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
106      */
107     r.stepen = p->stepen + q->stepen;
108     if (r.stepen > MAKS_STEPEN) {
109         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
110         exit(EXIT_FAILURE);
111     }
112
113     /* Svi koeficijenti rezultujuceg polinoma se inicijalizuju na nulu
114      */
115     for (i = 0; i <= r.stepen; i++)
116         r.koef[i] = 0;
117
118     /* U svakoj iteraciji odgovarajuci koeficijent rezultata se uvecava
119      za proizvod odgovarajucih koeficijenata iz polaznih polinoma */
120     for (i = 0; i <= p->stepen; i++)
121         for (j = 0; j <= q->stepen; j++)
122             r.koef[i + j] += p->koef[i] * q->koef[j];
123
124     /* Vraca se dobijeni polinom */
125     return r;
126 }
127
128 Polinom izvod(const Polinom * p)
129 {
130     int i;
131     Polinom r;
132
133     /* Izvod polinoma ce imati stepen za jedan stepen manji od stepena
134      polaznog polinoma. Ukoliko je stepen polinoma p vec nula, onda
135      je rezultujuci polinom nula (izvod od konstante je nula). */
136     if (p->stepen > 0) {
137         r.stepen = p->stepen - 1;
138
139         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
140          polaznog polinoma */

```

```

140     for (i = 0; i <= r.stepen; i++)
141         r.koef[i] = (i + 1) * p->koef[i + 1];
142     } else
143         r.koef[0] = r.stepen = 0;
144
145     /* Vraca se dobijeni polinom */
146     return r;
147 }
148
Polinom n_izvod(const Polinom * p, int n)
149 {
150     int i;
151     Polinom r;
152
153     /* Provera da li je n nenegativna vrednost */
154     if (n < 0) {
155         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >= 0 \n");
156         exit(EXIT_FAILURE);
157     }
158
159     /* Nulti izvod je bas taj polinom */
160     if (n == 0)
161         return *p;
162
163     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove funkcija
164      za racunanje prvog izvoda polinoma */
165     r = izvod(p);
166     for (i = 1; i < n; i++)
167         r = izvod(&r);
168
169     /* Vraca se dobijeni polinom */
170     return r;
171 }
172 }
```

Datoteka 1.6: *main.c*

```

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

```

16    /* Ispis polinoma p */
17    ispisi(&p);

18    /* Unos polinoma q */
19    printf
20        ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
21         od najveceg stepena do nultog):\n");
22    q = ucitaj();

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

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

31    /* Ispisuje se vrednost polinoma u unetoj tacki */
32    printf("Unesite tacku u kojoj racunate vrednost polinoma r:\n");
33    scanf("%lf", &x);
34    printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&r, x));

35    /* Racuna se n-ti izvoda polinoma i ispisuje se dobijeni polinoma
36     */
37    printf("Unesite izvod polinoma koji zelite:\n");
38    scanf("%d", &n);
39    r = n_izvod(&r, n);
40    printf("%d. izvod polinoma r je: ", n);
41    ispisi(&r);

42    exit(EXIT_SUCCESS);
43}

```

### Rešenje 1.5

Datoteka 1.7: *stampa\_bitova.h*

```

1 #ifndef _STAMPANJE_BITOVA_H
2 #define _STAMPANJE_BITOVA_H

4 #include <stdio.h>

6 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
7  celog broja u memoriji. Bitove koji predstavljaju binarnu
8   reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
9    najveće tezine ka bitu najmanje tezine */
10 void stampaj_bitove(unsigned x);

```

```
12 #endif
```

Datoteka 1.8: *stampanje\_bitova.c*

```
#include <stdio.h>
2 #include "stampanje_bitova.h"

4 void stampaj_bitove(unsigned x)
{
6 /* Broj bitova celog broja */
7     unsigned velicina = sizeof(unsigned) * 8;

8 /* Maska koja se koristi za "ocitavanje" bitova */
9     unsigned maska;

12 /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
13     na mestu bita najvece tezine sadrzi jedinicu, a na svim ostalim
14     mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
15     se jedini bit jedinica pomera udesno, kako bi se odredio naredni
16     bit broja x koji je argument funkcije. Zatim se odgovarajuca
17     cifra, ('0' ili '1'), ispisuje na standardnom izlazu. Neophodno
18     je da promenljiva maska bude deklarisana kao neoznacen ceo broj
19     kako bi se pomeranjem u desno vrstilo logicko pomeranje
20     (popunjavanje nulama), a ne aritmeticko pomeranje (popunjavanje
21     znakom broja). */
22     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
23         putchar(x & maska ? '1' : '0');

24     putchar('\n');
25 }
```

Datoteka 1.9: *main.c*

```
#include <stdio.h>
2 #include "stampanje_bitova.h"

4 int main()
{
6     int broj;

8     /* Ucitava se broj sa ulaza */
9     scanf("%x", &broj);

10    /* I ispisuje se njegova binarna reprezentacija */
11    stampaj_bitove(broj);

13    return 0;
14 }
```

## Rešenje 1.6

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
5    kreiranjem odgovarajuce maske i njenim pomeranjem */
6 int prebroj_bitove_1(int x)
7 {
8     int br = 0;
9     unsigned broj_pomeranja = sizeof(unsigned) * 8 - 1;
10
11    /* Formiranje se maska cija binarna reprezentacija izgleda
12       100000...0000000, koja sluzi za ocitavanje bita najvece tezine.
13       U svakoj iteraciji maska se pomera u desno za 1 mesto, i
14       ocitavamo sledeci bit. Petlja se zavrsava kada vise nema
15       jedinica tj. kada maska postane nula. */
16    unsigned maska = 1 << broj_pomeranja;
17    for (; maska != 0; maska >>= 1)
18        x & maska ? br++ : 1;
19
20    return br;
21 }
22
23 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
24    formiranjem odgovarajuce maske i pomeranjem promenljive x */
25 int prebroj_bitove_2(int x)
26 {
27     int br = 0;
28     unsigned broj_pomeranja = sizeof(int) * 8 - 1;
29
30     /* Kako je argument funkcije ozначен ceo broj x naredba x>>=1 bi
31        vrsila aritmeticko pomeranje u desno, tj. popunjavanje bita
32        najvece tezine bitom znaka. U tom slucaju nikad ne bi bio
33        ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
34        petlji. Zbog toga se koristi pomeranje broja x uлево i maska
35        koja ocitava bit najvece tezine. */
36
37     unsigned maska = 1 << broj_pomeranja;
38     for (; x != 0; x <<= 1)
39         x & maska ? br++ : 1;
40
41     return br;
42 }
43
44 int main()
45 {
46     int x, i;
47
48     /* Ucitava se broj sa ulaza */
49     printf("Unesite broj:\n");
50     scanf("%x", &x);

```

```

51  /* Dozvoljava se korisniku da bira na koji nacin ce biti izracunat
53   broj jedinica u zapisu broja */
54   printf("Unesite redni broj funkcije:\n");
55   scanf("%d", &i);

56  /* Ispisuje se rezultat */
57  if (i == 1){
58    printf("Broj jedinica u zapisu je\n");
59    printf("funkcija prebroj_bitove_1: %d\n", prebroj_bitove_1(x));
60  }else if (i == 2){
61    printf("Broj jedinica u zapisu je\n");
62    printf("funkcija prebroj_bitove_2: %d\n", prebroj_bitove_2(x));
63  }else {
64    fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
65    exit(EXIT_FAILURE);
66  }

67  exit(EXIT_SUCCESS);
68 }

```

**Rešenje 1.7** NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```

1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija vraca najveci neoznacen broj sastavljen od istih bitova
5   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
6 unsigned najveci(unsigned x)
7 {
8     unsigned velicina = sizeof(unsigned) * 8;
9
10    /* Formira se maska 100000...0000000 */
11    unsigned maska = 1 << (velicina - 1);
12
13    /* Rezultat se inicijalizuje vrednoscu 0 */
14    unsigned rezultat = 0;
15
16    /* Promenljiva x se pomera u levo sve dok postoje jedinice u njenoj
17       binarnoj reprezentaciji (tj. sve dok je promenljiva x razlicita
18       od nule). */
19    for (; x != 0; x <= 1) {
20        /* Za svaku jedinicu koja se koriscenjem maske detektuje na
21           poziciji najvece tezine u binarnoj reprezentaciji promenjive
22           x, potiskuje se jedna nova jedinicu sa leva u rezultat */
23        if (x & maska) {
24            rezultat >>= 1;
25            rezultat |= maska;
26        }
27    }
28
29    return rezultat;
30 }

```

```

27    }
29    /* Vraca se dobijena vrednost */
31    return rezultat;
32 }
33 /* Funkcija vraca najmanji neoznaceni broj sastavljen od istih bitova
   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
34 unsigned najmanji(unsigned x)
35 {
36    /* Rezultat se inicijalizuje vrednoscu 0 */
37    unsigned rezultat = 0;
38
39    /* Promenljiva x se pomera u desno sve dok postoje jedinice u
   njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
   razlicita od nule). */
40    for (; x != 0; x >>= 1) {
41        /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
           detektuje na poziciji najmanje tezine u binarnoj
           reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
           sa desna u rezultat */
42        if (x & 1) {
43            rezultat <= 1;
44            rezultat |= 1;
45        }
46    }
47
48    /* Vraca se dobijena vrednost */
49    return rezultat;
50 }
51
52 int main()
53 {
54    int broj;
55
56    /* Ucitava se broj sa ulaza */
57    scanf("%x", &broj);
58
59    /* Ispisuju se, redom, najveci i najmanji broj formirani od bitova
       unetog broja */
60    printf("Najveci:\n");
61    stampaj_bitove(najveci(broj));
62
63    printf("Najmanji:\n");
64    stampaj_bitove(najmanji(broj));
65
66    return 0;
67 }

```

**Rešenje 1.8**NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova*

iz zadatka 1.5.

```

2 #include <stdio.h>
3 #include "stampanje_bitova.h"
4
5 /* Funckija postavlja na nulu n bitova pocev od pozicije p. */
6 unsigned postavi_0(unsigned x, unsigned n, unsigned p)
7 {
8     //*****
9     // Formira se maska cija binarna reprezentacija ima n bitova
10    // postavljenih na 0 pocev od pozicije p, dok su svi ostali
11    // postavljeni na 1. Na primer, za n=5 i p=10 formira se maska oblika
12    // 1111 1111 1111 1111 1111 1000 0011 1111
13    // To se postize na sledeci nacin:
14    // ~0          1111 1111 1111 1111 1111 1111 1111 1111
15    // (~0 << n)      1111 1111 1111 1111 1111 1111 1110 0000
16    // ~(~0 << n)      0000 0000 0000 0000 0000 0000 0001 1111
17    // (~(~0 << n) << (p-n+1)) 0000 0000 0000 0000 0000 0111 1100 0000
18    // ~(~(~0 << n) << (p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111
19    // *****
20    // unsigned maska = ~(~(~0 << n) << (p - n + 1));
21
22    return x & maska;
23 }
24
25 /* Funckija postavlja na jedinicu n bitova pocev od pozicije p. */
26 unsigned postavi_1(unsigned x, unsigned n, unsigned p)
27 {
28
29     //*****
30     // Formira se maska kod koje je samo n bitova pocev od pocev od
31     // pozicije p jednako 1, a ostali su 0.
32     // Na primer, za n=5 i p=10 formira se maska oblika
33     // 0000 0000 0000 0000 0000 0111 1100 0000
34     // *****
35     // unsigned maska = ~(~0 << n) << (p - n + 1);
36
37     return x | maska;
38 }
39
40 /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
41 predstavlja n bitova pocev od pozicije p u binarnoj
42 reprezentaciji broja x. */
43 unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
44 {
45
46     //*****
47     // Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
48     // Na primer, za n=5
49     // 0000 0000 0000 0000 0000 0001 1111
50     // *****

```

```

52     unsigned maska = ~(~0 << n);
54
55     /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
56      polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
57      zeljenih n i funkcija vraca tako dobijenu vrednost */
58     return maska & (x >> (p - n + 1));
59 }
60
61
62 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
63    postavljeni na vrednosti n bitova najmanje tezine binarne
64    reprezentacije broja y */
65 unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p,
66                             unsigned y)
67 {
68     /* Kreira se maska kod koje su poslednjih n bitova 1, a
69      ostali su 0. */
70     unsigned poslednjih_n_1 = ~(~0 << n);
71
72     /* Kao i kod funkcije postavi_0, i ovde se kreira maska koja ima n
73      bitova postavljenih na 0 pocevsi od pozicije p, dok su
74      ostali bitovi 1. */
75     unsigned srednjih_n_0 = ~(~(~0 << n) << (p - n + 1));
76
77     /* U promenljivu x_postavi_0 se smesta vrednost dobijena kada se u
78      binarnoj reprezentaciji vrednosti promenljive x postavi na 0 n
79      bitova na pozicijama pocev od p */
80     unsigned x_postavi_0 = x & srednjih_n_0;
81
82     /* U promenljivu y_pomeri_srednje se smesta vrednost dobijena od
83      binarne reprezentacije vrednosti promenljive y cijih je n bitova
84      najnize tezine pomera tako da stoje pocev od pozicije p. Ostali
85      bitovi su nule. Sa (y & poslednjih_n_1) postave na 0 svi bitovi
86      osim najnizih n */
87     unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);
88
89     return x_postavi_0 ^ y_pomeri_srednje;
90 }
91
92 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
93   njih n */
94 unsigned invertuj(unsigned x, unsigned n, unsigned p)
95 {
96     /* Formira se maska sa n jedinica pocev od pozicije p. */
97     unsigned maska = ~(~0 << n) << (p - n + 1);
98
99     /* Operator ekskluzivno ili invertuje sve bitove gde je
100    odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
101    return maska ^ x;
102 }

```

```

102 int main()
103 {
104     unsigned x, p, n, y;
105
106     /* Ucitavaju se vrednosti sa standardnog ulaza */
107     printf("Unesite neoznacen ceo broj x:\n");
108     scanf("%u", &x);
109     printf("Unesite neoznacen ceo broj n:\n");
110     scanf("%u", &n);
111     printf("Unesite neoznacen ceo broj p:\n");
112     scanf("%u", &p);
113     printf("Unesite neoznacen ceo broj y:\n");
114     scanf("%u", &y);
115
116     /* Ispisuju se binarne reprezentacije broja x i broja koji se
117      dobije kada se primeni funkcija postavi_0 za x, n i p*/
118     printf("x = %10u %36s = ", x, "");
119     stampaj_bitove(x);
120     printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
121     stampaj_bitove( postavi_0(x, n, p));
122     printf("\n");
123
124     /* Ispisuju se binarne reprezentacije broja x i broja koji se
125      dobije kada se primeni funkcija postavi_1 za x, n i p*/
126     printf("x = %10u %36s = ", x, "");
127     stampaj_bitove(x);
128     printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
129     stampaj_bitove( postavi_1(x, n, p));
130     printf("\n");
131
132     /* Ispisuju se binarne reprezentacije broja x i broja koji se
133      dobije kada se primeni funkcija vrati_bitove za x, n i p*/
134     printf("x = %10u %36s = ", x, "");
135     stampaj_bitove(x);
136     printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
137     stampaj_bitove( vrati_bitove(x, n, p));
138     printf("\n");
139
140     /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
141      dobije kada se primeni funkcija postavi_1_n_bitova za x, n i p*/
142     printf("x = %10u %36s = ", x, "");
143     stampaj_bitove(x);
144     printf("y = %10u %36s = ", y, "");
145     stampaj_bitove(y);
146     printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
147     stampaj_bitove( postavi_1_n_bitova(x, n, p, y));
148     printf("\n");
149
150     /* Ispisuju se binarne reprezentacije broja x i broja koji se
151      dobije kada se primeni funkcija invertuj za x, n i p*/
152     printf("x = %10u %36s = ", x, "");
153     stampaj_bitove(x);

```

```

154     printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
155     stampaj_bitove( invertuj(x, n, p));
156
157     return 0;
158 }
```

**Rešenje 1.9** NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```

1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija ceo broj x rotira u levo za n mesta. */
5 unsigned rotiraj_ulevo(int x, unsigned n)
6 {
7     unsigned bit_najvece_tezine;
8
9     /* Maska koja ima samo bit na poziciji najvece tezine postavljen na
10      1 je neophodna da bi pre pomeranja u levo za 1 bit na poziciji
11      najvece tezine bio sacuvan */
12     unsigned bit_najvece_tezine_maska = 1 << (sizeof(unsigned) * 8 - 1)
13     ;
14     int i;
15
16     /* n puta se vrsti rotaciju za jedan bit u levo. U svakoj iteraciji
17      se odredi bit na poziciji najvece tezine, a potom se pomera
18      binarna reprezentacija trenutne vrednosti promenljive x u levo
19      za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
20      na vrednost koju je imao bit na poziciji najvece tezine koji je
21      istisnut pomeranjem */
22     for (i = 0; i < n; i++) {
23         bit_najvece_tezine = x & bit_najvece_tezine_maska;
24         x = x << 1 | (bit_najvece_tezine ? 1 : 0);
25     }
26
27     /* Vraca se dobijena vrednost */
28     return x;
29 }
30
31 /* Funkcija neoznacen broj x rotira u desno za n mesta. */
32 unsigned rotiraj_udesno(unsigned x, unsigned n)
33 {
34     unsigned bit_najmanje_tezine;
35     int i;
36
37     /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
38      iteraciji se odredjuje bit na poziciji najmanje tezine broja x,
39      zatim tako odredjeni bit se pomera u levo tako da bit na
        poziciji najmanje tezine dodje do pozicije najveće tezine.
        Zatim, nakon pomeranja binarne reprezentacije trenutne vrednosti
```

```

41     promenljive x za 1 u desno, bit na poziciji najveće tezine se
43     postaljva na vrednost vec zapamćenog bita koji je bio na
44     poziciji najmanje tezine. */
45     for (i = 0; i < n; i++) {
46         bit_najmanje_tezine = x & 1;
47         x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
48     }
49
50     /* Vraca se dobijena vrednost */
51     return x;
52 }
53
54 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
55 argument funkcije x označeni ceo broj */
56 int rotiraj_udesno_oznaceni(int x, unsigned n)
57 {
58     unsigned bit_najmanje_tezine;
59     int i;
60
61     /* U svakoj iteraciji se određuje bit na poziciji najmanje tezine
62     i smesta u promenljivu bit_najmanje_tezine. Kako je x označen
63     ceo broj, tada se prilikom pomeranja u desno vrši aritmeticko
64     pomeranje i cuva se znak broja. Dakle, razlikuju se dva slučaja
65     u zavisnosti od znaka broja x. Nije dovoljno da se ova provera
66     izvrši pre petlje, s obzirom da rotiranjem u desno na poziciju
67     najveće tezine može doći i 0 i 1, nezavisno od početnog znaka
68     broja smestenog u promenljivu x. */
69     for (i = 0; i < n; i++) {
70         bit_najmanje_tezine = x & 1;
71
72         if (x < 0)
73             *****
74             Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
75             na poziciji najveće tezine. Na primer ako je x
76             1010 1011 1100 1101 1110 0001 0010 001b
77                 (sa b je označen ili 1 ili 0 na poziciji najmanje tezine)
78             Onda je sadržaj promenljive bit_najmanje_tezine:
79             0000 0000 0000 0000 0000 0000 000b
80             Nakon siftovanja sadržaja promenljive x za 1 u desno
81             1101 0101 1110 0110 1111 0000 1001 0001
82             Kako bi umesto 1 na poziciji najveće tezine u trenutnoj binarnoj
83             reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
84             poziciju najveće tezine jer bi se time dobile 0, a u ovom slučaju
85             su potrebne jedinice zbog bitovskog & zato se prvo vrši
86             komplementiranje, a zatim pomeranje
87             ~bit_najmanje_tezine << (sizeof(int)*8 -1)
88             B000 0000 0000 0000 0000 0000 0000
89             gde B označava -b.
90             Potom se ponovo vrši komplementiranje kako bi se b nalazilo na
91             poziciji najveće tezine i sve jedinice na ostalim pozicijama
92             ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
93             b111 1111 1111 1111 1111 1111 1111

```

```

93  ****
94      x = (x >> 1) & ~(~bit_najmanje_tezine << (sizeof(int) * 8 - 1))
95      ;
96      else
97          x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
98      }
99
100     /* Vraca se dobijena vrednost */
101    return x;
102}
103int main()
104{
105    unsigned x, k;
106
107    /* Ucitavaju se vrednosti sa standardnog ulaza */
108    printf("Unesite neoznacen ceo broj x:");
109    scanf("%x", &x);
110    printf("Unesite neoznacen ceo broj k:");
111    scanf("%x", &k);
112
113    /* Ispisuje se binarna reprezentacija broja x */
114    printf("x = ", "");
115    stampaj_bitove(x);
116
117    /* Testira se rad napisanih funkcija */
118    printf("rotiraj_ulevo(%10u,%10u) %10s= ", x, k, "");
119    stampaj_bitove(rotiraj_ulevo(x, k));
120
121    printf("rotiraj_udesno(%10u,%10u) %9s= ", x, k, "");
122    stampaj_bitove(rotiraj_udesno(x, k));
123
124    printf("rotiraj_udesno_oznaceni(%10u,%10u) = ", x, k);
125    stampaj_bitove(rotiraj_udesno_oznaceni(x, k));
126
127    return 0;
128}

```

**Rešenje 1.10** NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```

1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija vraca vrednost cija je binarna reprezentacija slika u
5    ogledalu binarne reprezentacije broja x. */
6 unsigned ogledalo(unsigned x)
7 {
8     unsigned najnizi_bit;
9     unsigned rezultat = 0;

```

```

11   int i;
12   /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuce
13      vrednosti broja x se odredjuje i pamti u promenljivoj
14      najnizi_bit, nakon cega se na promenljivu x primeni pomeranje u
15      desno */
16   for (i = 0; i < sizeof(x) * 8; i++) {
17     najnizi_bit = x & 1;
18     x >>= 1;
19     /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
20        postavljeni bitovi dobijaju vecu poziciju. Novi bit se
21        postavlja na najnizu poziciju */
22     rezultat <= 1;
23     rezultat |= najnizi_bit;
24   }
25
26   /* Vraca se dobijena vrednost */
27   return rezultat;
28 }
29
30 int main()
31 {
32   int broj;
33
34   /* Ucitava se broj sa ulaza */
35   scanf("%x", &broj);
36
37   /* Ispisuje se njegova binarna reprezentacija */
38   stampaj_bitove(broj);
39
40   /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
41      funkcije ogledalo */
42   stampaj_bitove(ogledalo(broj));
43
44   return 0;
45 }
```

### Rešenje 1.11

```

#include <stdio.h>
2
3   /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
4      jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
5   int broj_01(unsigned int n)
6   {
7
8     int broj_nula, broj_jedinica;
9     unsigned int maska;
10
11    broj_nula = 0;
12    broj_jedinica = 0;
```

```

14  /* Maska je inicializovana tako da moze da analizira bit najvece
   tezine */
16  maska = 1 << (sizeof(unsigned int) * 8 - 1);

18  /* Cilj je proći kroz sve bitove broja x, zato se maska u svakoj
   iteraciji pomera u desno pa će jedini bit koji je postavljen na
   1 biti na svim pozicijama u binarnoj reprezentaciji maske */
20  while (maska != 0) {
22
24      /* Provera da li se na poziciji koju određuje maska nalazi 0 ili
       1 i uveća se odgovarajući brojac */
26      if (n & maska) {
27          broj_jedinica++;
28      } else {
29          broj_nula++;
30      }
32
34      /* Pomera se maska u desnu stranu */
35      maska = maska >> 1;
36
38  }

40  int main()
42  {
43      unsigned int n;
44
45      /* Ucitava se broj sa ulaza */
46      scanf("%u", &n);

48      /* Ispisuje se rezultat */
49      printf("%d\n", broj_01(n));
50
51      return 0;
52  }

```

### Rešenje 1.12

```

#include <stdio.h>

/* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju u
   binarnom zapisu celog čneoznaenog broja x */
int broj_parova(unsigned int x)
{
    int broj_parova;

```

```

10     unsigned int maska;
12
13     /* Vrednost promenljive koja predstavlja broj parova se
14      inicijalizuje na 0 */
15     broj_parova = 0;
16
17     /* Postavlja se maska tako da moze da procitamo da li su dva
18      najmanja bita u zapisu broja x 11 */
19     /* Binarna reprezentacija broja 3 je 000....00011 */
20     maska = 3;
21
22     while (x != 0) {
23
24         /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
25          */
26         if ((x & maska) == maska) {
27             broj_parova++;
28         }
29
30         /* Pomeranje se broj u desnu stranu da bi se u narednoj iteraciji
31            proveravao sledeći par bitova. Pomeranjem u desno bit najveće
32            tezine se popunjava nulom jer je x neoznacen broj */
33         x = x >> 1;
34     }
35
36     /* Vraca se dobijena vrednost */
37     return broj_parova;
38 }
39
40 int main()
41 {
42     unsigned int x;
43
44     /* Ucitava se broj sa ulaza */
45     scanf("%u", &x);
46
47     /* Ispisuje se rezultat */
48     printf("%d\n", broj_parova(x));
49
50     return 0;
51 }
```

### Rešenje 1.14

```

1 #include <stdio.h>
2
3 /* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1 jer
4    su za svaku heksadekadinu cifru potrebne 4 binarne cifre i jedna
5    dodatna pozicija za terminirajuću nulu.
6    Prethodni izraz se može zapisati kao sizeof(unsigned int)*2+1. */
```

```

1 #define MAKS_DUZINA sizeof(unsigned int)*2 +1
2
3 /* Funkcija za neoznacen broj x formira nisku s koja sadrzi njegov
4    heksadekadni zapis */
5 void prevod(unsigned int x, char s[])
6 {
7
8     int i;
9     unsigned int maska;
10    int vrednost;
11
12
13    /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
14       maska za citanje 4 uzastopne cifre */
15    maska = 15;
16
17
18    /***** Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
19       tezine. Na primer za broj cija je binarna reprezentacija
20       0000000001101000100001111010101
21       u prvom koraku se citaju bitovi izdvojeni sa <...>:
22       000000000110100010000111101<0101>
23       u drugom koraku:
24       00000000011010001000011<1101>0101
25       u trećem koraku:
26       0000000001101000100<0011>11010101 i tako redom...
27
28
29    Indeks i označava poziciju na koju se smesta vrednost.
30    *****/
31    for (i = MAKS_DUZINA - 2; i >= 0; i--) {
32        /* Vrednost izdvojene cifre */
33        vrednost = x & maska;
34
35        /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
36           dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
37           10 do 15 odgovarajuci karakter se dobija tako sto se prvo
38           oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na takoj
39           dobijenu vrednost doda ASCII kod 'A' (time se dobija
40           odgovarajuce slovo 'A', 'B', ... 'F') */
41        if (vrednost < 10) {
42            s[i] = vrednost + '0';
43        } else {
44            s[i] = vrednost - 10 + 'A';
45        }
46
47        /* Primenljiva x se pomera za 4 bita u desnu stranu i time se u
48           narednoj iteraciji posmatraju sledeca 4 bita */
49        x = x >> 4;
50    }
51
52    /* Upisuje se terminirajuca nula */
53    s[MAKS_DUZINA - 1] = '\0';
54}

```

```

60 int main()
61 {
62     unsigned int x;
63     char s[MAKS_DUZINA];
64
65     /* Ucitava se broj sa ulaza */
66     scanf("%u", &x);
67
68     /* Poziva se funkcija za prevodjenje */
69     prevod(x, s);
70
71     /* I stampa se dobijena niska */
72     printf("%s\n", s);
73
74     return 0;
75 }
```

**Rešenje 1.17**

```

# include <stdio.h>
1
/************************************************************
2 Resenje linearne slozenosti:
3     x^0 = 1
4     x^k = x * x^(k-1)
5
6 int stepen(int x, int k)
7 {
8     if (k == 0)
9         return 1;
10
11     return x * stepen(x, k - 1);
12     /* kraci zapis: return k == 0 ? 1 : x * stepen(x,k-1); */
13 }
14
15 /************************************************************
16 Resenje logaritamske slozenosti:
17     x^0 = 1;
18     x^k = x * (x^2 )^(k/2) , za neparno k
19     x^k = (x^2)^(k/2) , za parno k
20
21 Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
22     se doslo do rezultata, i stoga je efikasnije.
23
24 int stepen_2(int x, int k)
25 {
26     if (k == 0)
27         return 1;
28
29     /* Ako je stepen paran */
30     if ((k % 2) == 0)
```

```

32     return stepen_2(x * x, k / 2);

34 /* Inace (ukoliko je stepen neparan) */
35 return x * stepen_2(x * x, k / 2);
36 }

38 int main()
{
39     int x, k, ind;

42     /* Ucitava se redni broj funkcije koja ce se primeniti */
43     printf("Unesite redni broj funkcije (1/2):\n");
44     scanf("%d", &ind);

46     /* Ucitavaju se vrednosti za x i k */
47     printf("Unesite broj x:\n");
48     scanf("%d%d", &x);
49     printf("Unesite broj k:\n");
50     scanf("%d%d", &k);

52     /* Ispisuje se vrednost koju vraca odgovarajuca funkcija */
53     if (x == 1)
54         printf("%d\n", stepen(x, k));
55     else if (x == 2)
56         printf("%d\n", stepen_2(x, k));
57     else {
58         fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
59         exit(EXIT_FAILURE);
60     }
61     return 0;
62 }
```

### Rešenje 1.18

```

#include <stdio.h>

2 /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
4 (posrednu) rekurziju */

6 /* Deklaracija funkcije neparan mora da bude navedena jer se ta
8 funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
10 definicije. Funkcija je mogla biti deklarisana i u telu funkcije
12 paran. */
14 unsigned neparan(unsigned n);

16 /* Funkcija vraca 1 ako broj n ima paran broj cifara, inace vraca 0
17 */
18 unsigned paran(unsigned n)
19 {
20     if (n <= 9)
21         return 0;
```

```

18     else
19         return neparan(n / 10);
20     }
21 }
22 /* Funkcija vraca 1 ako broj n ima neparan broj cifara, inace vraca
23 0 */
24 unsigned neparan(unsigned n)
25 {
26     if (n <= 9)
27         return 1;
28     else
29         return paran(n / 10);
30 }

31 int main()
32 {
33     int n;

34     /* Ucitava se ceo broj */
35     scanf("%d", &n);

36     /* Ispisuje se rezultat */
37     printf("Uneti broj ima %sparan broj cifara.\n",
38           (paran(n) == 1 ? "" : "ne"));

39     return 0;
40 }
```

### Rešenje 1.19

```

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

3 /*
4  * Pomocna funkcija koja izracunava n! * result. Koristi repnu
5  * rekurziju. Result je argument u kome se akumulira do tada
6  * izracunatu vrednost faktorijela. Kada dodje do izlaza iz rekurzije
7  * iz rekurzije potrebno je da vratimo result. */
8 int faktorijel_repna(int n, int result)
9 {
10     if (n == 0)
11         return result;
12
13     return faktorijel_repna(n - 1, n * result);
14 }
15
16 /* U sledecim funkcijama je prikazan postupak oslobadjanja od
17  repne rekurzije koja postoji u funkciji faktorijel_repna. */

18 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
19  naredbama kojima se vrednost argumenta funkcije postavlja na
```

```

21     vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
22     goto naredbe za vracanje na pocetak tela funkcije. */
23 int faktorijel_repna_v1(int n, int result)
24 {
25     pocetak:
26         if (n == 0)
27             return result;
28
29         result = n * result;
30         n = n - 1;
31         goto pocetak;
32 }
33
34 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
35    praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
36    iterativno resenje bez bezuslovnih skokova */
37 int faktorijel_repna_v2(int n, int result)
38 {
39     while (n != 0) {
40         result = n * result;
41         n = n - 1;
42     }
43
44     return result;
45 }
46
47 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
48    broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
49    ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
50    udobnosti korisnika, jer je sasvim prirodno da za faktorijel
51    zahteva samo 1 parametar. Funkcija faktorijel izracunava  $n!$ , tako
52    sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
53    faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
54    funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
55    poziv faktorijel_repna sa pozivom faktorijel_repna_v1, a zatim sa
56    pozivom funkcije faktorijel_repna_v2. */
57 int faktorijel(int n)
58 {
59     return faktorijel_repna(n, 1);
60 }
61
62 int main()
63 {
64     int n;
65
66     /* Ucitava se ceo broj */
67     printf("Unesite n (<= 12): ");
68     scanf("%d", &n);
69     if (n > 12) {
70         fprintf(stderr, "Greska: nedozvoljena vrednost za n!\n");
71         exit(EXIT_FAILURE);
72     }

```

```

73  /* Ispisuje se rezultat */
74  printf("%d! = %d\n", n, faktorijel(n));
75
76  exit(EXIT_SUCCESS);
77 }

```

## Rešenje 1.21

```

1 #include <stdio.h>

3 /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
4 int f_iterativna(int n, int a, int b)
5 {
6     int i;
7     int f_0 = 0;
8     int f_1 = 1;
9     int tmp;
10
11    if (n == 0)
12        return 0;
13
14    for (i = 2; i <= n; i++) {
15        tmp = a * f_1 + b * f_0;
16        f_0 = f_1;
17        f_1 = tmp;
18    }
19
20    return f_1;
21 }

23 /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
24 int f_rekurzivna(int n, int a, int b)
25 {
26     /* Izlaz iz rekurzije */
27     if (n == 0 || n == 1)
28         return n;
29
30     /* Rekurzivni pozivi */
31     return a * f_rekurzivna(n - 1, a, b) +
32           b * f_rekurzivna(n - 2, a, b);
33 }
34
35 /* NAPOMENA: U slučaju da se rekurzijom problem svodi na vise manjih
36 podproblema koji se mogu preklapati, postoji opasnost da se
37 pojedini podproblemi manjih dimenzija resavaju veci broj puta.
38 Npr.  $F(20) = a*F(19) + b*F(18)$ , a  $F(19) = a*F(18) + b*F(17)$ , tj.
39 problem  $F(18)$  se resava dva puta! Problemi manjih
40 dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
41 problem je kombinacija rekurzije sa dinamickim programiranjem.

```

```

43 Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
44 memoriji (obicno u nizovima ili matricama), odakle se koriste ako
45 tokom resavanja ponovo budu potrebni.

47 U narednoj funkciji vec izracunati clanovi niza se cuvaju u
48 statickom nizu celih brojeva, jer se staticki niz ne smesta
49 na stek, kao sto je to slucaj sa lokalnim promenljivama, vec na
50 segment podataka, odakle je dostupan svim pozivima
51 rekurzivne funkcije. */

53 /* c) Funkcija racuna n-ti broj niza F - napredna rekurzivna
54 verzija */
55 int f_napredna(int n, int a, int b)
{
56     /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
57     staticke promenljive na podrazumevane vrednosti. Stoga, elemente
58     celobrojnog niza inicijalizuje na 0 */
59     static int f[20];

60     /* Ako je podproblem vec ranije resen, koristi se resenje koje je
61     vec izracunato i */
62     if (f[n] != 0)
63         return f[n];

64     /* Izlaz iz rekurzije */
65     if (n == 0 || n == 1)
66         return f[n] = n;

67     /* Rekurzivni pozivi */
68     return f[n] =
69         a * f_napredna(n - 1, a, b) + b * f_napredna(n - 2, a, b);
}

75 int main()
76 {
77     int n, a, b, ind;
78
79     /* Unosi se redni broj funkcije koja ce se primeniti */
80     printf("Unesite redni broj funkcije koju zelite:\n");
81     printf("1 - iterativna\n");
82     printf("2 - rekurzivna\n");
83     printf("3 - rekurzivna napredna\n");
84     scanf("%d", &ind);

85     /* Ucitavaju se koeficijenti a i b */
86     printf("Unesite koeficijente:\n");
87     scanf("%d%d", &a, &b);

88     /* Ucitava se broj n */
89     printf("Unesite koji clan niza se racuna:\n");
90     scanf("%d", &n);
91
92
93

```

```

95  /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
96   funkcijske f_iterativna, f_rekursivna ili f_napredna */
97  if (ind == 0)
98    printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
99  else if (ind == 1)
100   printf("F(%d) = %d\n", n, f_rekursivna(n, a, b));
101 else
102   printf("F(%d) = %d\n", n, f_napredna(n, a, b));
103
104 return 0;
105 }
```

**Rešenje 1.22**

```

1 #include <stdio.h>
2
3 /* Funkcija određuje zbir cifara zadatog broja x */
4 int zbir_cifara(unsigned int x)
5 {
6   /* Izlazak iz rekurzije: ako je broj jednocifern */
7   if (x < 10)
8     return x;
9
10  /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
11   poslednje cifre + poslednja cifra tog broja */
12  return zbir_cifara(x / 10) + x % 10;
13 }
14
15 int main()
16 {
17   unsigned int x;
18
19   /* Ucitava se ceo broj */
20   scanf("%u", &x);
21
22   /* Ispisuje se zbir cifara ucitanog broja */
23   printf("%d\n", zbir_cifara(x));
24
25   return 0;
26 }
```

**Rešenje 1.23**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAKS_DIM 100
5
6 /* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
```

```

8     suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
9     element niza. */
10    int suma_niza_1(int *a, int n)
11    {
12        if (n <= 0)
13            return 0;
14
15        return suma_niza_1(a, n - 1) + a[n - 1];
16    }
17
18 /* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
19 jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
20 elementa niza i sume preostalih n-1 elementa. */
21    int suma_niza_2(int *a, int n)
22    {
23        if (n <= 0)
24            return 0;
25
26        return a[0] + suma_niza_2(a + 1, n - 1);
27    }
28
29    int main()
30    {
31        int a[MAKS_DIM];
32        int n, i = 0, ind;
33
34        /* Ucitava se redni broj funkcije */
35        printf("Unesite redni broj funkcije (1 ili 2):\n");
36        scanf("%d", &ind);
37
38        /* Ucitava se broj elemenata niza */
39        printf("Unesite dimenziju niza:");
40        scanf("%d", &n);
41
42        /* Ucitava se n elemenata niza. */
43        printf("Unesite elemente niza:");
44        for (i = 0; i < n; i++)
45            scanf("%d", &a[i]);
46
47        /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
48         funkcije suma_niza_1, ondosno suma_niza_2 */
49        if (ind == 1)
50            printf("Suma elemenata je %d\n", suma_niza_1(a, n));
51        else if (ind == 2)
52            printf("Suma elemenata je %d\n", suma_niza_2(a, n));
53        else{
54            fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
55            exit(EXIT_FAILURE);
56        }
57
58        exit(EXIT_SUCCESS);
59    }

```

## Rešenje 1.24

```

1 #include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Rekurzivna funkcija koja određuje maksimum celobrojnog niza niz
   dimenzije n */
5 int maksimum_niza(int niz[], int n)
6 {
7     /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je
       ujedno i jedini element niza */
8     if (n == 1)
9         return niz[0];
10
11    /* Resavanje problema manje dimenzije */
12    int max = maksimum_niza(niz, n - 1);
13
14    /* Na osnovu poznatog resenja problema dimenzije n-1, resava se
       problem dimenzije n */
15    return niz[n - 1] > max ? niz[n - 1] : max;
16}
17
18 int main()
19 {
20     int brojevi[MAKS_DIM];
21     int n;
22
23     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
       Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
       ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
       i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
24     int i = 0;
25     while (scanf("%d", &brojevi[i]) != EOF) {
26         i++;
27         if (i == MAKS_DIM)
28             break;
29     }
30     n = i;
31
32     /* Stampa se maksimum unetog niza brojeva */
33     printf("%d\n", maksimum_niza(brojevi, n));
34
35     return 0;
36 }
37
38
39
40
41
42

```

## Rešenje 1.25

```

1 #include <stdio.h>
2 #define MAKS_DIM 256
3

```

```

5  /* Funkcija koja izracunava skalarni proizvod dva data vektora */
6  int skalarno(int a[], int b[], int n)
7  {
8      /* Izlazak iz rekurzije: vektori su duzine 0 */
9      if (n == 0)
10         return 0;
11
12      /* Na osnovu resenja problema dimenzije n-1, resava se problem
13         dimenzije n primenom definicije skalarnog proizvoda
14         a*b = a[0]*b[0] + a[1]*b[1] +...+ a[n-2]*a[n-2] + a[n-1]*a[n-1]
15         Dakle, skalarni proizvod dva vektora duzine n se dobija kada se
16         na skalarni proizvod dva vektora duzine n-1 koji se dobiju od
17         polazna dva vektora otklanjanjem poslednjih elemenata, doda
18         proizvod poslednja dva elementa polaznih vektora. */
19      else
20          return skalarno(a, b, n - 1) + a[n - 1] * b[n - 1];
21
22
23  int main()
24  {
25      int i, a[MAKS_DIM], b[MAKS_DIM], n;
26
27      /* Unosi se dimenzija nizova */
28      printf("Unesite dimenziju nizova:");
29      scanf("%d", &n);
30
31      /* Provera da li je dimenzija niza odgovarajuca */
32      if (n < 0 || n > MAKS_DIM) {
33          printf("Dimenzija mora biti prirodan broj <= %d!\n", MAKS_DIM);
34          return 0;
35      }
36
37      /* A zatim i elementi nizova */
38      printf("Unesite elemente prvog niza:");
39      for (i = 0; i < n; i++)
40          scanf("%d", &a[i]);
41
42      printf("Unesite elemente drugog niza:");
43      for (i = 0; i < n; i++)
44          scanf("%d", &b[i]);
45
46      /* Ispisuje se rezultat skalarnog proizvoda dva ucitana niza */
47      printf("Skalarni proizvod je %d\n", skalarno(a, b, n));
48
49      return 0;
50  }

```

### Rešenje 1.26

```

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

```

```

4  /* Funkcija koja racuna broj pojavljivanja elementa x u nizu a duzine
   n */
6  int br_pojave(int x, int a[], int n)
{
8    /* Izlazak iz rekurzije: za niz duzine jedan broj pojava broja x u
       nizu je 1 ukoliko je jedini element a[0] bas x ili 0 inace */
10   if (n == 1)
11     return a[0] == x ? 1 : 0;
12
13   /* U promenljivu bp se smesta broj pojave broja x u prvih n-1
      elemenata niza a. Ukupan broj pojavljivanja broja x u celom nizu
      a je jednak bp uvecanom za jedan ukoliko je se na poziciji n-1 u
      nizu a nalazi broj x */
14   int bp = br_pojave(x, a, n - 1);
15   return a[n - 1] == x ? 1 + bp : bp;
16 }
17
18 int main()
19 {
20   int x, a[MAKS_DIM];
21   int n, i = 0;
22
23   /* Ucitava se ceo broj */
24   printf("Unesite ceo broj:");
25   scanf("%d", &x);
26
27   /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz.
      Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
      ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
      i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
28   printf("Unesite elemente niza:");
29   i = 0;
30   while (scanf("%d", &a[i]) != EOF) {
31     i++;
32     if (i == MAKS_DIM)
33       break;
34   }
35   n = i;
36
37   /* Ispisuje se broj pojavljivanja */
38   printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
39
40   return 0;
41 }
```

**Rešenje 1.27**

```

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

```

4  /* Funkcija koja proverava da li su tri zadata broja uzastopni
5   * clanovi niza */
6  int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
7  {
8      /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i vraca
9       * se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi niza */
10     if (n < 3)
11         return 0;
12
13     /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
14      dovoljno je da su oni poslednja tri clana niza ili da se oni
15      rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
16     return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
17           || tri_uzastopna_clana(x, y, z, a, n - 1);
18 }
19
20 int main()
21 {
22     int x, y, z, a[MAKS_DIM];
23     int n;
24
25     /* Ucitavaju se tri cela broja za koje se ispituje da li su
26      uzastopni clanovi niza */
27     printf("Unesite tri cela broja:");
28     scanf("%d%d%d", &x, &y, &z);
29
30     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
31      Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
32      ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
33      i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
34     printf("Unesite elemente niza:");
35     int i = 0;
36     while (scanf("%d", &a[i]) != EOF) {
37         i++;
38         if (i == MAKS_DIM)
39             break;
40     }
41     n = i;
42
43     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana ispisuje
44      se odgovarajuci poruka */
45     if (tri_uzastopna_clana(x, y, z, a, n))
46         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
47     else
48         printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
49
50     return 0;
51 }
```

## Rešenje 1.28

```

1 #include <stdio.h>
2
3 /* Funkcija koja broji bitove postavljene na 1. */
4 int prebroj(int x)
5 {
6     /* Izlaz iz rekurzije */
7     if (x == 0)
8         return 0;
9
10    /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x je
11       postavljen na 1. Koriscenjem odgovarajuce maske proverava se
12       vrednost bita na poziciji najvece tezine i na osnovu toga se
13       razlikuju dva slucaja. Ukoliko je na toj poziciji nula, onda je
14       broj jedinica u zapisu x isti kao broj jedinica u zapisu broja
15       x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0. Ako je
16       na poziciji najvece tezine jedinica, rezultat dobijen
17       rekurzivnim pozivom funkcije za x<<1 treba uvecati za jedan.
18       Za rekurzivni poziv se salje vrednost koja se dobija kada se x
19       pomeri u levo. Napomena: argument funkcije x je ozначен ceo
20       broj, usled cega se ne koristi pomeranje udesno, jer funkciji
21       moze biti prosledjen i negativan broj. Iz tog razloga,
22       odlucujemo se da proveramo najvisi, umesto najnizeg bita */
23    if (x & (1 << (sizeof(x) * 8 - 1)))
24        return 1 + prebroj(x << 1);
25    else
26        return prebroj(x << 1);
27    //*****
28    Krace zapisano
29    return ((x & (1 << (sizeof(x)*8-1))) ? 1 : 0) + prebroj(x<<1);
30    //*****
31 }
32
33 int main()
34 {
35     int x;
36
37     /* Ucitava se ceo broj */
38     scanf("%x", &x);
39
40     /* Ispisuje se rezultat */
41     printf("%d\n", prebroj(x));
42
43     return 0;
44 }
```

**Rešenje 1.30**

```

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

```

1  /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
2   */
4 int maks_oktalna_cifra(unsigned x)
{
6  /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
7   vrednost najveće oktalne cifre u broju 0 */
8  if (x == 0)
9    return 0;
10
11 /* Odredjivanje poslednje oktalne cifre u broju */
12 int poslednja_cifra = x & 7;
13
14 /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
15   izbriše poslednja oktalna cifra */
16 int maks_bez_poslednje_cifre = maks_oktalna_cifra(x >> 3);
17
18 return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
19       : maks_bez_poslednje_cifre;
20}
21
22 int main()
23 {
24  unsigned x;
25
26  /* Ucitava se neoznacen ceo broj */
27  scanf("%u", &x);
28
29  /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
30  printf("%d\n", maks_oktalna_cifra(x));
31
32  return 0;
33}

```

### Rešenje 1.31

```

1 #include <stdio.h>
2
3 /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
4   broju */
5 int maks_heksadekadna_cifra(unsigned x)
6 {
7  /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8   vrednost najveće heksadekadne cifre u broju 0 */
9  if (x == 0)
10    return 0;
11
12  /* Odredjivanje poslednje heksadekadne cifre u broju */
13  int poslednja_cifra = x & 15;
14
15  /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
16   njega izbriše poslednja heksadekadna cifra */

```

```

18     int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);
19
20     return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
21         : maks_bez_poslednje_cifre;
22 }
23
24 int main()
25 {
26     unsigned x;
27
28     /* Ucitava se neoznacen ceo broj */
29     scanf("%u", &x);
30
31     /* Ispisuje se vrednost najvece heksadekadne cifre unetog broja */
32     printf("%d\n", maks_heksadekadna_cifra(x));
33
34     return 0;
35 }
```

**Rešenje 1.32**

```

1 #include <stdio.h>
2 #include <string.h>
3
4 /* Niska moze imati najvise 31 karaktera + 1 za terminalnu nulu */
5 #define MAKS_DIM 32
6
7 /* Funkcija ispituje da li je zadata niska duzine n palindrom */
8 int palindrom(char s[], int n)
9 {
10    /* Izlaz iz rekurzije - trivijalno, niska duzine 0 ili 1 je
11       palindrom */
12    if ((n == 1) || (n == 0))
13        return 1;
14
15    /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
16       poslednji karakter i da je palindrom niska koja nastaje kada se
17       polaznoj nisci otklane prvi i poslednji karakter */
18    return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
19 }
20
21 int main()
22 {
23     char s[MAKS_DIM];
24     int n;
25
26     /* Ucitavanje niske sa standardnog ulaza */
27     scanf("%s", s);
28
29     /* Odredjuje se duzina niske */
30     n = strlen(s);
```

```

32  /* Ispisuje se poruka da li je niska palindrom ili nije */
33  if (palindrom(s, n))
34      printf("da\n");
35  else
36      printf("ne\n");
37
38  return 0;
}

```

### Rešenje 1.33

```

#include <stdio.h>
#include <stdlib.h>
#define MAKS_DUZINA_PERMUTACIJE 15

/* Funkcija koja ispisuje elemente niza a duzine n */
void ispisi_niz(int a[], int n)
{
    int i;

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

/* Funkcija koja proverava da li se broj x nalazi u nizu a duzine n
 */
int koriscen(int a[], int n, int x)
{
    int i;

    /* Obilaze se svi elementi niza */
    for (i = 1; i <= n; i++)
        /* Ukoliko se naidje na trazenu vrednost, pretraga se prekida */
        if (a[i] == x)
            return 1;

    /* Zaključuje se da broj nije pronadjen */
    return 0;
}

/* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n} dobija
   kao argument niz a[] u koji se smesta permutacija, broj m označava
   da se u okviru tog poziva funkciju na m-tu poziciju u permutaciji
   smesta jedan od preostalih celih brojeva, n je velicina skupa koji
   se permutuje. Funkciju se inicijalno poziva sa argumentom m = 1,
   jer formiranje permutacije pocinje od pozicije broj 1. Stoga, a[0]
   se ne koristi. */
void permutacija(int a[], int m, int n)
{

```

```

13     int i;
14
15     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
16      premasila velicinu skupa, onda se svi brojevi vec nalaze u
17      permutaciji i ispisuje se permutacija. */
18
19     if (m > n) {
20         ispisni_niz(a, n);
21         return;
22     }
23
24     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
25      mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
26      Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
27      ovako postavljenom pocetku permutacije. Kada se to zavrsi, vrsti
28      se provera da li postoji jos neki broj koji moze da se stavi na
29      m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
30      funkcija zavrsava sa radom. Ukoliko takav broj postoji, onda se
31      ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
32      ali sada sa drugacije postavljenim prefiksom. */
33
34     for (i = 1; i <= n; i++) {
35         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
36            pozicije, onda se on postavlja na poziciju m i poziva se
37            ponovo funkcija da dopuni ostatak permutacije posle upisivanja
38            i na poziciju m. Inace, nastavlja se dalje, trazeci broj koji
39            se nije pojavio do sada u permutaciji. */
40         if (!koriscen(a, m - 1, i)) {
41             a[m] = i;
42             permutacija(a, m + 1, n);
43         }
44     }
45
46 }
47
48 int main(void)
49 {
50     int n;
51     int a[MAKS_DUZINA_PERMUTACIJE+1];
52
53     /* Ucitava se broja n i provera se da li je u odgovarajucem opsegu
54      */
55     scanf("%d", &n);
56     if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
57         fprintf(stderr,
58                 "Duzina permutacije mora biti broj iz intervala [0, %d]!\n",
59                 MAKS_DUZINA_PERMUTACIJE);
60         exit(EXIT_FAILURE);
61     }
62
63     /* Ispisuju se permutacije duzine n */
64     permutacija(a, 1, n);
65
66     exit(EXIT_SUCCESS);
67
68 }

```

```
}
```

### Rešenje 1.34

```

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

4 /* Rekurzivna funkcija za racunanje binomnog koeficijenta */
5 int binomni_koeficijent(int n, int k)
6 {
7     /* Ukoliko je k=0 ili k=n, onda je binomni koeficijent 0.
8        Ukoliko je k strog izmedju 0 i n, onda se koristi formula
9        bk(n,k) = bk(n-1,k-1) + bk(n-1,k)
10       koja se moze izvesti iz definicije binomnog koeficijenata */
11       return (0 < k && k < n) ?
12           binomni_koeficijent(n - 1, k - 1) + binomni_koeficijent(n - 1,
13                           k) : 1;
14 }

16 **** Iterativno izracunavanje datog binomnog koeficijenta
17
18 int binomni_koeficijent (int n, int k) {
19     int i, j, b;
20
21     for (b=i=1, j=n; i<=k; b = b * j-- / i++);
22
23     return b;
24 }
25
26 Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
27 resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
28 ****
29
30 /* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
31    zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
32    ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
33    veca od sume elemenata prethodne hipotenuze. */
34 int suma_elemenata_hipotenuze(int n)
35 {
36     return n > 0 ? 2 * suma_elemenata_hipotenuze(n - 1) : 1;
37 }
38
39 int main()
40 {
41     int n, k, i, d, r;
42
43     /* Ucitavaju se brojevi d i r */
44     scanf("%d %d", &d, &r);
45
46     /* Ispisuje se Paskalov trougao */

```

```
48     putchar('\n');
49     for (n = 0; n <= d; n++) {
50         for (i = 0; i < d - n; i++)
51             printf(" ");
52         for (k = 0; k <= n; k++)
53             printf("%4d", binomni_koeficijent(n, k));
54         putchar('\n');
55     }
56
57     /* Provera da li je r nenegativan */
58     if (r < 0) {
59         fprintf(stderr,
60                 "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
61         );
62         exit(EXIT_FAILURE);
63     }
64
65     /* Ispisuje se suma elemenata hipotenuze */
66     printf("%d\n", suma_elemenata_hipotenuze(r));
67
68     exit(EXIT_SUCCESS);
69 }
```

# 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: neodgovarajuća dimenzija niza.
```

[Rešenje 2.1]

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

## 2.1 Pokazivačka aritmetika

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

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

Primer 4

```
INTERAKCIJA SA PROGRAMOM:  
Unesite dimenziju niza: 101  
Greska: neodgovarajuća 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 argументе,
koriscenjem indeksне ili pokazivачке
sintakсе (I ili P)? I
Argumenti komandne linije su:
0 ./a.out
1 prvi
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 argументе,
koriscenjem indeksне или pokazivачке
sintаксе (I или 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. Smatratи 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 zavrsavaju na ke  
  
INTERAKCIJA SA PROGRAMOM:  
Broj reci koje se zavrsavaju 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  $n \times n$ .

- 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 broj vrsta (ili kolona) kvadratne matrice  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrice. Na standardni izlaz ispisati učitanu matricu, a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 -2 3
4 -5 6
7 -8 9
Trag matrice je 5.
Euklidска норма матрице је 16.88.
Вандижагонална норма матрице је 11.
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 0
Greska: neodgovarajuća dimenzija matrice.
```

[Rešenje 2.8]

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

- Napisati funkciju koja proverava da li su matrice jednake.
- Napisati funkciju koja izračunava zbir matrica.
- Napisati funkciju koja izračunava proizvod matrica.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati broj vrsta 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 broj vrsta 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: element  $i$  je u relaciji sa elementom  $j$  ukoliko se u preseku  $i$ -te vrste i  $j$ -te kolone matrice nalazi broj 1, a nije u relaciji ukoliko se tu nalazi broj 0.

- Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.

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

Napisati program koji učitava matricu iz datoteke čije se ime zadaje kao prvi argument komandne linije. U prvoj liniji datoteke nalazi se broj vrsta kvadratne 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 zatvorene relacije:
1 0 0 0
0 1 1 0
0 0 1 0
0 0 0 1
Simetricno zatvorene relacije:
1 0 0 0
0 1 1 0
0 1 1 0
0 0 0 0
Refleksivno-tranzitivno zatvorene 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  $n \times n$ .

- (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 čiji se broj vrsta  $n$  ( $0 < n \leq 32$ ) 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 3x3:  
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 najvecim elementom je 2.  
Broj negativnih elemenata matrice je 3.
```

*Primer 2*

```
POZIV: ./a.out 4
```

```
INTERAKCIJA SA PROGRAMOM:  
Unesite elemente matrice dimenzije 4x4:  
-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 najvecim 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  $n \times n$  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 broj vrsta celobrojne kvadratne matrice  $n$  ( $0 < n \leq 32$ ), a zatim i njene elemente. Na standardni izlaz ispisati rezultat primene napisane funkcije na učitanu matricu.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj vrsta 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 broj vrsta 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  $n \times m$ .

- Napisati funkciju koja učitava elemente matrice sa standardnog ulaza
- Napisati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice, u smeru kretanja kazaljke na satu.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati broj vrsta  $n$  ( $0 < n \leq 10$ ) i broj kolona  $m$  ( $0 < m \leq 10$ ) matrice, a zatim i njene elemente. 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  $n \times n$  ( $0 < n \leq 32$ ). Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati broj vrsta 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 broj vrsta 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

## 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:

- realokaciju memorije niza vršiti korišćenjem `malloc()` funkcije,
- 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 (prepostaviti 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 broj vrsta  $n$  i broj kolona  $m$  matrice (ne praviti nikakve prepostavke o njihovoj veličini), a zatim i elementi matrice. Na standardni izlaz ispisati trag matrice.

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.18]

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

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

## 2.3 Dinamička alokacija memorije

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

Napisati programe koji testiraju napisanu biblioteku.

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

*Primer 1*

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

*Primer 2*

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

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

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

[Rešenje 2.19]

**Zadatak 2.20** Data je celobrojna matrica dimenzije  $n \times m$ . Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu). Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati  $n$  i  $m$  (ne praviti nikakve pretpostavke o njihovoj veličini), zatim učitati elemente matrice i na standardni izlaz ispisati elemente ispod glavne dijagonale matrice. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

#### *Primer 1*

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

[Rešenje 2.20]

**Zadatak 2.21** Za 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. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

*Primer 1*

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

*Primer 2*

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

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

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

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

*Primer 1*

```
Poziv: ./a.out matrica.txt  
  
MATRICA.TXT  
3  
1.1 -2.2 3.3  
-4.4 5.5 -6.6  
7.7 -8.8 9.9
```

```
INTERAKCIJA SA PROGRAMOM:  
Zbir apsolutnih vrednosti ispod  
sporedne dijagonale je 25.30.  
Transformisana matrica je:  
1.10 -1.10 1.65  
-8.80 5.50 -3.30  
15.40 -17.60 9.90
```

[Rešenje 2.22]

**Zadatak 2.23** Napisati program koji na osnovu dve realne matrice dimenzija  $m \times n$  formira matricu dimenzije  $2 \cdot m \times n$  tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matrice su zapisane u datoteci **matrice.txt**. U prvom redu se nalaze dimenzije matrica  $m$  i  $n$ , u narednih  $m$  redova se nalaze vrste prve matrice, a u narednih  $m$  redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz.

*Primer 1*

```

POZIV: ./a.out matrice.txt
MATRICE.TXT
3
1.1 -2.2 3.3
-4.4 5.5 -6.6
7.7 -8.8 9.9
-1.1 2.2 -3.3
4.4 -5.5 6.6
-7.7 8.8 -9.9
INTERAKCIJA SA PROGRAMOM:
1.1 -2.2 3.3
-1.1 2.2 -3.3
-4.4 5.5 -6.6
4.4 -5.5 6.6
7.7 -8.8 9.9
-7.7 8.8 -9.9

```

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

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite elemente niza, nulu za kraj:
1 2 3 0
Trazena matrica je:
1 2 3
2 3 1
3 1 2

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite elemente niza, nulu za kraj:
-5 -2 -4 -1 0
Trazena matrica je:
-5 -2 -4 -1
-2 -4 -1 -5
-4 -1 -5 -2
-1 -5 -2 -4

```

**Zadatak 2.25** Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci *slicice.txt* se nalaze informacije o sličicama koje mu nedostaju u formatu:

*redni\_broj\_sličice ime\_reprezentacije\_kojoj\_sličica\_pripada*  
Pomožite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronade ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. NAPOMENA: Za realokaciju memorije koristiti *realloc()* funkciju.

*Primer 1*

```

SLICICE.TXT
3 Brazil
6 Nemacka
2 Kamerun
1 Brazil
2 Engleska
4 Engleska
5 Brazil

```

```

INTERAKCIJA SA PROGRAMOM:
Petru ukupno nedostaje 7 sličica.
Reprezentacija za koju je sakupio
najmanji broj sličica je Brazil.

```

**\*\* Zadatak 2.26** U datoteci `temena.txt` se nalaze tačke koje predstavljaju temena nekog  $n$ -tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom  $n$ -touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan.

*Primer 1*

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

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

*Primer 2*

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

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

## 2.4 Pokazivači na funkcije

**Zadatak 2.27** Napisati program koji tabelarno štampa vrednosti proizvoljne realne funkcije sa jednim realnim argumentom, odnosno izračunava i ispisuje vrednosti date funkcije u  $n$  ekvidistantnih tačaka na intervalu  $[a, b]$ . 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 (uključujući 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 (uključujući krajeve intervala)?
4
x cos(x)
-----
| 0.00000 | 1.00000 |
| 0.66667 | 0.78589 |
| 1.33333 | 0.23524 |
| 2.00000 | -0.41615 |
```

[Rešenje 2.27]

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

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

Primer 1

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

Primer 2

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

**Zadatak 2.29** Napisati funkciju koja određuje integral funkcije  $f(x)$  na intervalu  $[a, b]$ . Adresa funkcije  $f$  se prenosi kao parametar. Integral se računa prema formuli:

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

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

Primer 1

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

Primer 1

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

**Zadatak 2.30** Napisati funkciju koja približno izračunava integral funkcije  $f(x)$  na intervalu  $[a, b]$ . Funkcija  $f$  se prosleđuje kao parametar, a integral se procenjuje po Simpsonovoj formuli:

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

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

*Primer 1*

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

*Primer 2*

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

## 2.5 Rešenja

### Rešenje 2.1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija obrće elemente niza koriscenjem indekse sintakse */
7 void obrni_niz_v1(int a[], int n)
8 {
9     int i, j;
10
11    for (i = 0, j = n - 1; i < j; i++, j--) {
12        int t = a[i];
13        a[i] = a[j];
14        a[j] = t;
15    }
16}
17
18 /* Funkcija obrće elemente niza koriscenjem pokazivacke 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
            pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
            umanjuje za jedan, sto za posledicu ima da pokazivac
```

```

        "poslednji" sada pokazuje na element koji mu prethodi u nizu
    */
    *poslednji-- = t;
}

/***********************
 Drugi nacin za obrtanje niza

 for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
                  prvi++, poslednji--) {
    int t = *prvi;
    *prvi = *poslednji;
    *poslednji = t;
}
***********************/

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

    /* Broj elemenata niza a */
    int n;

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

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

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

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

    obrni_niz_v1(a, n);

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

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

    obrni_niz_v2(a, n);
}

```

```

89   printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
91
92     for (p = a; p - a < n; p++)
93       printf("%d ", *p);
94     printf("\n");
95
96     exit(EXIT_SUCCESS);
97 }
```

### 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; 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     return min;
42 }
44 /* Funkcija izracunava maksimalni element niza */
45 double max(double *a, int n)
46 {
47     /* Na pocetku, maksimalni element je prvi element */
48     double max = *a;
49
50     /* Ispituje se da li se medju ostalim elementima niza nalazi
51      maksimalni */
52     for (a++, n--; n > 0; a++, n--)
53         if (*a > max)
54             max = *a;
55
56     return max;
57 }
58
59
60 int main()
61 {
62     double a[MAX];
63     int n, i;
64
65     printf("Unesite dimenziju niza: ");
66     scanf("%d", &n);
67
68     /* Proverava se da li je doslo do prekoracenja ogranicenja
69      dimenzije */
70     if (n <= 0 || n > MAX) {
71         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72         exit(EXIT_FAILURE);
73     }
74
75     printf("Unesite elemente niza:\n");
76     for (i = 0; i < n; i++)
77         scanf("%lf", a + i);
78
79     /* Vrsi se testiranje definisanih funkcija */
80     printf("Zbir elemenata niza je %5.3f.\n", zbir(a, n));
81     printf("Proizvod elemenata niza je %5.3f.\n", proizvod(a, n));
82     printf("Minimalni element niza je %5.3f.\n", min(a, n));
83     printf("Maksimalni element niza je %5.3f.\n", max(a, n));
84
85     exit(EXIT_SUCCESS);
86 }
```

**Rešenje 2.3**

```

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

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;

13     while (prvi < poslednji) {

15         /* Povecava se vrednost elementa na koji pokazuje pokazivac prvi
16         */
17         (*prvi)++;

19         /* Pokazivac prvi se pomera na sledeci element */
20         prvi++;

21         /* Smanjuje se vrednost elementa na koji pokazuje pokazivac
22             poslednji */
23         (*poslednji)--;

25         /* Pokazivac poslednji se pomera na prethodni element */
26         poslednji--;
27     }

29     *****
30     Drugi nacin:
31     while (prvi < poslednji) {
32         (*prvi)++;
33         (*poslednji)--;
34     }
35     *****
36 }
37
38 int main()
39 {
40     int a[MAX];
41     int n;
42     int *p;
43
44     printf("Unesite dimenziju niza: ");
45     scanf("%d", &n);

46     /* Proverava se da li je doslo do prekoracenja ogranicenja
47         dimenziije */
48     if (n <= 0 || n > MAX) {
49         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
50         exit(EXIT_FAILURE);
51     }

```

```

53     }
54
55     printf("Unesite elemente niza:\n");
56     for (p = a; p - a < n; p++)
57         scanf("%d", p);
58
59     povecaj_smanji(a, n);
60
61     printf("Transformisan niz je:\n");
62     for (p = a; p - a < n; p++)
63         printf("%d ", *p);
64     printf("\n");
65
66     exit(EXIT_SUCCESS);
}

```

### Rešenje 2.4

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     int i;
6     char tip_ispisa;
7
8     printf("Broj argumenata komandne linije je %d.\n", argc);
9
10    printf("Kako zelite da ispisete argumente, koriscenjem"
11          " indeksne ili pokazivacke sintakse (I ili P)? ");
12    scanf("%c", &tip_ispisa);
13
14    printf("Argumenti komandne linije su:\n");
15    if (tip_ispisa == 'I') {
16        /* Ispisuju se argumenti komandne linije koriscenjem indeksne
17           sintakse */
18        for (i = 0; i < argc; i++)
19            printf("%d %s\n", i, argv[i]);
20    } else if (tip_ispisa == 'P') {
21        /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
22           sintakse */
23        i = argc;
24        for (; argc > 0; argc--)
25            printf("%d %s\n", i - argc, *argv++);
26
27        /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje na
28           polje u memoriji koje se nalazi iza poslednjeg argumenta
29           komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
30           broja argumenta komandne linije to sada moze ponovo da se
31           postavi "argv" da pokazuje na nulti argument komandne linije
32        */
33        argv = argv - i;
34
35    }
36
37    return 0;
38 }

```

```

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

**Rešenje 2.5**

```

1 #include <stdio.h>
2 #include <string.h>
3 #define MAX 100
4
5 /* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
6    cita 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
20     /* Nulti argument komandne linije je ime izvrsnog programa */
21     for (i = 1; i < argc; i++)
22         if (palindrom(*(argv + i)))
23             n++;
24
25     printf
26         ("Broj argumenata komandne linije koji su palindromi je %d.\n",
27          n);
28     return 0;
29 }
```

## Rešenje 2.6

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX_KARAKTERA 100
5
6 /* Implementacija funkcije strlen() iz standardne biblioteke */
7 int duzina(char *s)
8 {
9     int i;
10    for (i = 0; *(s + i); i++);
11    return i;
12 }
13
14 int main(int argc, char **argv)
15 {
16     char rec[MAX_KARAKTERA+1];
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     {
45         if (duzina(rec) == n)
46             br++;
47     }
48
49     printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);
50
51     /* Zatvara se datoteka */
52     fclose(in);

```

```

50     exit(EXIT_SUCCESS);
51 }

```

### 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    sufiks niske zadate prvi argumentom funkcije */

```

```

46     prefiks niske zadate prvi argumentom funkcije */
47 int prefiks_niske(char *niska, char *prefiks)
48 {
49     int i;
50     int duzina_prefiksa = duzina_niske(prefiks);
51     int duzina_niske_pom = duzina_niske(niska);
52     if (duzina_prefiksa <= duzina_niske_pom) {
53         for (i = 0; i < duzina_prefiksa; i++)
54             if (*(prefiks + i) != *(niska + i))
55                 return 0;
56     return 1;
57 } else
58     return 0;
59 }
60
61 int main(int argc, char **argv)
62 {
63     /* Ukoliko korisnik nije uneo trazene argumete, prijavljuje se
64      greska */
65     if (argc < 4) {
66         printf("Greska: ");
67         printf("Nedovoljan broj argumenata komandne linije.\n");
68         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
69                argv[0]);
70         exit(EXIT_FAILURE);
71     }
72
73     FILE *in;
74     int br = 0;
75     char rec[MAX_KARAKTERA+1];
76
77     in = fopen(*(argv + 1), "r");
78     if (in == NULL) {
79         fprintf(stderr, "Greska: ");
80         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
81         exit(EXIT_FAILURE);
82     }
83
84     /* Provera se opcija kojom je pozvan program a zatim se ucitavaju
85      reci iz datoteke i broji se koliko njih zadovoljava trazeni
86      uslov */
87     if (!(poredjenje_niski(*(argv + 3), "-s"))) {
88         while (fscanf(in, "%s", rec) != EOF)
89             br += sufiks_niske(rec, *(argv + 2));
90         printf("Broj reci koje se zavrsavaju na %s je %d.\n", *(argv + 2),
91               br);
92     } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
93         while (fscanf(in, "%s", rec) != EOF)
94             br += prefiks_niske(rec, *(argv + 2));
95         printf("Broj reci koje pocinju na %s je %d.\n", *(argv + 2), br);
96     }

```

```

98     fclose(in);
100    exit(EXIT_SUCCESS);
}

```

### Rešenje 2.8

```

1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 #define MAX 100
6
7 /* Funkcija izracunava trag matrice */
8 int trag(int M[][][MAX], int n)
9 {
10     int trag = 0, i;
11     for (i = 0; i < n; i++)
12         trag += M[i][i];
13     return trag;
14 }
15
16 /* Funkcija izracunava euklidsku normu matrice */
17 double euklidska_norma(int M[][][MAX], int n)
18 {
19     double norma = 0.0;
20     int i, j;
21
22     for (i = 0; i < n; i++)
23         for (j = 0; j < n; j++)
24             norma += M[i][j] * M[i][j];
25
26     return sqrt(norma);
27 }
28
29 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
30 int gornja_vandijagonalna_norma(int M[][][MAX], int n)
31 {
32     int norma = 0;
33     int i, j;
34
35     for (i = 0; i < n; i++) {
36         for (j = i + 1; j < n; j++)
37             norma += abs(M[i][j]);
38     }
39
40     return norma;
41 }
42
43 int main()

```

```

45    {
46        int A[MAX][MAX];
47        int i, j, n;
48
49        printf("Unesite broj vrsta matrice: ");
50        scanf("%d", &n);
51
52        /* Provera prekoracenja dimenzije matrice */
53        if (n > MAX || n <= 0) {
54            fprintf(stderr, "Greska: neodgovarajuca dimenzija matrice.\n");
55            exit(EXIT_FAILURE);
56        }
57
58        printf("Unesite elemente matrice, vrstu po vrstu:\n ");
59        for (i = 0; i < n; i++)
60            for (j = 0; j < n; j++)
61                scanf("%d", &A[i][j]);
62
63        /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
64        for (i = 0; i < n; i++) {
65            /* Ispis elemenata i-te vrste */
66            for (j = 0; j < n; j++)
67                printf("%d ", A[i][j]);
68            printf("\n");
69        }
70
71        /**************************************************************************
72         Ispisuju se elemenati matrice koriscenjem pokazivacke sintakse.
73         Kod ovako definisane matrice, elementi su uzastopno smesteni u
74         memoriju, kao na traci. To znaci da su svi elementi prve vrste
75         redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
76         prve vrste smesten je prvi element druge vrste za kojim sledi
77         svi elementi te vrste i tako dalje redom.
78
79        for( i = 0; i < n ; i++) {
80            for ( j=0 ; j<n ; j++)
81                printf("%d ", *(*(A+i)+j));
82            printf("\n");
83        }
84        /*************************************************************************/
85
86        /* Ispisuje se rezultat na standardni izlaz */
87        int tr = trag(A, n);
88        printf("Trag matrice je %d.\n", tr);
89
90        printf("Euklidska norma matrice je %.2f.\n", euklidska_norma(A, n))
91        ;
92        printf("Vandijagonalna norma matrice je = %d.\n",
93               gornja_vandijagonalna_norma(A, n));
94
95        exit(EXIT_SUCCESS);
96    }

```

## Rešenje 2.9

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
7 standardnog ulaza */
8 void ucitaj_matricu(int m[][][MAX], int n)
9 {
10     int i, j;
11
12     for (i = 0; i < n; i++)
13         for (j = 0; j < n; j++)
14             scanf("%d", &m[i][j]);
15 }
16
17 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
18 standardni izlaz */
19 void ispisi_matricu(int m[][][MAX], int n)
20 {
21     int i, j;
22
23     for (i = 0; i < n; i++) {
24         for (j = 0; j < n; j++)
25             printf("%d ", m[i][j]);
26         printf("\n");
27     }
28 }
29
30 /* Funkcija proverava da li su zadate kvadratne matrice a i b
31 dimenzije n jednake */
32 int jednake_matrice(int a[][][MAX], int b[][][MAX], int n)
33 {
34     int i, j;
35
36     for (i = 0; i < n; i++)
37         for (j = 0; j < n; j++)
38             if (a[i][j] != b[i][j])
39                 return 0;
40
41     /* Prosla je provjera jednakosti za sve parove elemenata koji su na
42     istim pozicijama. To znači da su matrice jednake */
43     return 1;
44 }
45
46 /* Funkcija izracunava zbir dve kvadratne matice */
47 void saberi(int a[][][MAX], int b[][][MAX], int c[][][MAX], int n)
48 {
49     int i, j;

```

```

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 zbir i proizvoda */
76   int zbir[MAX][MAX], proizvod[MAX][MAX];
77
78   /* Dimenzija matrica */
79   int n;
80
81   printf("Unesite broj vrsta 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_maticu(a, n);
93   printf("Unesite elemente druge matrice, vrstu po vrstu:\n");
94   ucitaj_maticu(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)
102    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     exit(EXIT_SUCCESS);
113 }
```

### Rešenje 2.10

```

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

4 #define MAX 64

6 /* Funkcija proverava da li je relacija refleksivna. Relacija je
8   refleksivna ako je svaki element u relaciji sa sobom, odnosno ako
se u matrici relacije na glavnoj dijagonali nalaze jedinice */
int refleksivnost(int m[][MAX], int n)
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 /* Funkcija određuje refleksivno zatvorene zadate relacije. Ono je
22   određeno matricom koja sadrži sve elemente polazne matrice
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
42   relaciji sa elementom "j", onda je i element "j" u relaciji sa
44   elementom "i". Ovakve matrice su simetricne u odnosu na glavnu
46   dijagonalu */
48   int simetricnost(int m[][][MAX], int n)
49   {
50     int i, j;
52
54     /* Obilaze se elementi ispod glavne dijagonale matrice i uporedjuju
56      se sa njima simetricnim elementima */
58     for (i = 0; i < n; i++)
59       for (j = 0; j < i; j++)
60         if (m[i][j] != m[j][i])
61           return 0;
63
64     return 1;
65   }
66
68  /* Funkcija odredjuje simetricno zatvorene zadate relacije. Ono je
70   odredjeno matricom koja sadrzi sve elemente polazne matrice
72   dopunjene tako da matrica postane simetricna u odnosu na glavnu
74   dijagonalu */
76  void sim_zatvorene(int m[][][MAX], int n, int zatvorene[][][MAX])
77  {
78    int i, j;
79
80    for (i = 0; i < n; i++)
81      for (j = 0; j < n; j++)
82        zatvorene[i][j] = m[i][j];
83
84    for (i = 0; i < n; i++)
85      for (j = 0; j < n; j++)
86        if (zatvorene[i][j] == 1)
87          zatvorene[j][i] = 1;
88  }
89
90  /* Funkcija proverava da li je relacija tranzitivna. Relacija je
92   tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
94   relaciji sa elementom "j" i element "j" u relaciji sa elementom
96   "k", onda je i element "i" u relaciji sa elementom "k" */
98  int tranzitivnost(int m[][][MAX], int n)
99  {
100    int i, j, k;
101
102    for (i = 0; i < n; i++)
103      for (j = 0; j < n; j++)
104        /* Ispituje se da li postoji element koji narusava *
105        tranzitivnost */
106        for (k = 0; k < n; k++)
107          if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
108            return 0;
109
110    return 1;
111  }

```

```

90         return 0;
92     return 1;
94 }
95
96 /* Funkcija određuje refleksivno-tranzitivno zatvorene zadate
   relacije koriscenjem Varsalovog algoritma */
97 void ref_tran_zatvorene(int m[][][MAX], int n, int zatvorene[][][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            zatvorene[i][j] = m[i][j];
105
106    /* Određuje se reflektivno zatvorene matrice */
107    for (i = 0; i < n; i++)
108        zatvorene[i][i] = 1;
109
110    /* Primenom Varsalovog algoritma određuje se tranzitivno
   zatvorene matrice */
111    for (k = 0; k < n; k++)
112        for (i = 0; i < n; i++)
113            for (j = 0; j < n; j++)
114                if ((zatvorene[i][k] == 1) && (zatvorene[k][j] == 1)
115                    && (zatvorene[i][j] == 0))
116                    zatvorene[i][j] = 1;
117    }
118
119    /* Funkcija ispisuje elemente matrice */
120 void pisi_matricu(int m[][][MAX], int n)
121 {
122     int i, j;
123
124     for (i = 0; i < n; i++) {
125         for (j = 0; j < n; j++)
126             printf("%d ", m[i][j]);
127         printf("\n");
128     }
129 }
130
131 int main(int argc, char *argv[])
132 {
133     FILE *ulaz;
134     int m[MAX][MAX];
135     int pomocna[MAX][MAX];
136     int n, i, j;
137
138     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
   */
139
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 zatvorene relacije:\n");
183     ref_zatvorenje(m, n, pomocna);
184     pisi_matricu(pomocna, n);
185
186     printf("Simetricno zatvorene relacije:\n");
187     sim_zatvorenje(m, n, pomocna);
188     pisi_matricu(pomocna, n);
189
190     printf("Refleksivno-tranzitivno zatvorene relacije:\n");
191     ref_tran_zatvorenje(m, n, pomocna);
192     pisi_matricu(pomocna, n);

```

```

194     /* Zatvara se datoteka */
195     fclose(ulaz);
196
197     exit(EXIT_SUCCESS);
198 }
```

### Rešenje 2.11

```

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

```

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

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

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

61     return broj_negativnih;
62 }

63 int main(int argc, char *argv[])
64 {
65     int m[MAX][MAX];
66     int n;
67     int i, j;

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

75     /* Ucitava se broj vrsta matrice */
76     n = atoi(argv[1]);

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

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

87     printf("Najveci element sporedne dijagonale je %.1f",
88 
```

```

96         max_sporedna_dijagonala(m, n));

98     printf("Indeks kolone sa najmanjim elementom je %d.\n",
100        indeks_min(m, n));
102
104     printf("Indeks vrste sa najvecim elementom je %d.\n",
106        indeks_max(m, n));
108
109     printf("Broj negativnih elemenata matrice je %d.\n",
110        broj_negativnih(m, n));
111
112     exit(EXIT_SUCCESS);
113 }
```

### Rešenje 2.12

```

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

```

35    {
36        int i, j, k;
37        int proizvod;
38
39        /* Ispituje se uslov normiranosti */
40        for (i = 0; i < n; i++) {
41            proizvod = 0;
42
43            for (j = 0; j < n; j++)
44                proizvod += m[i][j] * m[i][j];
45
46            if (proizvod != 1)
47                return 0;
48        }
49
50        /* Ispituje se uslov ortogonalnosti */
51        for (i = 0; i < n - 1; i++) {
52            for (j = i + 1; j < n; j++) {
53
54                proizvod = 0;
55
56                for (k = 0; k < n; k++)
57                    proizvod += m[i][k] * m[j][k];
58
59                if (proizvod != 0)
60                    return 0;
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 broj vrsta 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");

```

```

87     exit(EXIT_SUCCESS);
}

```

### Rešenje 2.13

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX_V 10
5 #define MAX_K 10
6
7 /* Funkcija proverava da li su ispisani svi elementi iz matrice,
   odnosno da li se narušio prirodan poređak medju granicama */
8 int krajIspisa(int top, int bottom, int left, int right)
9 {
10    return !(top <= bottom && left <= right);
11}
12
13 /* Funkcija spiralno ispisuje elemente matrice */
14 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
15 {
16    int i, j, top, bottom, left, right;
17
18    top = left = 0;
19    bottom = n - 1;
20    right = m - 1;
21
22    while (!krajIspisa(top, bottom, left, right)) {
23
24        for (j = left; j <= right; j++)
25            printf("%d ", a[top][j]);
26
27        /* Spusta se prvi red */
28        top++;
29
30        if (krajIspisa(top, bottom, left, right))
31            break;
32
33        for (i = top; i <= bottom; i++)
34            printf("%d ", a[i][right]);
35
36        /* Pomera se desna kolona za naredni krug ispisa blize levom
           kraju */
37        right--;
38
39        if (krajIspisa(top, bottom, left, right))
40            break;
41
42        /* Ispisuje se donja vrsta */
43        for (j = right; j >= left; j--)
44

```

```

47     printf("%d ", a[bottom][j]);
48
49     /* Podize se donja vrsta za naredni krug ispisa */
50     bottom--;
51
52     if (krajIspisa(top, bottom, left, right))
53         break;
54
55     /* Ispisuje se prva kolona */
56     for (i = bottom; i >= top; i--)
57         printf("%d ", a[i][left]);
58
59     /* Priprema se leva kolona za naredni krug ispisa */
60     left++;
61 }
62 putchar('\n');
63
64 /* Funkcija ucitava matricu */
65 void ucitaj_matricu(int a[][MAX_K], int n, int m)
66 {
67     int i, j;
68
69     for (i = 0; i < n; i++)
70         for (j = 0; j < m; j++)
71             scanf("%d", &a[i][j]);
72 }
73
74 int main()
75 {
76     int a[MAX_V][MAX_K];
77     int m, n;
78
79     printf("Unesite broj vrsta i broj kolona matrice:\n");
80     scanf("%d %d", &n, &m);
81
82     if (n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
83         fprintf(stderr, "Greska: neodgovarajuce dimenzije ");
84         fprintf(stderr, "matrice.\n");
85         exit(EXIT_FAILURE);
86     }
87
88     printf("Unesite elemente matrice, vrstu po vrstu:\n");
89     ucitaj_matricu(a, n, m);
90
91     printf("Spiralno ispisana matrica: ");
92     ispisiti_matricu_spiralno(a, n, m);
93
94     exit(EXIT_SUCCESS);
95 }

```

## Rešenje 2.15

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int *p = NULL;
7     int i, n;
8
9     printf("Unesite dimenziju niza: ");
10    scanf("%d", &n);
11
12    /* Alocira se prostor za n celih brojeva */
13    if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
14        fprintf(stderr, "malloc(): ");
15        fprintf(stderr, "greska pri alokaciji memorije.\n");
16        exit(EXIT_FAILURE);
17    }
18
19    printf("Unesite elemente niza: ");
20    for (i = 0; i < n; i++)
21        scanf("%d", &p[i]);
22
23    printf("Niz u obrnutom poretku je: ");
24    for (i = n - 1; i >= 0; i--)
25        printf("%d ", p[i]);
26    printf("\n");
27
28    /* Oslobadja se prostor rezervisan funkcijom malloc() */
29    free(p);
30
31    exit(EXIT_SUCCESS);
32 }
```

## Rešenje 2.16

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define KORAK 10
4
5 int main()
6 {
7     /* Adresa prvog alociranog bajta */
8     int *a = NULL;
9
10    /* Velicina alocirane memorije */
11    int alocirano;
12
13    /* Broj elemenata niza */
```

```

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

```

```

66         exit(EXIT_FAILURE);
67     }
68 }
69
70     a[n++] = x;
71
72     scanf("%d", &x);
73 }
74
75
76     printf("Niz u obrnutom poretku je: ");
77     for (n--; n >= 0; n--)
78         printf("%d ", a[n]);
79     printf("\n");
80
81 /* Oslobadja se dinamicki alocirana memorija */
82 free(a);
83
84 exit(EXIT_SUCCESS);
85 }
```

### Rešenje 2.17

```

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

```

28 int main()
29 {
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     exit(EXIT_SUCCESS);
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 niz vrsta matrice realnih brojeva */
10    double **A = NULL;
11
12    /* Broj vrsta i broj kolona */
13    int n = 0, m = 0;
14
15    /* Trag matice */
16    double trag = 0;
17
18    printf("Unesite broj vrsta i broj kolona matrice:\n ");
19    scanf("%d%d", &n, &m);
20
21    /* Činjam da se alocira prostor za niz vrsta matrice */
22    A = (double **)malloc(sizeof(double *) * n);
23
24    /* Provera se da li je doslo do greske pri alokaciji */
25    if (A == NULL) {
26        fprintf(stderr, "malloc(): ");
27        fprintf(stderr, "greska pri alokaciji memorije.\n");
28        exit(EXIT_FAILURE);
29 }
```

```

29 }
30
31 /* Dinamicki se alocira prostor za elemente u vrstama */
32 for (i = 0; i < n; i++) {
33     A[i] = (double **)malloc(sizeof(double) * m);
34
35     /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
36      potrebno je oslobiti svih i-1 prethodno alociranih vrsta, i
37      alociran niz pokazivaca */
38     if (A[i] == NULL) {
39         for (j = 0; j < i; j++)
40             free(A[j]);
41         free(A);
42         exit(EXIT_FAILURE);
43     }
44 }
45
46 printf("Unesite elemente matrice, vrstu po vrstu:\n");
47 for (i = 0; i < n; i++)
48     for (j = 0; j < m; j++)
49         scanf("%lf", &A[i][j]);
50
51 /* Izracunava se trag matrice, odnosno suma elemenata na glavnoj
52   dijagonali */
53 trag = 0.0;
54
55 for (i = 0; i < n; i++)
56     trag += A[i][i];
57
58 printf("Trag unete matrice je %.2f.\n", trag);
59
60 /* Oslobadja se prostor rezervisan za svaku vrstu */
61 for (j = 0; j < n; j++)
62     free(A[j]);
63
64 /* Oslobadja se memorija za niz pokazivaca na vrste */
65 free(A);
66
67 exit(EXIT_SUCCESS);
68 }

```

### Rešenje 2.19

Datoteka 2.1: *matrica.h*

```

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

```

```

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

11 /* Funkcija dealocira memoriju za matricu sa n vrsta */
12 int **dealociraj_matricu(int **matrica, int n);

13 /* Funkcija ucitava vec alociranu matricu dimenzija n x m sa
   standardnog ulaza */
14 void ucitaj_matricu(int **matrica, int n, int m);

17 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n sa
   standardnog ulaza */
18 void ucitaj_kvadratnu_matricu(int **matrica, int n);

21 /* Funkcija ispisuje matricu dimenzija n x m na standardnom izlazu */
22 void ispisi_matricu(int **matrica, int n, int m);

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

29 /* Funkcija ucitava vec alociranu matricu dimenzija n x m iz datoteke
   f */
30 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
    ;

33 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n iz
   datoteke f */
34 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
                                             FILE * f);

37 /* Funkcija upisuje matricu dimenzija n x m u datoteku f */
38 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);

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

45 #endif

```

Datoteka 2.2: *matrica.c*

```

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

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

```

```

1   int i, j;
2
3   /* Alocira se prostor za niz vrsta matrice */
4   matrica = (int **) malloc(n * sizeof(int *));
5   /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije ce
6    biti NULL, sto mora biti provereno u main funkciji */
7   if (matrica == NULL)
8       return NULL;
9
10  /* Alocira se prostor za svaku vrstu matrice */
11  for (i = 0; i < n; i++) {
12      matrica[i] = (int *) malloc(m * sizeof(int));
13      /* Ako alokacija nije prosla uspesno, oslobadaju se svi
14       prethodno alocirani resursi, i povratna vrednost je NULL */
15      if (matrica[i] == NULL) {
16          for (j = 0; j < i; j++)
17              free(matrica[j]);
18          free(matrica);
19          return NULL;
20      }
21  }
22  return matrica;
23}
24
25 int **alociraj_kvadratnu_matricu(int n)
26{
27    /* Alociranje matrice dimenzije n x n */
28    return alociraj_matricu(n, n);
29}
30
31 int **dealociraj_matricu(int **matrica, int n)
32{
33    int i;
34    /* Oslobadja se prostor rezervisan za svaku vrstu */
35    for (i = 0; i < n; i++)
36        free(matrica[i]);
37    /* Oslobadja se memorija za niz pokazivaca na vrste */
38    free(matrica);
39
40    /* Matrica postaje prazna, tj. nealocirana */
41    return NULL;
42}
43
44 void ucitaj_matricu(int **matrica, int n, int m)
45{
46    int i, j;
47    /* Elementi matrice se ucitacaju po vrstama */
48    for (i = 0; i < n; i++)
49        for (j = 0; j < m; j++)
50            scanf("%d", &matrica[i][j]);
51}
52
53
54
55
56
57
58
59

```

```

61 void ucitaj_kvadratnu_matricu(int **matrica, int n)
62 {
63     /* Ucitavanje matrice n x n */
64     ucitaj_matricu(matrica, n, n);
65 }
66
67 void ispisi_matricu(int **matrica, int n, int m)
68 {
69     int i, j;
70     /* Ispis po vrstama */
71     for (i = 0; i < n; i++) {
72         for (j = 0; j < m; j++)
73             printf("%d ", matrica[i][j]);
74         printf("\n");
75     }
76 }
77
78 void ispisi_kvadratnu_matricu(int **matrica, int n)
79 {
80     /* Ispis matrice n x n */
81     ispisi_matricu(matrica, n, n);
82 }
83
84 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
85 {
86     int i, j;
87     /* Elementi matrice se ucitacaju po vrstama */
88     for (i = 0; i < n; i++)
89         for (j = 0; j < m; j++)
90             /* Ako je nemoguce ucitati sledeci element, povratna vrednost
91                funkcije je 1, kao indikator neuspesnog ucitavanja */
92             if (fscanf(f, "%d", &matrica[i][j]) != 1)
93                 return 1;
94
95     /* Uspesno ucitana matrica */
96     return 0;
97 }
98
99 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
100                                            FILE * f)
101 {
102     /* Ucitavanje matrice n x n iz datoteke */
103     return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
104 }
105
106 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
107 {
108     int i, j;
109     /* Ispis po vrstama */
110     for (i = 0; i < n; i++) {
111         for (j = 0; j < m; j++)
112             /* Ako je nemoguce ispisati sledeci element, povratna vrednost
113                funkcije je 1, kao indikator neuspesnog ucitavanja */
114             if (fprintf(f, "%d ", matrica[i][j]) != 1)
115                 return 1;
116
117     }
118 }

```

```

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

Datoteka 2.3: *main\_a.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 int main()
6 {
7     int **matrica = NULL;
8     int n, m;
9     FILE *f;
10
11     /* Ucitavanje dimenzije matrice */
12     printf("Unesi broj vrsta matrice: ");
13     scanf("%d", &n);
14     printf("Unesi broj kolona matrice: ");
15     scanf("%d", &m);
16
17     /* Provera dimenzija matrice */
18     if (n <= 0 || m <= 0) {
19         fprintf(stderr, "Neodgovarajce dimenzije matrice\n");
20         exit(EXIT_FAILURE);
21     }
22
23     /* Alokacija matrice i provera alokacije */
24     matrica = alociraj_matricu(n, m);
25     if (matrica == NULL) {
26         fprintf(stderr, "Neuspesna alokacija matrice\n");
27         exit(EXIT_FAILURE);
28     }
29
30     /* Ucitavanje matrice sa standardnog ulaza */
31     printf("Unesi elemente matrice po vrstama:\n");
32     ucitaj_matricu(matrica, n, m);
```

```

34  /* Otvaranje fajla za upis matrice */
35  if ((f = fopen("matrica.txt", "w")) == NULL) {
36      fprintf(stderr, "fopen() error\n");
37      matrica = dealociraj_matricu(matrica, n);
38      exit(EXIT_FAILURE);
39  }
40
41  /* Upis matrice u fajl */
42  if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
43      fprintf(stderr, "Neuspesno upisivanje matrice u datoteku\n");
44      matrica = dealociraj_matricu(matrica, n);
45      exit(EXIT_FAILURE);
46  }
47
48  /* Zatvaranje fajla */
49  fclose(f);
50
51  /* Dealokacija matrice */
52  matrica = dealociraj_matricu(matrica, n);
53
54  exit(EXIT_SUCCESS);
55 }
```

Datoteka 2.4: *main\_b.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 int main(int argc, char **argv)
6 {
7     int **matrica = NULL;
8     int n;
9     FILE *f;
10
11    /* Provera argumenata komandne linije */
12    if (argc != 2) {
13        fprintf(stderr, "Koriscenje programa: %s datoteka\n", argv[0]);
14        exit(EXIT_FAILURE);
15    }
16
17    /* Otvaranje fajla za citanje */
18    if ((f = fopen(argv[1], "r")) == NULL) {
19        fprintf(stderr, "fopen() error\n");
20        exit(EXIT_FAILURE);
21    }
22
23    /* Ucitavanje dimenzije matrice */
24    if (fscanf(f, "%d", &n) != 1) {
25        fprintf(stderr, "Neispravan pocetak fajla\n");
```

```

    exit(EXIT_FAILURE);
}

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

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

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

/* Zatvaranje fajla */
fclose(f);

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

/* Dealokacija matrice */
matrica = dealociraj_matricu(matrica, n);
exit(EXIT_SUCCESS);
}

```

### Rešenje 2.20

```

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

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

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

```

```

15    }
16}
17
18 int main()
19 {
20     int m, n;
21     int **matrica = NULL;
22
23     printf("Unesite broj vrsta i broj kolona matrice:\n ");
24     scanf("%d %d", &n, &m);
25
26     /* Alocira se matrica */
27     matrica = alociraj_matricu(n, m);
28     /* Provera alokacije */
29     if (matrica == NULL) {
30         fprintf(stderr, "Neuspesna alokacija matrice\n");
31         exit(EXIT_FAILURE);
32     }
33
34     printf("Unesite elemente matrice, vrstu po vrstu:\n ");
35     ucitaj_matricu(matrica, n, m);
36
37     printf("Elementi ispod glavne dijagonale matrice:\n ");
38     ispisi_elemente_ispod_dijagonale(matrica, n, m);
39
40     /* Oslobođanje memorije */
41     dealociraj_matricu(matrica, n);
42
43     exit(EXIT_SUCCESS);
44 }
```

### Rešenje 2.22

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 /* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni(float **a, int n)
7 {
8     int i, j;
9
10    for (i = 0; i < n; i++)
11        for (j = 0; j < n; j++)
12            if (i < j)
13                a[i][j] /= 2;
14            else if (i > j)
15                a[i][j] *= 2;
16 }
17
18 /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
```

```

20     sporedne dijagonale. Element se nalazi ispod sporedne dijagonale
21     ukoliko je zbir indeksa vrste i indeksa kolone elementa veci od
22     n-1 */
23 float zbir_ispod_sporedne_dijagonale(float **m, int n)
24 {
25     int i, j;
26     float zbir = 0;
27
28     for (i = 0; i < n; i++)
29         for (j = n-i; j < n; j++)
30             if (i + j > n - 1)
31                 zbir += fabs(m[i][j]);
32
33     return zbir;
34 }
35
36 /* Funkcija ucitava elemente kvadratne matrice dimenzije n iz zadate
37    datoteke */
38 void ucitaj_matricu(FILE * ulaz, float **m, int n)
39 {
40     int i, j;
41
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;
51
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 alocira memoriju za kvadratnu matricu dimenzije n */
59 float **alociraj_memoriju(int n)
60 {
61     int i, j;
62     float **m;
63
64     m = (float **) malloc(n * sizeof(float *));
65     if (m == NULL) {
66         fprintf(stderr, "malloc(): Neuspela alokacija\n");
67         exit(EXIT_FAILURE);
68     }
69
70 }

```

```

72     for (i = 0; i < n; i++) {
73         m[i] = (float *) malloc(n * sizeof(float));
74
75         if (m[i] == NULL) {
76             printf("malloc(): neuspela alokacija memorije!\n");
77             for (j = 0; j < i; j++)
78                 free(m[i]);
79             free(m);
80             exit(EXIT_FAILURE);
81         }
82     }
83     return m;
84 }
85 /* Funckija oslobođenja memoriju zauzetu kvadratnom matricom dimenzije
86  n */
87 void osloboodi_memoriju(float **m, int n)
88 {
89     int i;
90
91     for (i = 0; i < n; i++)
92         free(m[i]);
93     free(m);
94 }
95
96 int main(int argc, char *argv[])
97 {
98     FILE *ulaz;
99     float **a;
100    int n;
101
102    /* Ako korisnik nije uneo tražene argumente, prijavljuje se greska
103     */
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 čitanje */
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 */

```

```

122 a = alociraj_memoriju(n);
124 /* Ucitavaju se elementi matrice */
126 ucitaj_matricu(ulaz, a, n);
128 float zbir = zbir_ispod_sporedne_dijagonale(a, n);
130 /* Poziva se funkcija za transformaciju matrice */
132 izmeni(a, n);
134
136 /* Ispisuje se rezultat */
137 printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
138 printf("je %.2f.\n", zbir);
140
142 /* Transformisana matrica je:\n*/
143 ispisi_matricu(a, n);
144
146 /* Oslobadja se memorija */
147 oslobodi_memoriju(a, n);
148
150 /* Zatvara se datoteka */
151 fclose(ulaz);
153
155 exit(EXIT_SUCCESS);
156 }

```

### Rešenje 2.27

```

1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <math.h>
5 #include <string.h>
6
7 /* Funkcija tabela() prihvata granice intervala a i b, broj
8   ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
9   funkciju koja prihvata double argument, i vraca double vrednost.
10  Za tako datu funkciju ispisuju se njene vrednosti u intervalu
11  [a,b] u n ekvidistantnih tacaka intervala */
12 void tabela(double a, double b, int n, double (*fp) (double))
13 {
14     int i;
15     double x;
16
17     printf("-----\n");
18     for (i = 0; i < n; i++) {
19         x = a + i * (b - a) / (n - 1);
20         printf("| %8.5f | %8.5f |\n", x, (*fp)(x));
21     }
22     printf("-----\n");
23 }

```

```

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

```

```
75     fp = &sqrt;
77 else if (strcmp(ime_fje, "floor") == 0)
    fp = &floor;
79 else if (strcmp(ime_fje, "ceil") == 0)
    fp = &ceil;
81 else if (strcmp(ime_fje, "sqr") == 0)
    fp = &sqr;
83 else {
84     printf("Program jos uvek ne podrzava trazenu funkciju!\n");
85     exit(EXIT_SUCCESS);
86 }
87
88 printf("Unesite krajeve intervala:\n");
89 scanf("%lf %lf", &a, &b);
90
91 printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
92 printf("(uključujući krajeve intervala)?\n");
93 scanf("%d", &n);
94
95 /* Mreza mora da uključuje bar krajeve intervala, tako da se mora
96 uneti broj veci od 2 */
97 if (n < 2) {
98     fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
99     exit(EXIT_FAILURE);
100 }
101
102 /* Ispisuje se ime funkcije */
103 printf("      x %10s(x)\n", ime_fje);
104
105 /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
106   komandne linije */
107 tabela(a, b, n, fp);
108
109 exit(EXIT_SUCCESS);
110 }
```

# Glava 3

## Algoritmi pretrage i sortiranja

### 3.1 Algoritmi pretrage

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

### 3.1 Algoritmi pretrage

*Test 1*

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

*Test 2*

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

[Rešenje 3.1]

**Zadatak 3.2** Napisati rekurzivne funkcije algoritama linearne, binarne i interpolacione pretrage i program koji ih testira za brojeve koji se unose sa standardnog ulaza. Linearnu pretragu implementirati na dva načina, svodenjem pretrage na prefiks i na sufiks niza. 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.

### **3.1 Algoritmi pretrage**

---

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 složenoosti. 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 datateci veći od 0 i ne veći od 1024.

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

[Rešenje 3.5]

**Zadatak 3.6** Napisati funkciju koja određuje nulu funkcije  $\cos(x)$  na intervalu  $[0, 2]$  metodom polovljenja intervala. Algoritam se završava kada se vrednost kosinusne funkcije razlikuje za najviše 0.001 od nule. UPUTSTVO: *Korisiti algoritam analogan algoritmu binarne pretrage, metod polovljenja intervala.* NAPOMENA: *Ovaj metod se može primeniti na funkciju  $\cos(x)$  na intervalu  $[0, 2]$  zato što je ona na ovom intervalu neprekidna, i vrednosti funkcije na krajevima intervala su različitog znaka.*

<i>Test 1</i>
<pre>IZLAZ: 1.57031</pre>

[Rešenje 3.6]

**Zadatak 3.7** Napisati funkciju koja metodom polovljenja intervala određuje nulu izabrane funkcije na proizvolnjom intervalu sa tačnošću  $\epsilon$ . Ime funkcije se zadaje kao prvi argument komandne linije, a interval i tačnost se unose sa standardnog ulaza. Pretpostaviti da je izabrana funkcija na tom intervalu neprekidna. UPUTSTVO: *U okviru algoritma pretrage koristiti pokazivač na odgovarajuću funkciju (na primer, kao u zadatku 2.27).*

<i>Primer 1</i>	<i>Primer 2</i>
<pre>POZIV: ./a.out cos</pre> <pre>INTERAKCIJA SA PROGRAMOM: Unesite krajeve intervala: 0 2 Unesite preciznost: 0.001 1.57031</pre>	<pre>POZIV: ./a.out sin</pre> <pre>INTERAKCIJA SA PROGRAMOM: Unesite krajeve intervala: 1 5 Unesite preciznost: 0.00001 3.1416</pre>

### 3.1 Algoritmi pretrage

*Primer 3*

```
Poziv: ./a.out tan  
INTERAKCIJA SA PROGRAMOM:  
Unesite krajeve intervala: -1.1 1  
Unesite preciznost: 0.00001  
3.8147e-06
```

*Primer 4*

```
Poziv: ./a.out sin  
INTERAKCIJA SA PROGRAMOM:  
Unesite krajeve intervala: 1 3  
Funkcija sin na intervalu [1, 3]  
ne zadovoljava uslove
```

[Rešenje 3.7]

**Zadatak 3.8** 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*

```
ULAZ:  
-151 -44 5 12 13 15  
IZLAZ:  
2
```

*Test 2*

```
ULAZ:  
-100 -15 -11 -8 -7 -5  
IZLAZ:  
-1
```

*Test 3*

```
ULAZ:  
-100 -15 0 13 55 124  
258 315 516 7000  
IZLAZ:  
3
```

[Rešenje 3.8]

**Zadatak 3.9** 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.

*Test 1*

```
ULAZ:  
151 44 5 -12 -13 -15  
IZLAZ:  
3
```

*Test 2*

```
ULAZ:  
100 55 15 0 -15 -124  
-155 -258 -315 -516  
IZLAZ:  
4
```

*Test 3*

```
ULAZ:  
100 15 11 8 7 5 4 3 2  
IZLAZ:  
-1
```

[Rešenje 3.9]

**Zadatak 3.10** Napisati funkciju koja određuje ceo deo logaritma za osnovu 2 datog neoznačenog celog broja koristeći samo bitske i relacione operatore.

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

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 4	ULAZ: 17	ULAZ: 1031
IZLAZ: 2 2	IZLAZ: 4 4	IZLAZ: 10 10

[Rešenje 3.10]

**\*\* Zadatak 3.11** U prvom kvadrantu dano je  $1 \leq N \leq 10000$  duži svojim koordinatama (duži mogu da se sekut, 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  $[0, 90^\circ]$ .*

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 3</i>
INTERAKCIJA SA PROGRAMOM: Unesi broj tacaka: 2 Unesi koordinate tacaka: 2 0 2 1 1 2 2 2 26.57	INTERAKCIJA SA PROGRAMOM: Unesi broj tacaka: 2 Unesi koordinate tacaka: 1 0 1 1 0 1 1 1 45	INTERAKCIJA SA PROGRAMOM: Unesi broj tacaka: 3 Unesi koordinate tacaka: 1 0 1 1 2 0 2 1 1 2 2 2 26.57

## 3.2 Algoritmi sortiranja

**Zadatak 3.12** Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom

(engl. **selection sort**), sortiranja spajanjem (engl. **merge sort**), brzog sortiranja (engl. **quick sort**), mehurastog sortiranja (engl. **bubble sort**), sortiranja direktnim umetanjem (engl. **insertion sort**) i sortiranja umetanjem sa inkrementom (engl. **shell sort**). Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za sortiranje spajanjem, **-q** za brzo sortiranje, **-b** za mehurasto, **-i** za sortiranje direktnim umetanjem ili **-s** za sortiranje umetanjem sa inkrementom. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati algoritmom sortiranja izborom. Niz koji se sortira generisati neopadajuće ako je prisutna opcija **-r**, nerastuće ako je prisutna opcija **-o** ili potpuno slučajno ako nema nijedne opcije. Vreme meriti programom **time**. Analizirati porast vremena sa porastom dimenzije **n**.

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

[Rešenje 3.12]

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

## 3.2 Algoritmi sortiranja

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 anagr
nisu

Primer 3

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

[Rešenje 3.13]

**Zadatak 3.14** U datom nizu brojeva treba pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, ali neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati razliku pronađena dva broja. UPUTSTVO: *Prvo sortirati niz.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.*

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 3.14]

**Zadatak 3.15** Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza i neće biti duži od 256 i kraći od jednog elemenata. UPUTSTVO: *Prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.*

Test 1

ULAZ:
4 23 5 2 4 6 7 34 6 4 5
IZLAZ:
4

Test 2

ULAZ:
2 4 6 2 6 7 99 1
IZLAZ:
2

Test 3

ULAZ:
123
IZLAZ:
123

[Rešenje 3.15]

**Zadatak 3.16** 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. Prepostaviti da u niz neće biti uneto više od

## **3.2 Algoritmi sortiranja**

---

256 brojeva. UPUTSTVO: *Prvo sortirati niz.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite traženi zbir: 34  
Unesite elemente niza:  
134 4 1 6 30 23  
da
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite traženi zbir: 12  
Unesite elemente niza:  
53 1 43 3 56 13  
ne
```

*Primer 3*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite traženi zbir: 52  
Unesite elemente niza:  
52  
ne
```

[Rešenje 3.16]

**Zadatak 3.17** Napisati funkciju potpisa `int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3, int dim3)` koja prima dva sortirana niza, i na osnovu njih pravi novi sortirani niz koji koji sadrži elemente oba niza. Treća dimenzija predstavlja veličinu niza u koji se smešta rezultat. Ako je ona manja od potrebne dužine, funkcija vraća -1 kao indikator neuspeha, inače vraća 0. Napisati zatim program koji testira ovu funkciju. Nizovi se unose sa standardnog ulaza sve dok se ne unese 0 i može se pretpostaviti da će njihove dimenzije biti manje od 256.

*Primer 1*

```
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
```

*Primer 2*

```
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
```

[Rešenje 3.17]

**Zadatak 3.18** Napisati program koji čita sadržaj dveju datoteka od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima, a u slučaju istog imena po prezimenima, i kreira jedinstven spisak studenata sortiranih takođe po istom kriterijumu. Program dobija nazive datoteka iz komandne linije i jedinstveni spisak upisuje u datoteku `ceo-tok.txt`. Prepostaviti 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          CEO-TOK.TXT
Andrija Petrovic      Aleksandra Cvetic
Anja Ilic              Andrija Petrovic
Ivana Markovic         Anja Ilic
Lazar Micic           Bojan Golubovic
Nenad Brankovic       Dragan Markovic
Sofija Filipovic      Filip Dukic
Uros Milic             Ivana Markovic
Vladimir Savic         Ivana Stankovic
                         Lazar Micic
                         Marija Stankovic
                         Nenad Brankovic
                         Ognjen Peric
                         Sofija Filipovic
                         Vladimir Savic
                         Uros Milic

DRUGI-DEO.TXT
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Marija Stankovic
Ognjen Peric

```

[Rešenje 3.18]

**Zadatak 3.19** 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 3.19]

### **3.2 Algoritmi sortiranja**

---

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

*Test 1*

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

```
IZLAZ:
3
```

*Test 2*

```
BIRACKI-SPISAK.TXT
Milan Milicevic
IZLAZ:
1
```

*Test 3*

```
DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI
IZLAZ ZA GREŠKU:
Neupesno otvaranje
datoteke
biracki-spisak.txt.
```

[Rešenje 3.20]

**Zadatak 3.21** Definisati strukturu koja čuva imena, prezimena i godišta dece. Pretpostaviti da su imena i prezimena niske karaktera koje nisu duže od 30 karaktera. Napisati funkciju koja sortira niz dece po godištu, a decu istog godišta sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci čije se ime zadaje kao prvi argument komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

*Test 1*

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

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

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

*Test 2*

```
|| POZIV: ./a.out in.txt out.txt
|| IN.OUT          OUT.TXT
|| Milijana Maric 2009      Milijana Maric 2009
```

**Zadatak 3.22** Napisati funkciju koja sortira niz niski po broju suglasnika u niski. Ukoliko reči imaju isti broj suglasnika tada sortirati ih po dužini niske 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**. Prepostaviti 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
|| IZLIZ:
||     ana matej milos petar milica nikola andjela ljubica aleksandar
```

[Rešenje [3.22](#)]

**Zadatak 3.23** Napisati program koji simulira rad kase u prodavnici. Kupci prilaze kasi, a prodavac unošenjem bar-koda kupljenog proizvoda dodaje njegovu cenu na ukupan račun. Na kraju, program ispisuje ukupnu vrednost svih proizvoda. Sve artikle, zajedno sa bar-kodovima, prozivodačima 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 proizvodjaca 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 traenim kodom!
Unesite kod artikla [ili 0 za prekid]: 2530
Trazili ste: Napolitanke Bambi 230.00
Unesite kod artikla [ili 0 za prekid]: 0

UKUPNO: 230.00 dinara.

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

Kraj rada kase!
```

[Rešenje 3.23]

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

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	DAT2.TXT
Aleksandra Cvetic 4 6	Studenti sortirani po broju zadataka opadajuće, pa po duzini imena rastuce:
Bojan Golubovic 4 3	Aleksandra Cvetic 4 6
Dragan Markovic 3 5	Uros Milic 2 5
Ivana Stankovic 3 1	Dragan Markovic 3 5
Marija Stankovic 1 3	Andrija Petrovic 2 5
Ognjen Peric 1 2	Nenad Brankovic 2 4
Uros Milic 2 5	Lazar Micic 1 3
Andrija Petrovic 2 5	Bojan Golubovic 4 3
Anja Ilic 3 1	Marija Stankovic 1 3
Lazar Micic 1 3	Ognjen Peric 1 2
Nenad Brankovic 2 4	Anja Ilic 3 1
	Ivana Stankovic 3 1
DAT1.TXT	DAT3.TXT
Studenti sortirani po imenu leksikografski rastuce:	Studenti sortirani po prisustvu opadajuće, pa po broju zadataka, pa po prezimenima leksikografski opadajuće:
Aleksandra Cvetic 4 6	Aleksandra Cvetic 4 6
Andrija Petrovic 2 5	Bojan Golubovic 4 3
Anja Ilic 3 1	Dragan Markovic 3 5
Bojan Golubovic 4 3	Ivana Stankovic 3 1
Dragan Markovic 3 5	Anja Ilic 3 1
Ivana Stankovic 3 1	Andrija Petrovic 2 5
Lazar Micic 1 3	Uros Milic 2 5
Marija Stankovic 1 3	Nenad Brankovic 2 4
Nenad Brankovic 2 4	Marija Stankovic 1 3
Ognjen Peric 1 2	Lazar Micic 1 3
Uros Milic 2 5	Ognjen Peric 1 2

[Rešenje 3.24]

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

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

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

- nema opcija, sortiranje se vrši po broju gledanja;

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

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

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

[Rešenje 3.25]

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

#### *Primer 1*

INTERAKCIJA SA PROGRAMOM:
Unesi niz operacija: <b>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</b>

**\*\* Zadatak 3.27** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze

## **3.2 Algoritmi sortiranja**

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
1	1	4	4
2	2	—	5

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

*Test 1*

**ULAZ:**

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

**IZLAZ:**

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

**Zadatak 3.28** Za zadatu celobrojnu matricu dimenzije  $n \times m$  napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrstama. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

*Test 1*

**INTERAKCIJA SA PROGRAMOM:**

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

*Test 2*

**INTERAKCIJA SA PROGRAMOM:**

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

[Rešenje 3.28]

### 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.29** Za zadatu kvadratnu matricu dimenzije  $n$  napisati funkciju koja sortira kolone matrice opadajuće na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

*Primer 1*

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

*Primer 2*

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

## 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.30** 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 za greške. Niz struktura ima manje od 100 elemenata i uređen je u rastućem leksikografskom poretku po prezimenima. Pretazivanje niza vršiti bibliotečkom funkcijom **bsearch**. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

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

*Test 1*

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

*Test 2*

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

*Test 3*

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

### **3.3 Bibliotečke funkcije pretrage i sortiranja**

**Zadatak 3.31** Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova i mogućnost zadavanja različitih kriterijuma sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva, ne veća od 100, a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu. Na standardnom izlazu ispisati odgovarajuću poruku.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:  
Uneti dimenziju niza: 11  
Uneti elemente niza:  
5 3 1 6 8 90 34 5 3 432 34  
Sortirani niz u rastucem poretku:  
1 3 3 5 5 6 8 34 34 90 432  
Uneti element koji se trazi u nizu: 34  
Binarna pretraga:  
Element je nadjen na poziciji 8  
Linearna pretraga (lfind):  
Element je nadjen na poziciji 7
```

*Primer 2*

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

[Rešenje 3.31]

**Zadatak 3.32** Napisati program koji sa standardnog ulaza učitava dimenziju niza celih brojeva (ne veću od 100), a potom i same elemente niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku prema broju delilaca i tako dobijeni niz odštampati na standardnom izlazu.

*Primer 1*

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

*Primer 2*

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

*Primer 3*

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

[Rešenje 3.32]

**Zadatak 3.33** Korišćenjem bibliotečke funkcije `qsort` napisati program koji sortira niz niski po sledećim kriterijumima:

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

### **3.3 Bibliotečke funkcije pretrage i sortiranja**

---

Niske se učitavaju iz datoteke **niske.txt**. Prepostaviti 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 traženu 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 3.33]

**Zadatak 3.34** Uraditi prethodni zadatak 3.33 sa dinamički alociranim niskama i sortiranjem niza pokazivača, umesto niza niski.

[Rešenje 3.34]

**Zadatak 3.35** 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. Prepostaviti da neće biti više od 500 studenata i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

### **3.3 Bibliotečke funkcije pretrage i sortiranja**

#### *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 rastuce:  
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 3.35]

**Zadatak 3.36** Uraditi zadatak 3.13, ali korišćenjem bibliotečke `qsort` funkcije.

[Rešenje 3.36]

**Zadatak 3.37** 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 3.37]

**Zadatak 3.38** 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 3.38]

**Zadatak 3.39** 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 sledeći datum: 13.12.2016.
postoji
Unesi sledeći datum: 10.5.2015.
ne postoji
Unesi sledeći datum: 5.2.2009.
postoji
```

## 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 /* Naredne tri funkcije koje vrse pretragu, ukoliko se trazeni
10 element pronadje u nizu, vracaju indeks pozicije na kojoj je
11 element pronadjen. Ovaj indeks je uvek nenegativan. Ako element
12 nije pronadjen u nizu, funkcije vracaju negativnu vrednost -1, kao
13 indikator neuspesne pretrage. */
14
15 /* Linearna pretraga: Funkcija pretrazuje niz a[] celih brojeva
16 duzine n, trazeci u njemu prvo pojavljivanje elementa x. Pretraga
17 se vrsti prostom iteracijom kroz niz. */
18 int linearna_pretraga(int a[], int n, int x)
19 {
20     int i;
21     for (i = 0; i < n; i++)
22         if (a[i] == x)
23             return i;
24     return -1;
25 }
26
27 /* Binarna pretraga: Funkcija trazi u sortiranom nizu a[] duzine n
28 broj x. Pretraga koristi osobinu sortiranosti niza i u svakoj
29 iteraciji polovi interval pretrage. */
30 int binarna_pretraga(int a[], int n, int x)
31 {
32     int levi = 0;
33     int desni = n - 1;
34     int srednji;
35     /* Dokle god je indeks levi levo od indeksa desni */
36     while (levi <= desni) {
37         /* Srednji indeks je njihova aritmeticka sredina */
38         srednji = (levi + desni) / 2;
39         /* Ako je element sa sredisnjim indeksom veci od x, tada se x
40            mora nalaziti u levom delu niza */
41         if (x < a[srednji])
42             desni = srednji - 1;
43         /* Ako je element sa sredisnjim indeksom manji od x, tada se x
44            mora nalaziti u desnom delu niza */
45         else if (x > a[srednji])
46             levi = srednji + 1;

```

```

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

```

```

99     else if (x > a[srednji])
100        levi = srednji + 1;
101    else
102      /* Ako je element sa indeksom srednji jednak traženom, onda se
103         pretraga završava na poziciji srednji */
104      return srednji;
105  }
106  /* U slučaju neuspesne pretrage vraca se -1 */
107  return -1;
108 }

109 int main(int argc, char **argv)
110 {
111   int a[MAX];
112   int n, i, x;
113   struct timespec vreme1, vreme2, vreme3, vreme4, vreme5, vreme6;
114   FILE *f;
115   /* Provera argumenata komandne linije */
116   if (argc != 3) {
117     fprintf(stderr,
118             "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
119     ;
120     exit(EXIT_FAILURE);
121   }

122   /* Dimenzija niza */
123   n = atoi(argv[1]);
124   if (n > MAX || n <= 0) {
125     fprintf(stderr, "Dimenzija niza neodgovarajuća\n");
126     exit(EXIT_FAILURE);
127   }

128   /* Broj koji se trazi */
129   x = atoi(argv[2]);
130   /* Elementi niza se generisu slučajno, tako da je svaki sledeći
131      veci od prethodnog. Funkcija srand() inicijalizuje pocetnu
132      vrednost sa kojom se kreće u izracunavanje sekvence
133      pseudo-slučajnih brojeva. Kako generisani niz ne bi uvek bio
134      isti, ova vrednost se postavlja na tekuce vreme u sekundama od
135      Nove godine 1970, tako da je za svako sledeće pokretanje
136      programa (u vremenskim intervalima vecim od jedne sekunde) ove
137      vrednost drugacija. random()%100 vraca brojeve izmedju 0 i 99 */
138   srand(time(NULL));
139   for (i = 0; i < n; i++)
140     a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
141   /* Linearna pretraga */
142   printf("Linearna pretraga:\n");
143   /* Vreme proteklo od Nove godine 1970 */
144   clock_gettime(CLOCK_REALTIME, &vreme1);
145   i = linearna_pretraga(a, n, x);
146   /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
147      linearnu pretragu */
148 
```

```

151     clock_gettime(CLOCK_REALTIME, &vreme2);
152     if (i == -1)
153         printf("Element nije u nizu\n");
154     else
155         printf("Element je u nizu na poziciji %d\n", i);
156     /* Binarna pretraga */
157     printf("Binarna pretraga:\n");
158     clock_gettime(CLOCK_REALTIME, &vreme3);
159     i = binarna_pretraga(a, n, x);
160     clock_gettime(CLOCK_REALTIME, &vreme4);
161     if (i == -1)
162         printf("Element nije u nizu\n");
163     else
164         printf("Element je u nizu na poziciji %d\n", i);
165     /* Interpolaciona pretraga */
166     printf("Interpolaciona pretraga:\n");
167     clock_gettime(CLOCK_REALTIME, &vreme5);
168     i = interpolaciona_pretraga(a, n, x);
169     clock_gettime(CLOCK_REALTIME, &vreme6);
170     if (i == -1)
171         printf("Element nije u nizu\n");
172     else
173         printf("Element je u nizu na poziciji %d\n");
174     /* Podaci o izvrsavanju programa bivaju upisani u log fajl */
175     if ((f = fopen("vremena.txt", "a")) == NULL) {
176         fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
177         exit(EXIT_FAILURE);
178     }
179
180     fprintf(f, "Dimenzija niza: %d\n", n);
181     fprintf(f, "\tLinearna: %10ld ns\n",
182             (vreme2.tv_sec - vreme1.tv_sec) * 1000000000 +
183             vreme2.tv_nsec - vreme1.tv_nsec);
184     fprintf(f, "\tBinarna: %19ld ns\n",
185             (vreme4.tv_sec - vreme3.tv_sec) * 1000000000 +
186             vreme4.tv_nsec - vreme3.tv_nsec);
187     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
188             (vreme6.tv_sec - vreme5.tv_sec) * 1000000000 +
189             vreme6.tv_nsec - vreme5.tv_nsec);
190     /* Zatvaranje datoteke */
191     fclose(f);
192     exit(EXIT_SUCCESS);
193 }
```

### Rešenje 3.2

```

2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #define MAX 1024
```

```

6 int linearna_pretraga_r1(int a[], int n, int x)
7 {
8     int tmp;
9     /* Izlaz iz rekurzije */
10    if (n <= 0)
11        return -1;
12    /* Ako je prvi element traženi */
13    if (a[0] == x)
14        return 0;
15    /* Pretraga ostatka niza */
16    tmp = linearna_pretraga_r1(a + 1, n - 1, x);
17    return tmp < 0 ? tmp : tmp + 1;
18 }

20 int linearna_pretraga_r2(int a[], int n, int x)
21 {
22     /* Izlaz iz rekurzije */
23     if (n <= 0)
24         return -1;
25     /* Ako je poslednji element traženi */
26     if (a[n - 1] == x)
27         return n - 1;
28     /* Pretraga ostatka niza */
29     return linearna_pretraga_r2(a, n - 1, x);
30 }

32 int binarna_pretraga_r(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 sredisnoj poziciji traženi */
40     if (a[srednji] == x)
41         return srednji;
42     /* Ako je traženi broj veci od broja na sredisnoj poziciji,
43      pretrazuje se desna polovina niza */
44     if (a[srednji] < x)
45         return binarna_pretraga_r(a, srednji + 1, d, x);
46     /* Ako je traženi broj manji od broja na sredisnoj poziciji,
47      pretrazuje se leva polovina niza */
48     else
49         return binarna_pretraga_r(a, l, srednji - 1, x);
50 }

52 int interpolaciona_pretraga_r(int a[], int l, int d, int x)
53 {
54     int p;
55     if (x < a[l] || x > a[d])
56         return -1;

```

```

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

65 int main()
66 {
67     int a[MAX];
68     int x;
69     int i, indeks;

70  /* Ucitavanje traženog broja */
71  printf("Unesite traženi broj: ");
72  scanf("%d", &x);

73  /* Ucitavanje elemenata niza sve do kraja ulaza - očekuje se da
     korisnik pritisne CTRL+D za naznaku kraja */
74  i = 0;
75  printf("Unesite sortiran niz elemenata: ");
76  while (i < MAX && scanf("%d", &a[i]) == 1) {
77      if (i > 0 && a[i] < a[i - 1]) {
78          fprintf(stderr,
79                  "Elementi moraju biti uneseni u neopadajućem poretku\n");
80          exit(EXIT_FAILURE);
81      }
82      i++;
83  }

84  /* Linearna pretraga */
85  printf("Linearna pretraga\n");
86  indeks = linearna_pretraga_r1(a, i, x);
87  if (indeks == -1)
88      printf("Element se ne nalazi u nizu.\n");
89  else
90      printf("Pozicija elementa je %d.\n", indeks);

91  /* Binarna pretraga */
92  printf("Binarna pretraga\n");
93  indeks = binarna_pretraga_r(a, 0, i - 1, x);
94  if (indeks == -1)
95      printf("Element se ne nalazi u nizu.\n");
96  else
97      printf("Pozicija elementa je %d.\n", indeks);
98
99
100
101
102
103
104
105
106
107
108

```

```

110  /* Interpolaciona pretraga */
111  printf("Interpolaciona pretraga\n");
112  indeks = interpolaciona_pretraga_r(a, 0, i - 1, x);
113  if (indeks == -1)
114      printf("Element se ne nalazi u nizu.\n");
115  else
116      printf("Pozicija elementa je %d.\n", indeks);
117
118 }
```

### Rešenje 3.3

```

#include <stdio.h>
2 #include <stdlib.h>
# include <string.h>
4
#define MAX_STUDENATA 128
6 #define MAX_DUZINA 16

8 /* O svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
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;

18 /* Ucitani niz studenata ce biti sortiran rastuce prema indeksu, jer
20    su studenti u datoteci vec sortirani. Iz tog razloga pretraga po
22    indeksu se vrsti binarno, dok pretraga po prezimenu mora linearno,
24    jer nema garancije da postoji uredjenje po prezimenu. */

26 /* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
28    sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
30    ako element nije pronadjen. */
32 int binarna_pretraga(Student a[], int n, long x)
{
34     int levi = 0;
35     int desni = n - 1;
36     int srednji;
37     /* Dokle god je indeks levi levo od indeksa desni */
38     while (levi <= desni) {
39         /* Racuna se srednja pozicija */
40         srednji = (levi + desni) / 2;
41         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
42            se trazeni indeks mora nalaziti u levoj polovini niza */
43         if (x < a[srednji].indeks)
44             desni = srednji - 1;
45     }
46 }
```

```

40     /* Ako je pak manji od traženog, tada se on mora nalaziti u
41      desnoj polovini niza */
42     else if (x > a[srednji].indeks)
43         levi = srednji + 1;
44     else
45         /* Ako je jednak traženom indeksu x, tada je pronađen student
46          sa traženom indeksom na poziciji srednji */
47         return srednji;
48     /* Ako nije pronađen, vraca se -1 */
49     return -1;
50 }

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

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

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

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

```

```

    exit(EXIT_FAILURE);
}

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

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

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

/* Pretraga po indeksu */
if (bin_pretraga) {
    /* Unos indeksa koji se binarno trazi u nizu */
    printf("Unesite indeks studenta cije informacije zelite: ");
    scanf("%ld", &trazen_indeks);
    i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
    /* Rezultat binarne pretrage */
    if (i == -1)
        printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
        printf("Indeks: %ld, Ime i prezime: %s %s\n",
               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
}
/* Pretraga po prezimenu */
else {
    /* Unos prezimena koje se linearno trazi u nizu */
    printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
    i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
    /* Rezultat linearne pretrage */
    if (i == -1)
        printf("Ne postoji student sa prezimenom %s\n",
               trazeno_prezime);
    else
}

```

```

142     printf("Indeks: %ld, Ime i prezime: %s %s\n",
143            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
144 }
145 exit(EXIT_SUCCESS);
146 }
```

### Rešenje 3.4

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
7
8 typedef struct {
9     long indeks;
10    char ime[MAX_DUZINA];
11    char prezime[MAX_DUZINA];
12 } Student;
13
14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
15                                     long x)
16 {
17     /* Ako je pozicija elementa na levom kraju veca od pozicije
18        elementa na desnom kraju dela niza koji se pretrazuje, onda se
19        zapravo pretrazuje prazan deo niza. U praznom delu niza nema
20        trazenog elementa pa se vraca -1 */
21     if (levi > desni)
22         return -1;
23     /* Racunanje pozicije srednjeg elementa */
24     int srednji = (levi + desni) / 2;
25     /* Da li je srednji bas onaj trazeni */
26     if (a[srednji].indeks == x) {
27         return srednji;
28     }
29     /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
30        poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
31        je poznato da je niz sortiran po indeksu u rastucem poretku. */
32     if (x < a[srednji].indeks)
33         return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34     /* Inace ga treba traziti u desnoj polovini */
35     else
36         return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37 }
38
39 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
41     /* Ako je niz prazan, vraca se -1 */
42     if (n == 0)
43         return -1;
44 }
```

```

44  /* Kako se trazi prvi student sa traženim prezimenom, pocinje se sa
45   prvim studentom u nizu. */
46  if (strcmp(a[0].prezime, x) == 0)
47    return 0;
48  int i = linearna_pretraga_rekursivna_v2(a + 1, n - 1, x);
49  return i >= 0 ? 1 + i : -1;
50 }

52 int linearna_pretraga_rekursivna(Student a[], int n, char x[])
53 {
54  /* Ako je niz prazan, vraca se -1 */
55  if (n == 0)
56    return -1;
57  /* Ako se trazi poslednji student sa traženim 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_rekursivna(a, n - 1, x);
62 }

64 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
65  prosledjuju kao argumenti komandne linije */
66 int main(int argc, char *argv[])
67 {
68  Student dosije[MAX_STUDENATA];
69  FILE *fin = NULL;
70  int i;
71  int br_studenata = 0;
72  long tražen_indeks = 0;
73  char traženo_prezime[MAX_DUZINA];
74  int bin_pretraga;

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

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

94  /* Otvaranje datoteke */

```

```

96     fin = fopen(argv[1], "r");
98     if (fin == NULL) {
100         fprintf(stderr,
101             "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
102         exit(EXIT_FAILURE);
103     }
104
105     /* Citanje se vrši sve dok postoji red sa informacijama o studentu */
106     i = 0;
107     while (1) {
108         if (i == MAX_STUDENATA)
109             break;
110         if (fscanf(
111             (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
112             dosije[i].prezime) != 3)
113             break;
114         i++;
115     }
116     br_studenata = i;
117
118     /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
119     fclose(fin);
120
121     /* Pretraga po indeksu */
122     if (bin_pretraga) {
123         /* Unos indeksa koji se binarno trazi u nizu */
124         printf("Unesite indeks studenta cije informacije zelite: ");
125         scanf("%ld", &trazen_indeks);
126         i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata,
127                                         trazen_indeks);
128
129         /* Rezultat binarne pretrage */
130         if (i == -1)
131             printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
132         else
133             printf("Indeks: %ld, Ime i prezime: %s %s\n",
134                   dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135     }
136
137     /* Pretraga po prezimenu */
138     else {
139         /* Unos prezimena koje se linearно trazi u nizu */
140         printf("Unesite prezime studenta cije informacije zelite: ");
141         scanf("%s", trazeno_prezime);
142         i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
143                                              trazeno_prezime);
144
145         /* Rezultat linearne pretrage */
146         if (i == -1)
147             printf("Ne postoji student sa prezimenom %s\n",
148                   trazeno_prezime);
149         else
150             printf("Indeks: %ld, Ime i prezime: %s %s\n",
151                   dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
152     }

```

```

148     }
149     exit(EXIT_SUCCESS);
150 }
```

### Rešenje 3.5

```

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

```

    }
    return najbliza;
}

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

#define MAX 1024

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

    /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
       ime datoteke sa tackama */
    if (argc < 2) {
        fprintf(stderr,
                "koriscenje programa: %s ime_datoteke\n",
                argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Otvaranje datoteke za citanje */
    ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
                argv[1]);
        exit(EXIT_FAILURE);
    }

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

```

```

99  /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
100    dodatnih argumenata */
101  if (argc == 2)
102    /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
103    najbliza = najbliza_koordinatnom(tacke, n);
104  /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
105    pitanju opcija -x */
106  else if (strcmp(argv[2], "-x") == 0)
107    /* Racuna se rastojanje u odnosu na x osu */
108    najbliza = najbliza_x_osi(tacke, n);
109  /* Ako je u pitanju opcija -y */
110  else if (strcmp(argv[2], "-y") == 0)
111    /* Racuna se rastojanje u odnosu na y osu */
112    najbliza = najbliza_y_osi(tacke, n);
113  else {
114    /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
       korisnika i prekida se izvrsavanje programa */
115    fprintf(stderr, "Pogresna opcija\n");
116    exit(EXIT_FAILURE);
117  }

118  /* Stampanje koordinata trazene tacke */
119  printf("%g %g\n", najbliza.x, najbliza.y);
120
121  /* Zatvaranje datoteke */
122  fclose(ulaz);
123
124  exit(EXIT_SUCCESS);
}

```

### Rešenje 3.6

```

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

/* Tacnost */
#define EPS 0.001

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

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

    /* Sve dok se ne pronadje trazena vrednost argumenta */
    while (1) {
        /* Polovi se interval */
        s = (l + d) / 2;

```

```

20      /* Ako je absolutna vrednost kosinusa u ovoj tacki manja od
21         zadate tacnosti, prekida se pretraga */
22      if (fabs(cos(s)) < EPS) {
23          break;
24      }
25      /* Ako je nula u levom delu intervala, nastavlja se pretraga na
26         [l, s] */
27      if (cos(l) * cos(s) < 0)
28          d = s;
29      else
30          /* Inace, na intervalu [s, d] */
31          l = s;
32
33      /* Stampanje vrednosti trazene tacke */
34      printf("%g\n", s);
35
36  }

```

### Rešenje 3.7

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6 int main(int argc, char **argv)
7 {
8     double l, d, s, epsilon;
9
10    char ime_fje[6];
11
12    /* Pokazivac na funkciju koja ima jedan argument tipa double i
13       povratnu vrednost istog tipa */
14    double (*fp) (double);
15
16    /* Ako korisnik nije uneo argument, prijavljuje se greska */
17    if (argc != 2) {
18        fprintf(stderr, "Greska: ");
19        fprintf(stderr, "Nedovoljan broj argumenata komandne linije.\n");
20        fprintf(stderr,
21                "Program se poziva sa %s ime_funkcije iz math.h.\n",
22                argv[0]);
23        exit(EXIT_FAILURE);
24    }
25
26    /* Niska ime_fje sadrzi ime trazene funkcije koja je navedena u
27       komandnoj liniji */
28    strcpy(ime_fje, argv[1]);

```

```

30  /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
31  if (strcmp(ime_fje, "sin") == 0)
32    fp = &sin;
33  else if (strcmp(ime_fje, "cos") == 0)
34    fp = &cos;
35  else if (strcmp(ime_fje, "tan") == 0)
36    fp = &tan;
37  else if (strcmp(ime_fje, "atan") == 0)
38    fp = &atan;
39  else if (strcmp(ime_fje, "acos") == 0)
40    fp = &acos;
41  else if (strcmp(ime_fje, "asin") == 0)
42    fp = &asin;
43  else if (strcmp(ime_fje, "exp") == 0)
44    fp = &exp;
45  else if (strcmp(ime_fje, "log") == 0)
46    fp = &log;
47  else if (strcmp(ime_fje, "log10") == 0)
48    fp = &log10;
49  else if (strcmp(ime_fje, "sqrt") == 0)
50    fp = &sqrt;
51  else {
52    fprintf(stderr, "Program ne podrzava trazenu funkciju!\n");
53    exit(EXIT_SUCCESS);
54  }

55  printf("Unesite krajeve intervala: ");
56  scanf("%lf %lf", &l, &d);

57  if ((*fp) (l) * (*fp) (d) >= 0) {
58    fprintf(stderr,
59      "Funkcija %s na intervalu [%g, %g] ne zadovoljava uslove\
60      \n",
61      ime_fje, l, d);
62    exit(EXIT_FAILURE);
63  }

64  printf("Unesite preciznost: ");
65  scanf("%lf", &epsilon);

66  /* Sve dok se ne pronadje trazena vrednost argumenta */
67  while (1) {
68    /* Polovi se interval */
69    s = (l + d) / 2;
70    /* Ako je apsolutna vrednost trazene funkcije u ovoj tacki manja
71     od zadate tacnosti, prekida se pretraga */
72    if (fabs((*fp) (s)) < epsilon) {
73      break;
74    }
75    /* Ako je nula u levom delu intervala, nastavlja se pretraga na
76     [l, s] */
77    if ((*fp) (l) * (*fp) (s) < 0)
78
79
80

```

```

82     d = s;
83     else
84         /* Inace, na intervalu [s, d] */
85         l = s;
86     }
87
88     /* Stampa vrednosti trazene tacke */
89     printf("%g\n", s);
90
91     return 0;
92 }
```

## Rešenje 3.8

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 256
5
6 int prvi_veci_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 veci od nule, a eventualni njegov
16           prethodnik manji ili jednak nuli, pretraga je zavrsena */
17        if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
18            return s;
19        /* U slucaju broja manjeg ili jednakog nuli, pretrazuje se desna
20           polovina niza */
21        if (niz[s] <= 0)
22            l = s + 1;
23        /* A inace, leva polovina */
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  /* Stampanje rezultata */
39  printf("%d\n", prvi_veci_od_nule(niz, n));
40
41  return 0;
42 }
```

## Rešenje 3.9

```

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 eventualni njegov
           prethodnik veci ili jednak nuli, pretraga se zavrsava */
16        if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
17            return s;
18        /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
           niza */
19        if (niz[s] >= 0)
20            l = s + 1;
21        /* A inace leva */
22        else
23            d = s - 1;
24    }
25    return -1;
26 }
27
28 int main()
29 {
30     int niz[MAX];
31     int n = 0;
32
33     /* Unos niza */
34     while (scanf("%d", &niz[n]) == 1)
35         n++;
36
37     /* Stampanje rezultata */
38     printf("%d\n", prvi_manji_od_nule(niz, n));
39
40     return 0;
41 }
```

43 }

**Rešenje 3.10**

```

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     /* Rekursivni 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 vrsti 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);

```

```

49    }
50
51 /* Ispis povratnih vrednosti funkcija */
52 printf("%u %u\n", logaritam_a(x), logaritam_b(x));
53
54 exit(EXIT_SUCCESS);
55 }
```

### Rešenje 3.12

Datoteka 3.1: *sort.h*

```

1 #ifndef _SORT_H_
2 #define _SORT_H_ 1

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

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

28 /* Bubble sort: Funkcija sortira niz celih brojeva metodom mehurica.
   29  Ideja algoritma je sledeća: prolazi se kroz niz redom poredeći
   30  susedne elemente, i pri tom ih zamenjujući ako su u pogresnom
   31  poretku. Ovim se najveći element poput mehurica istiskuje na
   32  "površinu", tj. na krajnju desnu poziciju. Nakon toga je potrebno
   33  ovaj postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih
   34  n-1 elemenata niza bez poslednjeg koji je postavljen na pravu
   35  poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
   36  kracim prefiksima niza, cime se jedan po jedan istiskuju
   37  elemenenti na svoje prave pozicije. */
 38 }
```

```

1 void bubble_sort(int a[], int n);
2
3 /* Selsort: Ovaj algoritam je jednostavno prosirenje sortiranja
4  umetanjem koje dopusta direktnu razmenu udaljenih elemenata.
5  Prosirenje se sastoji u tome da se kroz algoritam umetanja prolazi
6  vise puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
7  koji je manji od n (sto omogucuje razmenu udaljenih elemenata) i
8  tako se dobija h-sortiran niz, tj. niz u kome su elementi na
9  rastojanju h sortirani, mada susedni elementi to ne moraju biti. U
10 drugom prolazu kroz isti algoritam sprovodi se isti postupak ali
11 za manji korak h. Sa prolazima se nastavlja sve do koraka h = 1, u
12 kome se dobija potpuno sortirani niz. Izbor pocetne vrednosti za
13 h, i nacina njegovog smanjivanja menja u nekim slucajevima brzinu
14 algoritma, ali bilo koja vrednost ce rezultovati ispravnim
15 sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
16 vrednost 1. */
17 void shell_sort(int a[], int n);
18
19 /* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
20 Sortiranje se vrsti od elementa na poziciji l do onog na poziciji
21 d. Na pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a
22 d je jednak poslednjem validnom indeksu u nizu. Funkcija niz
23 podeli na dve polovine, levu i desnu, koje zatim rekurzivno
24 sortira. Od ova dva sortirana podniza, sortiran niz se dobija
25 ucesljavanjem, tj. istovremenim prolaskom kroz oba niza i izborom
26 trenutnog manjeg elementa koji se smesta u pomocni niz. Na kraju
27 algoritma, sortirani elementi su u pomocnom nizu, koji se kopira u
28 originalni niz. */
29 void merge_sort(int a[], int l, int d);
30
31 /* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
32 i d. Njena ideja sortiranja je izbor jednog elementa niza, koji se
33 naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
34 svi elementi levo od njega bice manji, a svi desno bice veci od
35 njega. Kako je pivot doveden na svoje mesto, da bi niz bio
36 kompletno sortiran, potrebno je sortirati elemente levo (manje) od
37 njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
38 manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
39 deo moze se uraditi rekurzivno. */
40 void quick_sort(int a[], int l, int d);
41
42 #endif

```

Datoteka 3.2: *sort.c*

```

1 #include "sort.h"
2
3 #define MAX 1000000
4
5 void selection_sort(int a[], int n)
6 {

```

```

    int i, j;
8   int min;
9   int pom;
10
11  /* U svakoj iteraciji ove petlje pronalazi se najmanji element
12     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
14     min, na kojoj se nalazio najmanji od gore navedenih elemenata.
15 */
16  for (i = 0; i < n - 1; i++) {
17    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
18       najmanji od elemenata a[i],...,a[n-1]. */
19    min = i;
20    for (j = i + 1; j < n; j++)
21      if (a[j] < a[min])
22        min = j;
23
24    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
25       su (i) i min razliciti, inace je nepotrebno. */
26    if (min != i) {
27      pom = a[i];
28      a[i] = a[min];
29      a[min] = pom;
30    }
31  }
32
33 void insertion_sort(int a[], int n)
34 {
35   int i, j;
36
37   /* Na pocetku iteracije prepostavlja se da je niz a[0],...,a[i-1]
38      sortiran */
39   for (i = 1; i < n; i++) {
40
41     /* U ovoj petlji se redom potiskuje element a[i] uлево koliko je
42        potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...,a[i]
43        bude sortiran. Indeks j je trenutna pozicija na kojoj se
44        element koji se umesti nalazi. Petlja se zavrsava ili kada
45        element dodje do levog kraja (j==0) ili kada se naidje na
46        element a[j-1] koji je manji od a[j]. */
47     int temp = a[i];
48     for (j = i; j > 0 && temp < a[j - 1]; j--)
49       a[j] = a[j - 1];
50     a[j] = temp;
51   }
52 }
53
54 void bubble_sort(int a[], int n)
55 {
56   int i, j;
57   int ind;

```

```

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

```

110  /* Odredjivanje srednjeg indeksa */
111  s = (l + d) / 2;
112
113  /* Rekurzivni pozivi */
114  merge_sort(a, l, s);
115  merge_sort(a, s + 1, d);
116
117  /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
118  niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
119  prolazi kroz pomocni niz b[] */
120  i = l;
121  j = s + 1;
122  k = 0;
123
124  /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
125  while (i <= s && j <= d) {
126      if (a[i] < a[j])
127          b[k++] = a[i++];
128      else
129          b[k++] = a[j++];
130  }
131
132  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive j
133  iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
134  polovine niza */
135  while (i <= s)
136      b[k++] = a[i++];
137
138  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive i
139  iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
140  polovine niza */
141  while (j <= d)
142      b[k++] = a[j++];
143
144  /* Prepisuje se "ucesljani" niz u originalni niz */
145  for (k = 0, i = l; i <= d; i++, k++)
146      a[i] = b[k];
147
148  /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
149  void swap(int a[], int i, int j)
150  {
151      int tmp = a[i];
152      a[i] = a[j];
153      a[j] = tmp;
154  }
155
156  void quick_sort(int a[], int l, int d)
157  {
158      int i, pivot_pozicija;
159
160      /* Izlaz iz rekurzije -- prazan niz */

```

```

162 if (l >= d)
    return;
164
166 /* Partitionisanje niza. Svi elementi na pozicijama levo od
   pivot_pozicija (izuzev same pozicije l) su strogo manji od
   pivota. Kada se pronadje neki element manji od pivota, uvecava
   se promenljiva pivot_pozicija i na tu poziciju se premeta
   nadjeni element. Na kraju ce pivot_pozicija zaista biti pozicija
   na koju treba smestiti pivot, jer ce svi elementi levo od te
   pozicije biti manji a desno biti veci ili jednaki od pivota. */
172 pivot_pozicija = l;
174 for (i = l + 1; i <= d; i++)
    if (a[i] < a[l])
        swap(a, ++pivot_pozicija, i);
176
178 /* Postavljanje pivota na svoje mesto */
180 swap(a, l, pivot_pozicija);
182
184 /* Rekurzivno sortiranje elemenata manjih od pivota */
quick_sort(a, l, pivot_pozicija - 1);
/* Rekurzivno sortiranje elemenata vecih od pivota */
quick_sort(a, pivot_pozicija + 1, d);
}

```

Datoteka 3.3: *main.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "sort.h"
5
6 /* Maksimalna duzina niza */
7 #define MAX 1000000
8
9 int main(int argc, char *argv[])
{
10    //*****
11    tip_sortiranja == 0 => selectionsort, (podrazumevano)
12    tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
13    tip_sortiranja == 2 => bubblesort,      -b opcija komandne linije
14    tip_sortiranja == 3 => shellsort,       -s opcija komandne linije
15    tip_sortiranja == 4 => mergesort,        -m opcija komandne linije
16    tip_sortiranja == 5 => quicksort,        -q opcija komandne linije
17    //*****
18    int tip_sortiranja = 0;
19    //*****
20    tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
21    tip_niza == 1 => rastuce sortirani nizovi,      -r opcija
22    tip_niza == 2 => opadajuce sortirani nizovi, -o opcija
23    //*****

```

```

25     int tip_niza = 0;

27     /* Dimenzija niza koji se sortira */
28     int dimenzija;
29     int i;
30     int niz[MAX];

31     /* Provera argumenata komandne linije */
32     if (argc < 2) {
33         fprintf(stderr,
34             "Program zahteva bar 2 argumenta komandne linije!\n");
35         exit(EXIT_FAILURE);
36     }

37     /* Ocitavanje opcija i argumenata prilikom poziva programa */
38     for (i = 1; i < argc; i++) {
39         /* Ako je u pitanju opcija... */
40         if (argv[i][0] == '-') {
41             switch (argv[i][1]) {
42                 case 'i':
43                     tip_sortiranja = 1;
44                     break;
45                 case 'b':
46                     tip_sortiranja = 2;
47                     break;
48                 case 's':
49                     tip_sortiranja = 3;
50                     break;
51                 case 'm':
52                     tip_sortiranja = 4;
53                     break;
54                 case 'q':
55                     tip_sortiranja = 5;
56                     break;
57                 case 'r':
58                     tip_niza = 1;
59                     break;
60                 case 'o':
61                     tip_niza = 2;
62                     break;
63                 default:
64                     printf("Pogresna opcija -%c\n", argv[i][1]);
65                     return 1;
66                     break;
67             }
68         }
69     }

70     /* Ako je u pitanju argument, onda je to duzina niza koji treba
71      da se sortira */
72     else {
73         dimenzija = atoi(argv[i]);
74         if (dimenzija <= 0 || dimenzija > MAX) {
75             fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
76         }
77     }
78 }
```

```

77         exit(EXIT_FAILURE);
78     }
79 }
80
81 /* Elementi niza se odredjuju slučajno, ali vodeći računa o tipu
82  niza dobijenom iz komandne linije. srand() funkcija obezbeđuje
83  novi seed za pozivanje rand funkcije, i kako generisani niz ne
84  bi uvek bio isti seed je postavljen na tekuće vreme u sekundama
85  od Nove godine 1970. rand()%100 daje brojeve između 0 i 99 */
86 srand(time(NULL));
87 if (tip_niza == 0)
88     for (i = 0; i < dimenzija; i++)
89         niz[i] = rand();
90 else if (tip_niza == 1)
91     for (i = 0; i < dimenzija; i++)
92         niz[i] = i == 0 ? rand() % 100 : niz[i - 1] + rand() % 100;
93 else
94     for (i = 0; i < dimenzija; i++)
95         niz[i] = i == 0 ? rand() % 100 : niz[i - 1] - rand() % 100;
96
97 /* Ispisivanje elemenata niza */
98 ****
99 Ovaj deo je iskomentarisan jer sledeći ispis ne treba da se nadje
100 na standardnom izlazu. Njegova svrha je samo bila provera da li
101 je niz generisan u skladu sa opcijama komandne linije.
102
103 printf("Niz koji sortiramo je:\n");
104 for (i = 0; i < dimenzija; i++)
105     printf("%d\n", niz[i]);
106 ****
107
108
109 /* Sortiranje niza na odgovarajući način */
110 if (tip_sortiranja == 0)
111     selection_sort(niz, dimenzija);
112 else if (tip_sortiranja == 1)
113     insertion_sort(niz, dimenzija);
114 else if (tip_sortiranja == 2)
115     bubble_sort(niz, dimenzija);
116 else if (tip_sortiranja == 3)
117     shell_sort(niz, dimenzija);
118 else if (tip_sortiranja == 4)
119     merge_sort(niz, 0, dimenzija - 1);
120 else
121     quick_sort(niz, 0, dimenzija - 1);
122
123 /* Ispis elemenata niza */
124 ****
125 Ovaj deo je iskomentarisan jer vreme potrebno za njegovo
126 izvršavanje ne bi trebalo da bude uključeno u vreme izmereno
127 programom time. Takođe, kako je svrha ovog programa da prikaze

```

```

129     vremena razlicitih algoritama sortiranja, dimenzije nizova ce
130     biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
131     od toliko elemenata. Ovaj deo je koriscen u razvoju programa
132     zarad testiranja korektnosti.
133
134     printf("Sortiran niz je:\n");
135     for (i = 0; i < dimenzija; i++)
136         printf("%d\n", niz[i]);
137     ****
138
139     exit(EXIT_SUCCESS);
}

```

### Rešenje 3.13

```

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

#define MAX_DIM 128

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

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

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

    /* Sortiramo niske */
    selectionSort(s);
    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
49   /* Ucitavanje niski sa ulaza */
50   printf("Unesite prvu nisku: ");
51   scanf("%s", s);
52   printf("Unesite drugu nisku: ");
53   scanf("%s", t);
54
55   /* Poziv funkcije */
56   if (anagrami(s, t))
57     printf("jesu\n");
58   else
59     printf("nisu\n");
60
61   return 0;
62 }
```

### Rešenje 3.14

```

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

4 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
5    sortiranom nizu celih brojeva */
6 int najmanje_rastojanje(int a[], int n)
7 {
8   int i, min;
9   min = a[1] - a[0];
10  for (i = 2; i < n; i++)
11    if (a[i] - a[i - 1] < min)
12      min = a[i] - a[i - 1];
13  return min;
14 }

16
17 int main()
18 {
19   int i, a[MAX];
20
21   /* Ucitavaju se elementi niza sve do kraja ulaza */
```

```

23     i = 0;
24     while (scanf("%d", &a[i]) != EOF)
25         i++;
26
27     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
28      sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
29      se selection sort. */
30     selection_sort(a, i);
31
32     /* Ispis rezultata */
33     printf("%d\n", najmanje_rastojanje(a, i));
34
35     return 0;
36 }
```

### Rešenje 3.15

```

1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX_DIM 256
4
5 /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
6    najvise puta pojavio u tom nizu */
7 int najvise_puta(int a[], int n)
8 {
9     int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
10    /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
11    for (i = 0; i < n; i = j) {
12        br_pojava = 1;
13        for (j = i + 1; j < n && a[i] == a[j]; j++)
14            br_pojava++;
15        /* Ispitivanje da li se do tog trenutka i-ti element pojavio
16           najvise puta u nizu */
17        if (br_pojava > max_br_pojava) {
18            max_br_pojava = br_pojava;
19            i_max_pojava = i;
20        }
21    }
22    /* Vraca se element koji se najvise puta pojavio u nizu */
23    return a[i_max_pojava];
24 }
25
26 int main()
27 {
28     int a[MAX_DIM], i;
29
30     /* Ucitavanje elemenata niza sve do kraja ulaza */
31     i = 0;
32     while (scanf("%d", &a[i]) != EOF)
33         i++;
```

```

35  /* Za sortiranje niza može se koristiti bilo koja od funkcija
36   sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
37   se merge sort. */
38   merge_sort(a, 0, i - 1);
39
40  /* Određjuje se broj koji se najviše puta pojavio u nizu */
41  printf("%d\n", najvise puta(a, i));
42
43  return 0;
}

```

### Rešenje 3.16

```

1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX_DIM 256
4
5 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
6  u nizu, a 0 inace. Prepostavlja se da je niz sortiran u rastucem
7  poretku */
8 int binarna_pretraga(int a[], int n, int x)
9 {
10    int levi = 0, desni = n - 1, srednji;
11
12    while (levi <= desni) {
13        srednji = (levi + desni) / 2;
14        if (a[srednji] == x)
15            return 1;
16        else if (a[srednji] > x)
17            desni = srednji - 1;
18        else if (a[srednji] < x)
19            levi = srednji + 1;
20    }
21    return 0;
22}
23
24 int main()
25 {
26    int a[MAX_DIM], n = 0, zbir, i;
27
28    /* Ucitava se traženi zbir */
29    printf("Unesite traženi zbir: ");
30    scanf("%d", &zbir);
31
32    /* Ucitavaju se elementi niza sve do kraja ulaza */
33    i = 0;
34    printf("Unesite elemente niza: ");
35    while (scanf("%d", &a[i]) != EOF)
36        i++;
37    n = i;
}

```

```

39  /* Za sortiranje niza moze se koristiti bilo koja od funkcija
40   sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
41   se quick sort. */
42   quick_sort(a, 0, n - 1);
43
44   for (i = 0; i < n; i++)
45     /* Za i-ti element niza binarno se pretrazuje da li se u ostatku
46      niza nalazi element koji sabran sa njim ima ucitanu vrednost
47      zbiru */
48     if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
49       printf("da\n");
50       return 0;
51     }
52   printf("ne\n");
53
54   return 0;
55 }
```

### Rešenje 3.17

```

1 #include <stdio.h>
2 #define MAX_DIM 256
3
4 int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
5           int dim3)
6 {
7   int i = 0, j = 0, k = 0;
8   /* U slučaju da je dimenzija treceg niza manja od neophodne,
9    funkcija vraca -1 */
10  if (dim3 < dim1 + dim2)
11    return -1;
12
13  /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog
14   od njih */
15  while (i < dim1 && j < dim2) {
16    if (niz1[i] < niz2[j])
17      niz3[k++] = niz1[i++];
18    else
19      niz3[k++] = niz2[j++];
20  }
21  /* Ostatak prvog niza prepisujemo u treći */
22  while (i < dim1)
23    niz3[k++] = niz1[i++];
24
25  /* Ostatak drugog niza prepisujemo u treći */
26  while (j < dim2)
27    niz3[k++] = niz2[j++];
28  return dim1 + dim2;
29 }
30
31 int main()
```

```

32 {
33     int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
34     int i = 0, j = 0, k, dim3;
35
36     /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
37      Pretpostavka je da na ulazu nece biti vise od MAX_DIM elemenata
38      */
39     printf("Unesite elemente prvog niza: ");
40     while (1) {
41         scanf("%d", &niz1[i]);
42         if (niz1[i] == 0)
43             break;
44         i++;
45     }
46     printf("Unesite elemente drugog niza: ");
47     while (1) {
48         scanf("%d", &niz2[j]);
49         if (niz2[j] == 0)
50             break;
51         j++;
52     }
53     /* 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 3.18

```

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    }
```

```

16     exit(EXIT_FAILURE);
17 }
18
19 /* Otvaranje datoteke zadate prvim argumentom komandne linije */
20 fin1 = fopen(argv[1], "r");
21 if (fin1 == NULL) {
22     fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
23     exit(EXIT_FAILURE);
24 }
25
26 /* Otvaranje datoteke zadate drugim argumentom komandne linije */
27 fin2 = fopen(argv[2], "r");
28 if (fin2 == NULL) {
29     fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
30     exit(EXIT_FAILURE);
31 }
32
33 /* Otvaranje datoteke za upis rezultata */
34 fout = fopen("ceo-tok.txt", "w");
35 if (fout == NULL) {
36     fprintf(stderr,
37             "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
38     exit(EXIT_FAILURE);
39 }
40
41 /* Citanje narednog studenta iz prve datoteke */
42 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
43     kraj1 = 1;
44
45 /* Citanje narednog studenta iz druge datoteke */
46 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
47     kraj2 = 1;
48
49 /* Sve dok nije dostignut kraj neke datoteke */
50 while (!kraj1 && !kraj2) {
51     int tmp = strcmp(ime1, ime2);
52     if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 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     /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
70      druge datoteke, onda ima jos studenata u prvoj datoteci, koje
71      treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
72      */
73      while (!kraj1) {
74          fprintf(fout, "%s %s\n", ime1, prezime1);
75          if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
76              kraj1 = 1;
77      }
78
79      /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
80         datoteke, onda ima jos studenata u drugoj datoteci, koje treba
81         prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
82      while (!kraj2) {
83          fprintf(fout, "%s %s\n", ime2, prezime2);
84          if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
85              kraj2 = 1;
86      }
87
88      /* Zatvaranje datoteka */
89      fclose(fin1);
90      fclose(fin2);
91      fclose(fout);
92
93      exit(EXIT_SUCCESS);
94  }

```

### Rešenje 3.19

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 #define MAX_BR_TACAKA 128
7
8 /* Struktura koja reprezentuje koordinate tacke */
9 typedef struct Tacka {
10     int x;
11     int y;
12 } Tacka;
13
14 /* Funkcija racuna rastojanje zadate tacke od koordinatnog pocetka
15   (0,0) */
16 float rastojanje(Tacka A)
17 {
18     return sqrt(A.x * A.x + A.y * A.y);
19 }
20
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;
26     Tacka tmp;

27     for (i = 0; i < n - 1; i++) {
28         min = i;
29         for (j = i + 1; j < n; j++) {
30             if (rastojanje(t[j]) < rastojanje(t[min])) {
31                 min = j;
32             }
33         }
34         if (min != i) {
35             tmp = t[i];
36             t[i] = t[min];
37             t[min] = tmp;
38         }
39     }
40 }

43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
void sortiraj_po_x(Tacka t[], int n)
{
45     int min, i, j;
46     Tacka tmp;

49     for (i = 0; i < n - 1; i++) {
50         min = i;
51         for (j = i + 1; j < n; j++) {
52             if (abs(t[j].x) < abs(t[min].x)) {
53                 min = j;
54             }
55         }
56         if (min != i) {
57             tmp = t[i];
58             t[i] = t[min];
59             t[min] = tmp;
60         }
61     }
62 }

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

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

```

```

    min = j;
}
}
if (min != i) {
    tmp = t[i];
    t[i] = t[min];
    t[min] = tmp;
}
}

int main(int argc, char *argv[])
{
FILE *ulaz;
FILE *izlaz;
Tacka tacke[MAX_BR_TACAKA];
int i, n;
/*
Proveravanje broja argumenata komandne linije: očekuje se ime
izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
datoteke, tj. 4 argumenta */
if (argc != 4) {
    fprintf(stderr,
            "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
    return 0;
}

/* Otvaranje datoteke u kojoj su zadate tacke */
ulaz = fopen(argv[2], "r");
if (ulaz == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[2]);
    return 0;
}

/* Otvaranje datoteke u koju treba upisati rezultat */
izlaz = fopen(argv[3], "w");
if (izlaz == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[3]);
    return 0;
}

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

/* Ukupan broj procitanih tacaka */

```

```

127     n = i;
129
130     /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
131     "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
132     (karakter -), a karakter argv[1][1] odreduje kriterijum
133     sortiranja */
134     switch (argv[1][1]) {
135         case 'x':
136             /* Sortiranje po vrednosti x koordinate */
137             sortiraj_po_x(tacke, n);
138             break;
139         case 'y':
140             /* Sortiranje po vrednosti y koordinate */
141             sortiraj_po_y(tacke, n);
142             break;
143         case 'o':
144             /* Sortiranje po udaljenosti od koordinatnog pocetka */
145             sortiraj_po_rastojanju(tacke, n);
146             break;
147     }
148
149     /* Upisivanje dobijenog niza u izlaznu datoteku */
150     for (i = 0; i < n; i++) {
151         fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
152     }
153
154     /* Zatvaranje otvorenih datoteka */
155     fclose(ulaz);
156     fclose(izlaz);
157 }
```

### Rešenje 3.20

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 1000
6 #define MAX_DUZINA 16
7
8     /* Struktura koja reprezentuje jednog gradjanina */
9     typedef struct gr {
10         char ime[MAX_DUZINA];
11         char prezime[MAX_DUZINA];
12     } Gradjanin;
13
14     /* Funkcija sortira niz gradjana rastuce po imenima */
15     void sort_ime(Gradjanin a[], int n)
16     {
```

```

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

```

```

69         return i;
71     }
73
73     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]))
123       isti_rbr++;
124
125   /* Alternativno (efikasnije) resenje */
126   //*****
127   for(i=0; i<n ;i++)
128     if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
129         strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
130       isti_rbr++;
131   //*****
132
133   /* Ispis rezultata */
134   printf("%d\n", isti_rbr);
135
136   exit(EXIT_SUCCESS);
137 }
```

### Rešenje 3.22

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 #define MAX_BR_RECI 128
6 #define MAX_DUZINA_RECI 32
7
8 /* Funkcija koja izracunava broj suglasnika u reci */
9 int broj_suglasnika(char s[])
10 {
11   char c;
12   int i;
13   int suglasnici = 0;
14   /* 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
duzini reci se mora proslediti zbog pravilnog upravljanja
```

```

31     memorijom */
32 void sortiraj_reci(char reci[][MAX_DUZINA_REC], int n)
33 {
34     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35         duzina_j, duzina_min;
36     char tmp[MAX_DUZINA_REC];
37     for (i = 0; i < n - 1; i++) {
38         min = i;
39         for (j = i; j < n; j++) {
40             /* Prvo se uporedjuje 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 uporedjuju
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                  uporedjuju 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 označava broj reci,
74      a drugi maksimalnu duzinu pojedinačne reci */
75     char reci[MAX_BR_REC][MAX_DUZINA_REC];
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     }
83     return 0;

```

```

83    }

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

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

101   /* Ispis sortiranog niza reci */
102   for (i = 0; i < n; i++) {
103       printf("%s ", reci[i]);
104   }
105   printf("\n");

107   /* Zatvaranje datoteke */
108   fclose(ulaz);
109
110   return 0;
111 }
```

**Rešenje 3.23**

```

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

#define MAX_ARTIKALA 100000

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

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

```

20 int desni = n - 1;

22 /* Dokle god je indeks levi levo od indeksa desni */
24 while (levi <= desni) {
25     /* Racuna se sredisnji indeks */
26     int srednji = (levi + desni) / 2;
27     /* Ako je sredisnji element veci od trazenog, tada se trazeni
28      mora nalaziti u levoj polovini niza */
29     if (x < a[srednji].kod)
30         desni = srednji - 1;
31     /* Ako je sredisnji element manji od trazenog, tada se trazeni
32      mora nalaziti u desnoj polovini niza */
33     else if (x > a[srednji].kod)
34         levi = srednji + 1;
35     else
36         /* Ako je sredisnji element jednak trazenom, tada je artikal sa
37          bar kodom x pronadjen na poziciji srednji */
38         return srednji;
39     }
40     /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
41     return -1;
42 }

43 /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44 void selection_sort(Artikal a[], int n)
45 {
46     int i, j;
47     int min;
48     Artikal pom;

49     for (i = 0; i < n - 1; i++) {
50         min = i;
51         for (j = i + 1; j < n; j++)
52             if (a[j].kod < a[min].kod)
53                 min = j;
54         if (min != i) {
55             pom = a[i];
56             a[i] = a[min];
57             a[min] = pom;
58         }
59     }
60 }

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

67     FILE *fp = NULL;

```

```

72  /* Otvaranje datoteke */
73  if ((fp = fopen("artikli.txt", "r")) == NULL) {
74      fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
75      exit(EXIT_FAILURE);
76  }

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

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

86  n = i;

87

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

103

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

112

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

```

```

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

```

### Rešenje 3.24

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 500
6
7 /* Struktura sa svim informacijama o pojedinacnom studentu */
8 typedef struct {
9     char ime[21];
10    char prezime[26];
11    int prisustvo;
12    int zadaci;
13 } Student;
14
15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
16   rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
18 {
19     int i, j;
20     int min;
21
22     for (i = 0; i < n - 1; i++) {
23         min = i;
24         for (j = i + 1; j < n; j++) {
25             if (strcmp(niz[j].ime, niz[min].ime) < 0) {
26                 min = j;
27             }
28         }
29         if (min != i) {
30             Student temp = niz[i];
31             niz[i] = niz[min];
32             niz[min] = temp;
33         }
34     }
35 }

```

```

21     Student pom;
23
24     for (i = 0; i < n - 1; i++) {
25         min = i;
26         for (j = i + 1; j < n; j++)
27             if (strcmp(niz[j].ime, niz[min].ime) < 0)
28                 min = j;
29
30         if (min != i) {
31             pom = niz[min];
32             niz[min] = niz[i];
33             niz[i] = pom;
34         }
35     }
36
37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
38    zadataka opadajuce, a ukoliko neki studenti imaju isti broj
39    uradjenih zadataka sortiraju se po duzini imena rastuce. */
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 opadajuce. Ukoliko neki studenti imaju isti broj casova,
63    sortiraju se opadajuce po broju uradjenih zadataka, a ukoliko se
64    i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65    prezimenu opadajuce. */
66 void sort_prisustvo_pa_zadatke_pa_prezimena(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, "Neunesno otvaranje datoteke aktivnost.txt.\n");
101         exit(EXIT_FAILURE);
102     }
103
104     /* Ucitavanje sadrzaja */
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, "Neunesno otvaranje datoteke dat1.txt.\n");
119         exit(EXIT_FAILURE);
120     }
121     /* Upis niza u datoteku */
122     fprintf
123         (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
124     for (i = 0; i < br_studenata; i++)
125         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,

```

```

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

```

### Rešenje 3.25

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define KORAK 10
6

```

```

8  /* Struktura koja opisuje jednu pesmu */
9  typedef struct {
10    char *izvodjac;
11    char *naslov;
12    int broj_gledanja;
13 } Pesma;
14
15 /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
16   rad qsort funkcije) */
17 int uporedi_gledanost(const void *pp1, const void *pp2)
18 {
19   Pesma *p1 = (Pesma *) pp1;
20   Pesma *p2 = (Pesma *) pp2;
21
22   return p2->broj_gledanja - p1->broj_gledanja;
23 }
24
25 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
26   funkcije) */
27 int uporedi_naslove(const void *pp1, const void *pp2)
28 {
29   Pesma *p1 = (Pesma *) pp1;
30   Pesma *p2 = (Pesma *) pp2;
31
32   return strcmp(p1->naslov, p2->naslov);
33 }
34
35 /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
36   qsort funkcije) */
37 int uporedi_izvodjace(const void *pp1, const void *pp2)
38 {
39   Pesma *p1 = (Pesma *) pp1;
40   Pesma *p2 = (Pesma *) pp2;
41
42   return strcmp(p1->izvodjac, p2->izvodjac);
43 }
44
45 int main(int argc, char *argv[])
46 {
47   FILE *ulaz;                      /* Pokazivac na deo memorije za
48   Pesma *pesme;                   cuvanje pesama */
49   int alocirano_za_pesme;          /* Broj mesta alociranih za pesme */
50   int i;                           /* Redni broj pesme cije se
51                                     informacije citaju */
52   int n;                           /* Ukupan broj pesama */
53   int j, k;
54   char c;
55   int alocirano;                  /* Broj mesta alociranih za proratne
56                                     informacije o pesmama */
57   int broj_gledanja;
58 }
```

```

13  /* Priprema datoteke za citanje */
14  ulaz = fopen("pesme_bez_prestavki.txt", "r");
15  if (ulaz == NULL) {
16      fprintf(stderr, "Greska pri otvaranju ulazne datoteke!\n");
17      return 0;
18  }

20  /* Citanje informacija o pesmama */
21  pesme = NULL;
22  alocirano_za_pesme = 0;
23  i = 0;

25  while (1) {

26      /* Proverava da li je dostignut kraj datoteke */
27      c = fgetc(ulaz);
28      if (c == EOF) {
29          /* Nema vise sadrzaja za citanje */
30          break;
31      } else {
32          /* Inace, vracamo procitani karakter nazad */
33          ungetc(c, ulaz);
34      }

35      /* Provera da li postoji dovoljno memorije za citanje nove pesme */
36      /*
37      if (alocirano_za_pesme == i) {

38          /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
39             novih KORAK mesta */
40          alocirano_za_pesme += KORAK;
41          pesme =
42              (Pesma *) realloc(pesme,
43                                alocirano_za_pesme * sizeof(Pesma));

44          /* Proverava da li je nova memorija uspesno realocirana */
45          if (pesme == NULL) {
46              /* Ako nije ispisuje se obavestenje */
47              fprintf(stderr, "Problem sa alokacijom memorije!\n");
48              /* I oslolobadja sva memorija zauzeta do ovog koraka */
49              for (k = 0; k < i; k++) {
50                  free(pesme[k].izvodjac);
51                  free(pesme[k].naslov);
52              }
53              free(pesme);
54              return 0;
55          }
56      }

57      /* Ako jeste, nastavlja se sa citanjem pesama ... */
58      /* Cita se ime izvodjaca */
59      j = 0;                               /* Pozicija na koju treba smestiti

```

```

110                      procitani karakter */
111      alocirano = 0;           /* Broj alociranih mesta */
112      pesme[i].izvodjac = NULL; /* Memorija za smestanje procitanih
113                                karaktera */

114      /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
115       izvodjaca) citaju se karakteri iz datoteke */
116      while ((c = fgetc(ulaz)) != '\n') {
117          /* Proverav da li postoji dovoljno memorije za smestanje
118             procitanog karaktera */
119          if (j == alocirano) {

120              /* Ako ne, ako je potrosena sva alocirana memorija, alocira
121               se novih KORAK mesta */
122              alocirano += KORAK;
123              pesme[i].izvodjac =
124                  (char *) realloc(pesme[i].izvodjac,
125                                  alocirano * sizeof(char));

126              /* Provera da li je nova alokacija uspesna */
127              if (pesme[i].izvodjac == NULL) {
128                  /* Ako nije oslobadja se sva memorija zauzeta do ovog
129                   koraka */
130                  for (k = 0; k < i; k++) {
131                      free(pesme[k].izvodjac);
132                      free(pesme[k].naslov);
133                  }
134                  free(pesme);
135                  /* I prekida sa izvrsavanjem programa */
136                  return 0;
137              }

138          }

139      }

140      /* Ako postoji dovoljno memorije, smestamo procitani karakter
141       */
142      pesme[i].izvodjac[j] = c;
143      j++;
144      /* I nastavlja se sa citanjem */
145  }

146      /* Upis terminirajuce nule na kraj reci */
147      pesme[i].izvodjac[j] = '\0';

148      /* Preskace se karakter - */
149      fgetc(ulaz);

150      /* Preskace se razmak */
151      fgetc(ulaz);

152      /* Cita se naslov pesme */
153      j = 0;                      /* Pozicija na koju treba smestiti
154                                procitani karakter */
155

```

```

162     alocirano = 0;           /* Broj alociranih mesta */
163     pesme[i].naslov = NULL; /* Memorija za smestanje procitanih
164                               karaktera */

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

204     /* Cita se broj gledanja */
205     broj_gledanja = 0;

206     /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
207      datoteke */
208     while ((c = fgetc(ulaz)) != '\n') {
209         broj_gledanja = broj_gledanja * 10 + (c - '0');
210     }

```

```

212     pesme[i].broj_gledanja = broj_gledanja;
214
215     /* Prelazi se na citanje sledeće pesme */
216     i++;
217 }
218
219 /* Informacija o broju procitanih pesama */
220 n = i;
221 /* Zatvaranje nepotrebne datoteke */
222 fclose(ulaz);
223
224 /* Analiza argumenta komandne linije */
225 if (argc == 1) {
226     /* Nema dodatnih opcija => sortiranje po broju gledanja */
227     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
228 } else {
229     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
230         /* Sortiranje po naslovu */
231         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
232     } else {
233         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
234             /* Sortiranje po izvodjacu */
235             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
236         } else {
237             fprintf(stderr, "Nedozvoljeni argumenti!\n");
238             free(pesme);
239             return 0;
240         }
241     }
242
243     /* Ispis rezultata */
244     for (i = 0; i < n; i++) {
245         printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
246                pesme[i].broj_gledanja);
247     }
248
249     /* Oslobadjanje memorije */
250     for (i = 0; i < n; i++) {
251         free(pesme[i].izvodjac);
252         free(pesme[i].naslov);
253     }
254     free(pesme);
255
256     return 0;
257 }
```

### Rešenje 3.28

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

```

4   #include "matrica.h"
5
6   /* Funkcija koja određuje zbir v-te vrste matrice a koja ima m
7    * kolona */
8   int zbir_vrste(int **a, int v, int m)
9   {
10    int i, zbir = 0;
11
12    for (i = 0; i < m; i++) {
13      zbir += a[v][i];
14    }
15    return zbir;
16}
17
18 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
19  osnovu zbirra koriscenjem selection sort algoritma */
20 void sortiraj_vrste(int **a, int n, int m)
21 {
22    int i, j, min;
23
24    for (i = 0; i < n - 1; i++) {
25      min = i;
26      for (j = i + 1; j < n; j++) {
27        if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
28          min = j;
29        }
30      }
31      if (min != i) {
32        int *tmp;
33        tmp = a[i];
34        a[i] = a[min];
35        a[min] = tmp;
36      }
37    }
38
39    int main(int argc, char *argv[])
40    {
41      int **a;
42      int n, m;
43
44      /* Unos dimenzija matrice */
45      printf("Unesite dimenzije matrice: ");
46      scanf("%d %d", &n, &m);
47
48      /* Alokacija memorije */
49      a = alociraj_matricu(n, m);
50      if (a == NULL) {
51        fprintf(stderr, "Neuspesna alokacija matrice\n");
52        exit(EXIT_FAILURE);
53      }
54

```

```

56    /* Ucitavanje elementa matrice */
57    printf("Unesite elemente matrice po vrstama:\n");
58    ucitaj_matricu(a, n, m);
59
60    /* Poziv funkcije koja sortira vrste matrice prema zbiru */
61    sortiraj_vrste(a, n, m);
62
63    /* Ispis rezultujuće matrice */
64    printf("Sortirana matrica je:\n");
65    ispisi_matricu(a, n, m);
66
67    /* Oslobođjanje memorije */
68    a = dealociraj_matricu(a, n);
69
70    return 0;
}

```

### Rešenje 3.31

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <search.h>
5
6 #define MAX 100
7
8 /* Funkcija poredjenja dva cela broja */
9 int poredi_int(const void *a, const void *b)
10 {
11     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
12      se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
13
14     /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
15      oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */
16
17     int b1 = *((int *) a);
18     int b2 = *((int *) b);
19
20     /* return b1 - b2; */
21     if (b1 > b2)
22         return 1;
23     else if (b1 < b2)
24         return -1;
25     else
26         return 0;
27 }
28
29 int poredi_int_nerastuce(const void *a, const void *b)
30 {
31     /* Za obrnuti poredak treba samo oduzimati a od b */
32     /* return *((int *)b) - *((int *)a); */

```

```

33  /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
35   funkcija */
36   return -poredi_int(a, b);
37 }

38 int main()
39 {
40   size_t n;
41   int i, x;
42   int a[MAX], *p = NULL;

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

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

52   /* Sortiranje niza celih brojeva */
53   qsort(a, n, sizeof(int), &poredi_int);

54   /* Prikaz sortiranog niza */
55   printf("Sortirani niz u rastucem poretku:\n");
56   for (i = 0; i < n; i++)
57     printf("%d ", a[i]);
58   putchar('\n');

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

63   /* Binarna pretraga */
64   printf("Binarna pretraga: \n");
65   p = bsearch(&x, a, n, sizeof(int), &poredi_int);
66   if (p == NULL)
67     printf("Elementa nema u nizu!\n");
68   else
69     printf("Element je nadjen na poziciji %ld\n", p - a);

70   /* Linearna pretraga */
71   printf("Linearna pretraga (lfind): \n");
72   p = lfind(&x, a, &n, sizeof(int), &poredi_int);
73   if (p == NULL)
74     printf("Elementa nema u nizu!\n");
75   else
76     printf("Element je nadjen na poziciji %ld\n", p - a);

```

```

85     return 0;
87 }
```

### Rešenje 3.32

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <search.h>
5
6 #define MAX 100
7
8 /* Funkcija racuna broj delilaca broja x */
9 int broj_deli_laca(int x)
10 {
11     int i;
12     int br;
13
14     /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
15     if (x < 0)
16         x = -x;
17     if (x == 0)
18         return 0;
19     if (x == 1)
20         return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22     br = 2;
23     for (i = 2; i < sqrt(x); i++)
24         if (x % i == 0)
25             /* Ako i deli x onda su delioci: i, x/i */
26             br += 2;
27     /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
28      promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29      jos jednog delioca */
30     if (i * i == x)
31         br++;
32
33     return br;
34 }
35
36 /* Funkcija poredjenja dva cela broja po broju delilaca */
37 int poredi_po_broju_deli_laca(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = broj_deli_laca(ak);
42     int n_d_b = broj_deli_laca(bk);
43
44     return n_d_a - n_d_b;
45 }
```

```

47 int main()
48 {
49     size_t n;
50     int i;
51     int a[MAX];
52
53     /* Unos dimenzije */
54     printf("Uneti dimenziju niza: ");
55     scanf("%ld", &n);
56     if (n > MAX)
57         n = MAX;
58
59     /* Unos elemenata niza */
60     printf("Uneti elemente niza:\n");
61     for (i = 0; i < n; i++)
62         scanf("%d", &a[i]);
63
64     /* Sortiranje niza celih brojeva prema broju delilaca */
65     qsort(a, n, sizeof(int), &poredi_po_broju_delilaca);
66
67     /* Prikaz sortiranog niza */
68     printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
69     for (i = 0; i < n; i++)
70         printf("%d ", a[i]);
71     putchar('\n');
72
73     return 0;
74 }
```

### Rešenje 3.33

```

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 31
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]
```

```

    i ona je tipa char*.

21
23 Kako pokazivaci a i b u sledecoj funkciji sadrže adrese elemenata
25 koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
27 reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
29 slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
31 nad njima.
32 ****
33 int poredi_leksikografski(const void *a, const void *b)
34 {
35     return strcmp((char *) a, (char *) b);
36 }

37 /* Funkcija sличna prethodnoj, osim sto elemente ne uporedjuje
38 leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
40 {
41     return strlen((char *) a) - strlen((char *) b);
42 }

43 int main()
44 {
45     int i;
46     size_t n;
47     FILE *fp = NULL;
48     char niske[MAX_NISKI][MAX_DUZINA];
49     char *p = NULL;
50     char x[MAX_DUZINA];

51     /* Otvaranje datoteke */
52     if ((fp = fopen("niske.txt", "r")) == NULL) {
53         fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
54         exit(EXIT_FAILURE);
55     }

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

58     /* Zatvaranje datoteke */
59     fclose(fp);
60     n = i;

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

65     printf("Leksikografski sortirane niske:\n");
66     for (i = 0; i < n; i++)
67         printf("%s ", niske[i]);
68     printf("\n");
69 }

70
71

```

```

13  /* Unos trazene niske */
73  printf("Uneti trazenu nisku: ");
    scanf("%s", x);
75
77  /* Binarna pretraga */
78  /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
   je niz vec sortiran leksikografski. */
79  p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
              &poredi_leksikografski);
81
82  if (p != NULL)
83      printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
             p, (p - (char *) niske) / MAX_DUZINA);
84  else
85      printf("Niska nije pronadjena u nizu\n");
86
87  /* Sortiranje po duzini */
88  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
89
90  printf("Niske sortirane po duzini:\n");
91  for (i = 0; i < n; i++)
92      printf("%s ", niske[i]);
93  printf("\n");
94
95  /* Linearna pretraga */
96  p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
             &poredi_leksikografski);
97
98  if (p != NULL)
99      printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
             p, (p - (char *) niske) / MAX_DUZINA);
100 else
101     printf("Niska nije pronadjena u nizu\n");
102
103 exit(EXIT_SUCCESS);
104 }
105 }
```

### Rešenje 3.34

```

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 31
8
9 *****
10 Niz pokazivaca na karaktere ovog potpisa
11 char *niske[3];
posle alokacije u main-u se moze graficki predstaviti ovako:
```

```

13   -----      -----
15   | X | -----> | a | b | c | \0|
16   -----      =====
17   | Y | -----> | d | e | \0|
18   -----      =====
19   | Z | -----> | f | g | h | \0|
20   -----      -----
21   Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
22   njegov element pokazivac koji pokazuje na alocirane karaktere i-te
23   reci. Njegov tip je char*.
24
25   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
26   koji trebaju biti uporedjeni (recimo adresu od X i adresu od Z), i
27   kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
28   moraju i kastovati. Da bi se leksikografski uporedili elementi X i
29   Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
30   u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
31   kastovani na odgovarajuci tip.
32 ****
33 int poredi_leksikografski(const void *a, const void *b)
34 {
35     return strcmp(*((char **) a, *((char **) b));
36 }
37 /* Funkcija sличna prethodnoj, osim sto elemente ne uporedjuje
38    leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
40 {
41     return strlen(*((char **) a) - strlen(*((char **) b);
42 }
43 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
44    element u nizu sa kojim se poredi, pa njega treba kastovati na
45    char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
46    ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
47    main funkciji je to x, koji je tipa char*, tako da pokazivac a
48    ovde samo treba kastovati i ne dereferencirati. */
49 int poredi_leksikografski_b(const void *a, const void *b)
50 {
51     return strcmp((char *) a, *((char **) b);
52 }
53
54 int main()
55 {
56     int i;
57     size_t n;
58     FILE *fp = NULL;
59     char *niske[MAX_NISKI];
60     char **p = NULL;
61     char x[MAX_DUZINA];
62
63     /* Otvaranje datoteke */

```

```

65  if ((fp = fopen("niske.txt", "r")) == NULL) {
66      fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
67      exit(EXIT_FAILURE);
68  }
69
70  /* Citanje sadrzaja datoteke */
71  i = 0;
72  while (fscanf(fp, "%s", x) != EOF) {
73      /* Alociranje dovoljne memorije za i-tu nisku */
74      if ((niske[i] = malloc((strlen(x) + 1) * sizeof(char))) == NULL)
75      {
76          fprintf(stderr, "Greska pri alociranju niske\n");
77          exit(EXIT_FAILURE);
78      }
79      /* Kopiranje procitane niske na svoje mesto */
80      strcpy(niske[i], x);
81      i++;
82  }
83
84  /* Zatvaranje datoteke */
85  fclose(fp);
86  n = i;
87
88  /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
89  prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
90  niske po duzini */
91  qsort(niske, n, sizeof(char *), &poredi_leksikografski);
92
93  printf("Leksikografski sortirane niske:\n");
94  for (i = 0; i < n; i++)
95      printf("%s ", niske[i]);
96  printf("\n");
97
98  /* Unos trazene niske */
99  printf("Uneti trazenu nisku: ");
100  scanf("%s", x);
101
102  /* Binarna pretraga */
103  p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
104  if (p != NULL)
105      printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
106             *p, p - niske);
107  else
108      printf("Niska nije pronadjena u nizu\n");
109
110  /* Linearna pretraga */
111  p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
112  if (p != NULL)
113      printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
114             *p, p - niske);
115  else
116      printf("Niska nije pronadjena u nizu\n");

```

```

117  /* Sortiramo po duzini */
118  qsort(niske, n, sizeof(char *), &poredi_duzine);
119
120  printf("Niske sortirane po duzini:\n");
121  for (i = 0; i < n; i++)
122      printf("%s ", niske[i]);
123  printf("\n");
124
125  /* Oslobođanje zauzete memorije */
126  for (i = 0; i < n; i++)
127      free(niske[i]);
128
129  exit(EXIT_SUCCESS);
}

```

### Rešenje 3.35

```

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

```

```

34     Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
35     parametar je element niza ciji se bodovi porede. */
36 int poredi2(const void *a, const void *b)
37 {
38     int bodovi = *(int *) a;
39     Student *s = (Student *) b;
40     return s->bodovi - bodovi;
41 }
42
43 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
44    Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
45    parametar je element niza cije se prezime poredi. */
46 int poredi3(const void *a, const void *b)
47 {
48     char *prezime = (char *) a;
49     Student *s = (Student *) b;
50     return strcmp(prezime, s->prezime);
51 }
52
53 int main(int argc, char *argv[])
54 {
55     Student kolokvijum[MAX];
56     int i;
57     size_t br_studenata = 0;
58     Student *nadjen = NULL;
59     FILE *fp = NULL;
60     int bodovi;
61     char prezime[21];
62
63     /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64        informacija korisniku kako se program koristi i prekida se
65        izvrsavanje. */
66     if (argc < 2) {
67         fprintf(stderr,
68                 "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
69                 argv[0]);
70         exit(EXIT_FAILURE);
71     }
72
73     /* Otvaranje datoteke */
74     if ((fp = fopen(argv[1], "r")) == NULL) {
75         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
76         exit(EXIT_FAILURE);
77     }
78
79     /* Ucitavanje sadrzaja */
80     for (i = 0;
81          fscanf(fp, "%s%s%d", kolokvijum[i].ime,
82                  kolokvijum[i].prezime,
83                  &kolokvijum[i].bodovi) != EOF; i++);
84
85     /* Zatvaranje datoteke */

```

```

86     fclose(fp);
87     br_studenata = i;
88
89     /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
90      studenata sa istim brojem bodova sortiranje vrsti po prezimenu */
91     qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
92
93     printf("Studenti sortirani po broju poena opadajuce, ");
94     printf("pa po prezimenu rastuce:\n");
95     for (i = 0; i < br_studenata; i++)
96         printf("%s %s %d\n", kolokvijum[i].ime,
97                kolokvijum[i].prezime, kolokvijum[i].bodovi);
98
99     /* Pretrazivanje studenata po broju bodova se vrsti binarnom
100       pretragom jer je niz sortiran po broju bodova. */
101    printf("Unesite broj bodova: ");
102    scanf("%d", &bodovi);
103
104    nadjen =
105        bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
106                &poredi2);
107
108    if (nadjen != NULL)
109        printf
110            ("Pronadjeni je student sa unetim brojem bodova: %s %s %d\n",
111             nadjen->ime, nadjen->prezime, nadjen->bodovi);
112    else
113        printf("Nema studenta sa unetim brojem bodova\n");
114
115    /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116       sortiran po bodovima. */
117    printf("Unesite prezime: ");
118    scanf("%s", prezime);
119
120    nadjen =
121        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
122              &poredi3);
123
124    if (nadjen != NULL)
125        printf
126            ("Pronadjeni je student sa unetim prezimenom: %s %s %d\n",
127             nadjen->ime, nadjen->prezime, nadjen->bodovi);
128    else
129        printf("Nema studenta sa unetim prezimenom\n");
130
131    exit(EXIT_SUCCESS);
132 }

```

## Rešenje 3.36

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 128
6
7 /* Funkcija poređi dva karaktera */
8 int uporedi_char(const void *pa, const void *pb)
9 {
10    return *(char *) pa - *(char *) pb;
11 }
12
13 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14 int anagrami(char s[], char t[])
15 {
16    /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
17    if (strlen(s) != strlen(t))
18        return 0;
19
20    /* Sortiranje niski */
21    qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22    qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);
23
24    /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
25       suprotnom, nisu */
26    return !strcmp(s, t);
27 }
28
29 int main()
30 {
31    char s[MAX], t[MAX];
32
33    /* Unos niski */
34    printf("Unesite prvu nisku: ");
35    scanf("%s", s);
36    printf("Unesite drugu nisku: ");
37    scanf("%s", t);
38
39    /* Ispituje se da li su niske anagrami */
40    if (anagrami(s, t))
41        printf("jesu\n");
42    else
43        printf("nisu\n");
44
45    return 0;
46 }

```

**Rešenje 3.37**

```

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

```

```

3 #include <stdlib.h>
5
7 /* Funkcija porenenjenja */
9 int uporedi_niske(const void *pa, const void *pb)
11 {
12     return strcmp((char *) pa, (char *) pb);
13 }
15
16 int main()
17 {
18     int i, n;
19     char S[MAX] [MAX_DUZINA];
20
21     /* Unos broja niski */
22     printf("Unesite broj niski:");
23     scanf("%d", &n);
24
25     /* Unos niza niski */
26     printf("Unesite niske:\n");
27     for (i = 0; i < n; i++)
28         scanf("%s", S[i]);
29
30     /* Sortiranje niza niski */
31     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
32
33     /***** Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis *****
34     sortiranih niski. Koriscen je samo u fazi testiranja programa.
35
36     printf("Sortirane niske su:\n");
37     for(i = 0; i < n; i++)
38         printf("%s ", S[i]);
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 3.38**

```

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 poređi studente prema broju poena, rastuce */
16 int uporedi_poeni(const void *a, const void *b)
17 {
18     Student s = *(Student *) a;
19     Student t = *(Student *) b;
20     return s.poeni - t.poeni;
21 }
22
23 /* Funkcija poređi studente prvo prema godini, zatim prema smeru i na
24 kraju prema indeksu */
25 int uporedi_nalog(const void *a, const void *b)
26 {
27     Student s = *(Student *) a;
28     Student t = *(Student *) b;
29     /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
30     broj indeksa */
31     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
32     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
33     char smer1 = s.nalog[1];
34     char smer2 = t.nalog[1];
35     int indeks1 =
36         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
37         s.nalog[6] - '0';
38     int indeks2 =
39         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
40         t.nalog[6] - '0';
41     if (godina1 != godina2)
42         return godina1 - godina2;
43     else if (smer1 != smer2)
44         return smer1 - smer2;
45     else
46         return indeks1 - indeks2;
47 }
48
49 int uporedi_bsearch(const void *a, const void *b)
50 {
51     /* Nalog studenta koji se trazi */
52 }

```

```

53     char *nalog = (char *) a;
54     /* Kljuc pretrage */
55     Student s = *(Student *) b;
56
56     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
57     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
58     char smer1 = nalog[1];
59     char smer2 = s.nalog[1];
60     int indeks1 =
61         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
62         ;
63     int indeks2 =
64         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
65         s.nalog[6] - '0';
66     if (godina1 != godina2)
67         return godina1 - godina2;
68     else if (smer1 != smer2)
69         return smer1 - smer2;
70     else
71         return indeks1 - indeks2;
72 }
73
73 int main(int argc, char **argv)
74 {
75     Student *nadjen = NULL;
76     char nalog_trazeni[8];
77     Student niz_studenata[100];
78     int i = 0, br_studenata = 0;
79     FILE *in = NULL, *out = NULL;
80
81     /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
82      korisnik nije ispravno pozvao program i prijavljuje se greska.
83      */
84     if (argc != 2 && argc != 3) {
85         fprintf(stderr,
86                 "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
87         );
88         exit(EXIT_FAILURE);
89     }
90
91     /* Otvaranje datoteke za citanje */
92     in = fopen("studenti.txt", "r");
93     if (in == NULL) {
94         fprintf(stderr,
95                 "Greska prilikom otvaranja datoteke studenti.txt!\n");
96         exit(EXIT_FAILURE);
97     }
98
99     /* Otvaranje datoteke za pisanje */
100    out = fopen("izlaz.txt", "w");
101    if (out == NULL) {
102        fprintf(stderr,
103

```

```

101         "Greska prilikom otvaranja datoteke izlaz.txt!\n");
103     exit(EXIT_FAILURE);
105 }
106 /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
107 while (fscanf(
108     (in, "%s %s %s %d", niz_studenata[i].nalog,
109     niz_studenata[i].ime, niz_studenata[i].prezime,
110     &niz_studenata[i].poeni) != EOF)
111     i++;
112
113 br_studenata = i;
114
115 /* Ako je prisutna opcija -p, vrsti se sortiranje po poenima */
116 if (strcmp(argv[1], "-p") == 0)
117     qsort(niz_studenata, br_studenata, sizeof(Student),
118           &uporedi_poeni);
119 /* Ako je prisutna opcija -n, vrsti se sortiranje po nalogu */
120 else if (strcmp(argv[1], "-n") == 0)
121     qsort(niz_studenata, br_studenata, sizeof(Student),
122           &uporedi_nalog);
123
124 /* Sortirani studenti se ispisuju u izlaznu datoteku */
125 for (i = 0; i < br_studenata; i++)
126     fprintf(out, "%s %s %s %d\n",
127             niz_studenata[i].nalog,
128             niz_studenata[i].ime, niz_studenata[i].prezime,
129             niz_studenata[i].poeni);
130
131 /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
132    studenta... */
133 if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
134     strcpy(nalog_trazeni, argv[2]);
135
136     /* ... pronalazi se student sa tim nalogom... */
137     nadjen =
138         (Student *) bsearch(nalog_trazeni, niz_studenata,
139                             br_studenata, sizeof(Student),
140                             &uporedi_bsearch);
141
142     if (nadjen == NULL)
143         printf("Nije nadjen!\n");
144     else
145         printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
146               nadjen->prezime, nadjen->poeni);
147 }
148
149 /* Zatvaranje datoteka */
150 fclose(in);
151 fclose(out);
152
153 exit(EXIT_SUCCESS);
154 }
```

## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

**Zadatak 4.1** Napisati biblioteku za rad sa jednostruko povezanim listom čiji čvorovi sadrže cele brojeve.

- (a) Definisati strukturu `Cvor` kojom se predstavlja čvor liste. Čvor treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.
- (b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje mu polja i vraća njegovu adresu.
- (c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.
- (d) Napisati funkciju `Cvor *pronadji_poslednji(Cvor * glava)` koja pronalazi poslednji čvor u listi.
- (e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste.
- (f) Napisati funkciju `Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.

- (g) Napisati funkciju `int dodaj_iza(Cvor * tekuci, int broj)` koja iza čvora `tekuci` dodaje novi čvor sa vrednošću `broj`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene zagradama `[, ]` i međusobno razdvojene zapetama.
- (j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija 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 prepostavlja 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 prepostavlja 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]

Unesite 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]

Unesite 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]

Unesite 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]

Unesite broj koji se brise: 3
Lista nakon brisanja: [23, 14, 35]
```

- (3) U 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. Etikete 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 href='http://www.google.com'> Link 1</a>
<a href='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 za gresku:
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:** *Prepostaviti 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.

- (a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.
- (b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi 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

IZLAC:
REZULTAT.TXT
12 15 16
```

*Test 2*

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAC ZA GREŠKU:
Greska prilikom otvaranja
datoteke brojevi.txt.
```

*Test 3*

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAC:
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

IZLAC:
[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.

IZLAC ZA GREŠKU:
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

IZLAC:
[5, 6, 11, 12, 14, 16]
```

*Test 4*

```
Poziv: ./a.out dat1.txt
IZLAC ZA GREŠKU:
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

IZLAC:
2 5 4 6 6 11 10 12 15 14 16
```

**Zadatak 4.8** Sadržaj datoteke je aritmetički izraz koji može sadržati zgrade {}, [ i ]. Napisati program koji učitava sadržaj datoteke **izraz.txt** i korišćenjem

steka utvrđuje da li su zgrade u aritmetičkom izrazu dobro uparene. Program stampa odgovarajuću poruku na standardni izlaz.

*Test 1*

```
IZRAZ.TXT
{[23 + 5344] * (24 - 234)} - 23
IZLIZ:
Zgrade su ispravno uparene.
```

*Test 2*

```
IZRAZ.TXT
{[23 + 5] * (9 * 2)} - {23}
IZLIZ:
Zgrade su ispravno uparene.
```

*Test 3*

```
IZRAZ.TXT
{[2 + 54] / (24 * 87)} + (234 + 23)
IZLIZ:
Zgrade nisu ispravno uparene.
```

*Test 4*

```
IZRAZ.TXT
{(2 - 14) / (23 + 11)} * (2 + 13)
IZLIZ:
Zgrade nisu ispravno uparene.
```

*Test 5*

```
DATOTEKA IZRAZ.TXT JE PRAZNA
IZLIZ:
Zgrade su ispravno uparene.
```

*Test 6*

```
DATOTEKA IZRAZ.TXT NE POSTOJI.
IZLIZ ZA GREŠKU:
Greska prilikom otvaranja
datoteke izraz.txt!
```

[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.

*Test 1*

```
PONIV: ./a.out datoteka.html
DATOTEKA.HTML
<html>
<head>
<title>Primer</title>
</head>
<body>
</body>
IZLIZ:
Etikete nisu pravilno uparene
(etiketa <html> nije zatvorena)
```

*Test 2*

```
PONIV: ./a.out datoteka.html
DATOTEKA.HTML
<html>
<head>
<title>Primer</title>
</head>
<body>
</body>
</html>
IZLIZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html>
koja nije otvorena)
```

*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>

IZLAC:
Etikete su pravilno uparene!
```

*Test 4*

```
POZIV: ./a.out datoteka.html
DATOTEKA.HTML
<html>
<head>
<title>Primer</title>
</head>
<body>
</html>

IZLAC:
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.

IZLAC ZA GREŠKU:
Greska prilikom otvaranja
datoteke datoteka.html.
```

*Test 6*

```
POZIV: ./a.out datoteka.html
DATOTEKA.HTML JE PRAZNA

IZLAC:
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, nevezano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: Za čuvanje korisničkih zahteva koristiti red implementirani korišćenjem listi.

### Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Sluzbenik evidentira korisnicke zahteve:  
|| Novi zahtev [CTRL+D za kraj]  
|| JMBG: 1234567890123  
|| Opis problema: Divaranje 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 povezanim listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

- (a) Napisati funkciju `void obrisi_tekuci(Cvor ** adresa_glave, Cvor **`

`adresa_kraja, Cvor * tekuci`) koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave` i poslednji čvor liste na adresi `adresa_kraja`.

- (b) Napisati funkciju `void ispisi_listu_unazad(Cvor * kraj)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. Zbog efikasnog izvršavanja operacija dodavanja na kraj liste i ispisivanja liste unazad treba, pored pokazivača na glavu liste, čuvati i pokazivač na poslednji čvor liste. NAPOMENA: *Funkcije testirati koristeći test primere iz zadatka 4.1*

[Rešenje 4.11]

**Zadatak 4.12** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ . Plesači najpre formiraju krug tako da brojevi sa njihovih kostima rastu u smeru kazaljke na satu. Plesač sa brojem 1 stavlja levu ruku na rame plesača sa brojem 2, a desnu na svoj kuk i tako redom. Plesač sa brojem  $n$  svoju levu ruku spušta na rame plesača sa brojem 1, a desnu na svoj kuk i tako zatvara krug. Svoju plesnu tačku izvode tako što iz formiranog kruga najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug tako što  $k - 1$ -vi stavljaju ruku na rame  $k + 1$ -og i zatvara 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 poveznu kružnu listu.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<b>ULAZ:</b> 5 3	<b>ULAZ:</b> 8 4	<b>ULAZ:</b> 3 8
<b>IZLAZ:</b> 3 1 5 2 4	<b>IZLAZ:</b> 4 8 5 2 1 3 7 6	<b>IZLAZ ZA GREŠKU:</b> n mora biti uvek veće od k, a 3 < 8!

**Zadatak 4.13** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ . Plesači najpre formiraju krug tako da brojevi sa njihovih kostima rastu u smeru kazaljke na satu. Svaki plesač levu ruku stavlja na rame plesača sa sledećim većim brojem, a desnu na rame plesača sa prvim manjim brojem. Plesač sa brojem 1 stavlja levu ruku na rame plesača sa brojem 2, a desnu na rame plesača sa brojem  $n$ . Plesač sa brojem  $n$  svoju desnu ruku spušta na rame plesača sa brojem  $n - 1$ ,

a levu na rame plesača sa brojem 1 i tako zatvara krug. Plesači izvode svoju plesnu tačku tako što iz formiranog kruga najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalо 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</pre>	<pre>   ULAZ:   8 4</pre>	<pre>   ULAZ:   5 8</pre>
<pre>   IZLAZ:   3 5 4 2 1</pre>	<pre>   IZLAZ:   4 8 5 7 6 3 2 1</pre>	<pre>   IZLAZ ZA GREŠKU:   n mora biti uvek vece   od k, a 5 &lt; 8!</pre>

## 4.2 Stabla

**Zadatak 4.14** Napisati biblioteku za rad sa binarnim pretraživačkim stablima.

- Definisati strukturu **Cvor** kojom se opisuje čvor stabla, a koja sadrži ceo broj **broj** i pokazivače **levo** i **desno** redom na levo i desno podstablo.
- Napisati funkciju **Cvor \*napravi\_cvor(int broj)** koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem **broj**.
- Napisati funkciju **int dodaj\_u\_stablo(Cvor \*\* adresa\_korena, int broj)** koja u stablu na koje pokazuje argument **adresa\_korena** dodaje ceo broj **broj**. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.
- Napisati funkciju **Cvor \*pretrazi\_stablo(Cvor \* koren, int broj)** koja proverava da li se ceo broj **broj** nalazi u stablu sa korenom **koren**. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.
- Napisati funkciju **Cvor \*pronadji\_najmanji(Cvor \* koren)** koja pronađe čvor koji sadrži najmanju vrednost u stablu sa korenom **koren**.

- (f) Napisati funkciju `Cvor *pronadji_najveci(Cvor * koren)` koja pronađe čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor ** adresa_korena, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `adresa_korena`.
- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor * koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor * koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor * koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void osloboodi_stablo(Cvor ** adresa_korena)` koja oslobođa memoriju zauzetu stablom na koje pokazuje argument `adresa_korena`.

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

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRTL+D za kraj unosa):
7 2 1 9 32 18
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Trazi se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuće stablo: 1 2 9 18 32
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRTL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
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 pretraživackog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova.

Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku **Nedostaje ime ulazne datoteke!**. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

*Test 1*

```
Poziv: ./a.out test.txt
TEST.TXT
Sunce utorak raCunar SUNCE programiranje
jabuka PROGramiranje sunCE JABUka

IZLAC:
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

IZLAC:
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
IZLAC:
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 stabbala 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 IMENIKI.TXT NE POSTOJI
INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik1.txt
Greska: Neuspesno otvaranje datoteke
imenik1.txt.
```

[Rešenje 4.16]

**Zadatak 4.17** U datoteci *prijemni.txt* nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niska od najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

*Test 1*

```
PRIJEMNI.TXT
Marko Markovic 45.4 12.3 11
Milan Jevremovic 35.2 1.3 9
Maja Agic 60 19 20
Nadica Zec 54.2 10 15.8
Jovana Milic 23.3 2 5.6

IZLAC:
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
IZLAC:
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 pronađi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj unosa. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumi - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu **DD.MM.** Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

*Primer 1*

```
POZIV: ./a.out rodjendani.txt
RODJENDANI.TXT
Marko Markovic 12.12.
Milan Jevremovic 04.06.
Maja Agic 23.04.
Nadica Zec 01.01.
Jovana Milic 05.05.

INTERAKCIJA SA PROGRAMOM:
Unesite datum: 23.04.
Slavljenik: Maja Agic
Unesite datum: 01.01.
Slavljenik: Nadica Zec
Unesite datum: 01.05.
Slavljenik: Jovana Milic 05.05.
Unesite datum: 20.12.
Slavljenik: Nadica Zec 01.01.
Unesite datum:
```

*Primer 2*

```
POZIV: ./a.out rodjendani.txt
DATOTEKA RODJENDANI.TXT NE POSTOJI
INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje datoteke
rodjendani.txt.
```

[Rešenje 4.18]

**Zadatak 4.19** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju **int identitet(Cvor \* koren1, Cvor \* koren2)** koja proverava da li su binarna stabla **koren1** i **koren2** koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Prvo stablo:
10 5 15 3 2 4 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla jesu identicna.
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Prvo stablo:
10 5 15 4 3 2 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla nisu identicna.
```

[Rešenje 4.19]

**\* Zadatak 4.20** Napisati program za rad sa skupovima u kojem se skupovi predstavljaju pomoću binarnih pretraživačkih stabala. Program za dva skupa čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku skupova. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Prvi skup: 1 7 8 9 2 2
Drugi skup: 3 9 6 11 1
Unija: 1 1 2 2 3 6 7 8 9 9 11
Presek: 1 9
Razlika: 2 2 7 8
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Prvi skup: 11 2 7 5
Drugi skup: 4 3 3 7
Unija: 2 3 3 4 5 7 7 11
Presek: 7
Razlika: 2 5 11
```

[Rešenje 4.20]

**Zadatak 4.21** Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
n: 7
a: 1 11 8 6 37 25 30
1 6 8 11 25 30 37
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
n: 55
Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

**Zadatak 4.22** Dato je binarno pretraživačko stablo celih brojeva.

- (a) Napisati funkciju koja izračunava broj čvorova stabla.
- (b) Napisati funkciju koja izračunava broj listova stabla.
- (c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.
- (d) Napisati funkciju koja izračunava zbir čvorova stabla.
- (e) Napisati funkciju koja izračunava najveći element stabla.
- (f) Napisati funkciju koja izračunava dubinu stabla.
- (g) Napisati funkciju koja izračunava broj čvorova na  $i$ -tom nivou stabla.
- (h) Napisati funkciju koja ispisuje sve elemente na  $i$ -tom nivou stabla.
- (i) Napisati funkciju koja izračunava maksimalnu vrednost na  $i$ -tom nivou stabla.
- (j) Napisati funkciju koja izračunava zbir čvorova na  $i$ -tom nivou stabla.
- (k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti  $x$ .

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara  $i$  i  $x$  pročitati kao argumente komandne linije. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Test 1

```
Poziv: ./a.out 2 15
Uzorak:
10 5 15 3 2 4 30 12 14 13
Izlaz:
Broj cvorova: 10
Broj listova: 4
Pozitivni listovi: 2 4 13 30
Zbir cvorova: 108
Najveci element: 30
Dubina stabla: 5
Broj cvorova na 2. nivou: 3
Elementi na 2. nivou: 3 12 30
Maksimalni element na 2. nivou: 30
Zbir elemenata na 2. nivou: 45
Zbir elemenata manjih ili jednakih od 15: 78
```

Test 2

```
Poziv: ./a.out 3 31
Uzorak:
24 53 61 9 7 55 20 16
Izlaz:
Broj cvorova: 8
Broj listova: 3
Pozitivni listovi: 7 16 55
Zbir cvorova: 245
Najveci element: 61
Dubina stabla: 4
Broj cvorova na 3. nivou: 2
Elementi na 3. nivou: 16 55
Maksimalni element na 3. nivou: 55
Zbir elemenata na 3. nivou: 71
Zbir elemenata manjih ili jednakih od 31: 76
```

[Rešenje 4.22]

**Zadatak 4.23** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>ULAZ: 10 5 15 3 2 4 30 12 14 13</pre>	<pre>ULAZ: 6 11 8 3 -2</pre>	<pre>ULAZ: 24 53 61 9 7 55 20 16</pre>
<pre>IZLAZ: 0.nivo: 10 1.nivo: 5 15 2.nivo: 3 12 30 3.nivo: 2 4 14 4.nivo: 13</pre>	<pre>IZLAZ: 0.nivo: 6 1.nivo: 3 11 2.nivo: -2 8</pre>	<pre>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 slična kao u ogledalu.</pre>	<pre>INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 20 15 0 Stabla nisu slična kao u ogledalu.</pre>

**Zadatak 4.25** AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor * koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Test 1*

```
|| ULAZ:
  ||| 10 5 15 2 11 16 1 13
  || IZLAZ:
  ||| 7
```

*Test 2*

```
|| ULAZ:
  ||| 16 30 40 24 10 18 45 22
  || IZLAZ:
  ||| 6
```

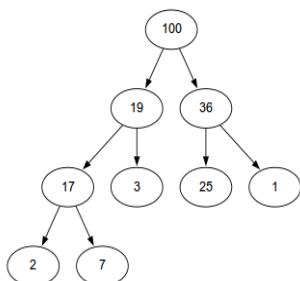
[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *heap* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int heap(Cvor * koren)` koja proverava da li je dato binarno stablo celih brojeva heap. Napisati zatim i glavni program koji kreira stablo zadato slikom 4.1, poziva funkciju `heap` i ispisuje rezultat na standardni izlaz. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

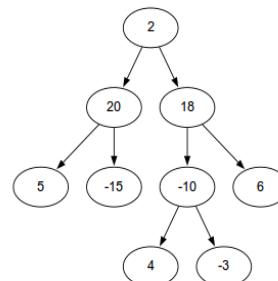
*Test 1*

```
|| IZLAZ:
  ||| Zadato stablo je heap!
```

[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 pronađe čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

*Test 1*

```
|| IZLAZ:  
|| Vrednost u cvoru sa maksimalnim desnim zbirom: 18  
|| Vrednost u cvoru sa minimalnim levim zbirom: 18  
|| 2 18 -10 4  
|| 2 20 -15
```

## 4.3 Rešenja

### Rešenje 4.1

Datoteka 4.1: *lista.h*

```
#ifndef _LISTA_H_  
#define _LISTA_H_  
  
/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni  
   podatak vrednost i pokazivac na sledeci cvor liste */  
typedef struct cvor {  
    int vrednost;  
    struct cvor *sledeci;  
} Cvor;  
  
/* Funkcija kreira cvor, vrednost novog cvora inicializuje na broj,  
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac  
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */  
Cvor *napravi_cvor(int broj);  
  
/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste  
   ciji se pokazivac glava nalazi na adresi adresa_glave. */  
void osloboodi_listu(Cvor ** adresa_glave);  
  
/* 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 dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
   dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
38 int dodaj_iza(Cvor * tekuci, int broj);

40 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   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 sadran trazeni broj ili
   NULL u slucaju da takav cvor ne postoji u listi. */
46 Cvor *pretrazi_listu(Cvor * glava, int broj);

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

52 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
   obrije stara glava. */
54 void obrisi_cvor(Cvor ** adresa_glave, int broj);

56 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista sortirana
   neopadajuće. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrije stara glava liste. */
58 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

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

64 #endif

```

Datoteka 4.2: lista.c

```

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

5 Cvor *napravi_cvor(int broj)
{
7     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
       alokacije */
9     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10    if (novi == NULL)
11        return NULL;

13    /* Inicijalizacija polja strukture */
14    novi->vrednost = broj;
15    novi->sledeci = NULL;

17    /* Vraca se adresa novog cvora */
18    return novi;
19}

21 void osloboodi_listu(Cvor ** adresu_glave)
{
23    Cvor *pomocni = NULL;

25    /* Ako lista nije prazna, onda treba oslobooditi memoriju */
26    while (*adresa_glave != NULL) {
27        /* Potrebno je prvo zapamtitи adresu sledeceg cvora i onda
           oslobooditi cvor koji predstavlja glavu liste */
28        pomocni = (*adresa_glave)->sledeci;
29        free(*adresa_glave);

31        /* Sledeci cvor je nova glava liste */
32        *adresa_glave = pomocni;
33    }
35}

37 int dodaj_na_pocetak_liste(Cvor ** adresu_glave, int broj)
{
38    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39    Cvor *novi = napravi_cvor(broj);
40    if (novi == NULL)
41        return 1;

43    /* Novi cvor se uvezuje na pocetak i postaje nova glava liste */
44    novi->sledeci = *adresa_glave;
45    *adresa_glave = novi;

47    /* Vraca se indikator uspesnog dodavanja */
48    return 0;
49}

```

```

    }

51 Cvor *pronadji_poslednji(Cvor * glava)
52 {
53     /* U praznoj listi nema cvorova pa se vraca NULL */
54     if (glava == NULL)
55         return NULL;

56     /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
57     glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
58     ce pokazivati na cvor liste koji nema sledbenika, tj. na
59     poslednji cvor liste pa se vraca vrednost pokazivaca glava.

60     Pokazivac glava je argument funkcije i njegove promene nece se
61     odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
62     while (glava->sledeci != NULL)
63         glava = glava->sledeci;

64     return glava;
65 }

66 int dodaj_na_kraj_liste(Cvor ** adresa_glage, int broj)
67 {
68     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
69     Cvor *novi = napravi_cvor(broj);
70     if (novi == NULL)
71         return 1;

72     /* Ako je lista prazna */
73     if (*adresa_glage == NULL) {
74         /* Glava nove liste je upravo novi cvor */
75         *adresa_glage = novi;
76     } else {
77         /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
78         se dodaje na kraj liste kao sledbenik poslednjeg */
79         Cvor *poslednji = pronadji_poslednji(*adresa_glage);
80         poslednji->sledeci = novi;
81     }

82     /* Vraca se indikator uspesnog dodavanja */
83     return 0;
84 }

85 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
86 {
87     /* U praznoj listi nema takvog mesta i vraca se NULL */
88     if (glava == NULL)
89         return NULL;

90     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
91     pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
92     vecu ili jednaku vrednosti novog cvora.
93 
```

```

103     Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
105     provera da li se doslo do poslednjeg cvora liste pre nego sto se
106     proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
107     sledeci ne postoji, pa ni njegova vrednost. */
108     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
109         glava = glava->sledeci;
110
111     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
112      poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
113      vrednost vecu od broj. */
114     return glava;
115 }
116
117 int dodaj_iza(Cvor * tekuci, int broj)
118 {
119     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
120     Cvor *novi = napravi_cvor(broj);
121     if (novi == NULL)
122         return 1;
123
124     /* Novi cvor se dodaje iza tekuceg cvora. */
125     novi->sledeci = tekuci->sledeci;
126     tekuci->sledeci = novi;
127
128     /* Vraca se indikator uspesnog dodavanja */
129     return 0;
130 }
131
132 int dodaj_sortirano(Cvor ** adresa_glove, int broj)
133 {
134     /* Ako je lista prazna */
135     if (*adresa_glove == NULL) {
136         /* Glava nove liste je novi cvor */
137
138         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
139         Cvor *novi = napravi_cvor(broj);
140         if (novi == NULL)
141             return 1;
142
143         *adresa_glove = novi;
144
145         /* Vraca se indikator uspesnog dodavanja */
146         return 0;
147     }
148
149     /* Inace... */
150
151     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
152      treba dodati na pocetak liste */
153     if ((*adresa_glove)->vrednost >= broj) {
154         return dodaj_na_pocetak_liste(adresa_glove, broj);

```

```

155    }
156
157    /* U slučaju da je glava liste cvor sa vrednoscu manjom od broj,
158     tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
159    Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glove, broj);
160    return dodaj_iza(pomocni, broj);
161}
162
163Cvor *pretrazi_listu(Cvor * glava, int broj)
164{
165    /* Obilaze se cvorovi liste */
166    for (; glava != NULL; glava = glava->sledeci)
167        /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
168         se obustavlja */
169        if (glava->vrednost == broj)
170            return glava;
171
172    /* Nema trazenog broja u listi i vraca se NULL */
173    return NULL;
174}
175
176Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
177{
178    /* Obilaze se cvorovi liste */
179    /* U uslovu ostanka u petlji, bitan je redosled provera u
180     konjunkciji */
181    while (glava != NULL && glava->vrednost < broj)
182        glava = glava->sledeci;
183
184    /* Iz petlje se moglo izaci na vise nacina. Prvi, tako sto je
185     glava->vrednost veca od trazenog broja i tada treba vratiti
186     NULL, jer tražen broj nije nadjen medju manjim brojevima pri
187     pocetku sortirane liste, pa se moze zakljuciti da ga nema u
188     listi. Drugi nacini, tako sto se doslo do kraja liste i glava je
189     NULL ili tako sto je glava->vrednost == broj. U oba poslednja
190     nacina treba vratiti pokazivac glava bilo da je NULL ili
191     pokazivac na konkretan cvor. */
192    if (glava->vrednost > broj)
193        return NULL;
194    else
195        return glava;
196}
197
198void obrisi_cvor(Cvor ** adres_glove, int broj)
199{
200    Cvor *tekuci = NULL;
201    Cvor *pomocni = NULL;
202
203    /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
204     broju i azurira se pokazivac na glavu liste */
205    while (*adres_glove != NULL && (*adres_glove)->vrednost == broj)
206    {

```

```

205     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
206      adresi adresa_glave */
207     pomocni = (*adresa_glave)->sledeci;
208     free(*adresa_glave);
209     *adresa_glave = pomocni;
210 }
211
212 /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
213   funkcije */
214 if (*adresa_glave == NULL)
215   return;
216
217 /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
218   pokazuje na cvor cija je vrednost razlicita od trazenog broja.
219   Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
220   sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
221   ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
222   Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
223 */
224 tekuci = *adresa_glave;
225 while (tekuci->sledeci != NULL)
226   if (tekuci->sledeci->vrednost == broj) {
227     /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
228       prvo cuva u promenljivoj pomocni. */
229     pomocni = tekuci->sledeci;
230     /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
231       njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
232       sledeci od cvora koji se brise. */
233     tekuci->sledeci = pomocni->sledeci;
234     /* Sada treba osloboditi cvor sa vrednoscu broj. */
235     free(pomocni);
236   } else {
237     /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
238       pomera na sledeci. */
239     tekuci = tekuci->sledeci;
240   }
241   return;
242 }
243 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
244 {
245   Cvor *tekuci = NULL;
246   Cvor *pomocni = NULL;
247
248   /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
249     broju i azurira se pokazivac na glavu liste. */
250   while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
251   {
252     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
253       adresi adresa_glave. */
254     pomocni = (*adresa_glave)->sledeci;
255     free(*adresa_glave);

```

```

255     *adresa_glave = pomocni;
256 }
257
258 /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
259    prekida. Isto se radi i ukoliko glava liste sadrzi vrednost koja
260    je veca od broja, jer kako je lista sortirana rastuce nema
261    potrebe broj traziti u repu liste. */
262 if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
263     return;
264
265 /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
266    na cvor cija vrednost je manja od trazenog broja, kao i svim
267    cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
268    je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
269    ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
270    cija vrednost je veca od trazenog broja. */
271 tekuci = *adresa_glave;
272 while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
273     )
274     if (tekuci->sledeci->vrednost == broj) {
275         pomocni = tekuci->sledeci;
276         tekuci->sledeci = tekuci->sledeci->sledeci;
277         free(pomocni);
278     } else {
279         /* Ne treba brisati sledeceg od tekuceg jer je manji od
280            trazenog i tekuci se pomera na sledeci cvor. */
281         tekuci = tekuci->sledeci;
282     }
283     return;
284 }
285 void ispisi_listu(Cvor * glava)
286 {
287     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
288        jer se lista nece menjati */
289     putchar('[');
290     /* Unutar zagradica ispisuju se vrednosti u cvorovima liste od
291        pocetka prema kraju liste. */
292     for (; glava != NULL; glava = glava->sledeci) {
293         printf("%d", glava->vrednost);
294         if (glava->sledeci != NULL)
295             printf(", ");
296     }
297     printf("]\n");
298 }
```

Datoteka 4.3: *main\_a.c*

```

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

```

4  /* 1) Glavni program */
6  int main()
{
8  /* Lista je prazna na pocetku */
10 Cvor *glava = NULL;
12 Cvor *trazeni = NULL;
14 int broj;
16
18 /* Testiranje dodavanja novog broja na pocetak liste */
20 printf("Unesite brojeve: (za kraj CTRL+D)\n");
22 while (scanf("%d", &broj) > 0) {
24     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
26         memorije za nov cvor. Memoriju alociranu za cvorove liste
28         treba oslobođiti pre napustanja programa. */
30     if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
32         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
34         oslobođi_listu(&glava);
36         exit(EXIT_FAILURE);
38     }
39     printf("\tLista: ");
40     ispisi_listu(glava);
42 }
44
46 /* Testiranje funkcije za pretragu liste */
48 printf("\nUnesite broj koji se trazi: ");
50 scanf("%d", &broj);
52
54 trazeni = pretrazi_listu(glava, broj);
56 if (trazeni == NULL)
58     printf("Broj %d se ne nalazi u listi!\n", broj);
60 else
62     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
64
66 /* Oslobadja se memorija zauzeta listom */
68 oslobođi_listu(&glava);
70
72 exit(EXIT_SUCCESS);
74 }
```

Datoteka 4.4: *main\_b.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* 2) Glavni program */
6 int main()
{
8  /* Lista je prazna na pocetku */
10 Cvor *glava = NULL;
```

```

10 int broj;

12 /* Testira se funkcija za dodavanja novog broja na kraj liste */
13 printf("Unesite brojeve: (za kraj CTRL+D)\n");
14 while (scanf("%d", &broj) > 0) {
15     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
16        memorije za nov cvor. Memoriju alociranu za cvorove liste
17        treba oslobođiti pre napustanja programa. */
18     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
19         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
20         osloboди_listu(&glava);
21         exit(EXIT_FAILURE);
22     }
23     printf("\tLista: ");
24     ispisi_listu(glava);
25 }

26 /* Testira se funkcije kojom se brise cvor liste */
27 printf("\nUnesite broj koji se brise: ");
28 scanf("%d", &broj);

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

32 printf("Lista nakon brisanja: ");
33 ispisi_listu(glava);

34 /* Oslobadja se memorija zauzeta listom */
35 osloboди_listu(&glava);

36 exit(EXIT_SUCCESS);
37 }
```

Datoteka 4.5: *main\_c.c*

```

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

4 /* 3) Glavni program */
5 int main()
6 {
7     /* Lista je prazna na pocetku */
8     Cvor *glava = NULL;
9     Cvor *trazenii = NULL;
10    int broj;

11    /* Testira se funkcija za dodavanje vrednosti u listu tako da bude
12       uređena neopadajuće */
13    printf("Unosite brojeve (za kraj CTRL+D)\n");
```

```

16  while (scanf("%d", &broj) > 0) {
18      /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
         memorije za nov cvor. Memoriju alociranu za cvorove liste
         treba oslobođiti pre napustanja programa. */
20      if (dodaj_sortirano(&glava, broj) == 1) {
22          fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23          oslobođi_listu(&glava);
24          exit(EXIT_FAILURE);
25      }
26      printf("\tLista: ");
27      ispisi_listu(glava);
28  }

29  /* Testira se funkcija za pretragu liste */
30  printf("\nUnesite broj koji se trazi: ");
31  scanf("%d", &broj);

32  traženi = pretrazi_listu(glava, broj);
33  if (traženi == NULL)
34      printf("Broj %d se ne nalazi u listi!\n", broj);
35  else
36      printf("Traženi broj %d je u listi!\n", traženi->vrednost);

37  /* Testira se funkcija kojom se brise cvor liste */
38  printf("\nUnesite broj koji se brise: ");
39  scanf("%d", &broj);

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

42  printf("Lista nakon brisanja: ");
43  ispisi_listu(glava);

44  /* Oslobadja se memorija zauzeta listom */
45  oslobođi_listu(&glava);

46  exit(EXIT_SUCCESS);
47 }
```

## Rešenje 4.2

Datoteka 4.6: *lista.h*

```

1 #ifndef _LISTA_H_
2 #define _LISTA_H_

4 /* Struktura kojom je predstavljen cvor liste sadrži celobrojni
   podatak vrednost i pokazivac na sledeći cvor liste. */
6 typedef struct cvor {
```

```

8     int vrednost;
9     struct cvor *sledeci;
10    } Cvor;
11
12   /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
13      dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14      na novokreirani cvor ili NULL ako alokacija nije bila uspesna.*/
15 Cvor *napravi_cvor(int broj);
16
17   /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
18      ciji se pokazivac glava nalazi na adresi adresa_glave. */
19 void osloboodi_listu(Cvor ** adresa_glave);
20
21   /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
22      bilo greske pri alokaciji memorije, inace vraca 0. */
23 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
24
25   /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
26      pri alokaciji memorije, inace vraca 0. */
27 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
28
29   /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
30      ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
31      memorije, inace vraca 0. */
32 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
33
34   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
35      Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
36      NULL u slucaju da takav cvor ne postoji u listi. */
37 Cvor *pretrazi_listu(Cvor * glava, int broj);
38
39   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
40      U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
41      neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
42      sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
43      */
44 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
45
46   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
47      pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
48      obrije stara glava liste. */
49 void obrisi_cvor(Cvor ** adresa_glave, int broj);
50
51   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
52      oslanjajući se na cinjenicu da je prosledjena lista sortirana
53      neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
54      promenjen ukoliko se obrije stara glava liste. */
55 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
56
57   /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
58      zapetama. */
59 void ispisi_vrednosti(Cvor * glava);

```

```

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

```

Datoteka 4.7: lista.c

```

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

5 Cvor *napravi_cvor(int broj)
{
7  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8   alokacije */
9  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10 if (novi == NULL)
11    return NULL;

13 /* Inicijalizacija polja strukture */
14 novi->vrednost = broj;
15 novi->sledeci = NULL;

17 /* Vraca se adresa novog cvora */
18 return novi;
19 }

21 void osloboodi_listu(Cvor ** adres_a_glave)
{
23 /* Ako je lista vec prazna */
24 if (*adres_a_glave == NULL)
25    return;

27 /* Ako lista nije prazna, treba oslobooditi memoriju. Pre
28   oslobadjanja memorije za glavu liste, treba oslobooditi rep liste
29   */
30 osloboodi_listu(&(*adres_a_glave)->sledeci);
31 /* Nakon osloboodenog repa, oslobadja se glava liste i azurira se
32   glava u pozivajucoj funkciji tako da odgovara praznoj listi */
33 free(*adres_a_glave);
34 *adres_a_glave = NULL;
35 }

37 int dodaj_na_pocetak_liste(Cvor ** adres_a_glave, int broj)
{
39 /* Kreira se novi cvor i proverava se uspesnost kreiranja */
40 Cvor *novi = napravi_cvor(broj);
41 if (novi == NULL)
42    return 1;

```

```

43     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
45     novi->sledeci = *adresa_glave;
46     *adresa_glave = novi;
47
48     /* Vraca se indikator uspesnog dodavanja */
49     return 0;
50 }
51
52 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
53 {
54     /* Ako je lista prazna */
55     if (*adresa_glave == NULL) {
56
57         /* Novi cvor postaje glava liste */
58         Cvor *novi = napravi_cvor(broj);
59         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */
60         /*
61         if (novi == NULL)
62             return 1;
63
64         /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
65            azurira i pokazivacka promenljiva u pozivajucoj funkciji */
66         *adresa_glave = novi;
67
68         /* Vraca se indikator uspesnog dodavanja */
69         return 0;
70     }
71
72     /* Ako lista nije prazna, broj se dodaje u rep liste. */
73     /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
74        novi broj se dodaje u rep liste u rekursivnom pozivu.
75        Informaciju o uspesnosti alokacije u rekursivnom pozivu funkcija
76        prosledjuje visem rekursivnom pozivu koji tu informaciju vraca u
77        rekursivni poziv iznad, sve dok se ne vrati u main. Tek je iz
78        main funkcije moguce pristupiti pravom pocetku liste i
79        osloboziti je celu, ako ima potrebe. Ako je funkcija vratila 0,
80        onda nije bilo greske. */
81     return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
82 }
83
84 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
85 {
86     /* Ako je lista prazna */
87     if (*adresa_glave == NULL) {
88
89         /* Novi cvor postaje glava liste */
90         Cvor *novi = napravi_cvor(broj);
91
92         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */
93         /*
94         if (novi == NULL)

```

```

93     return 1;
95
96     /* Azurira se glava liste */
97     *adresa_glave = novi;
98
99     /* Vraca se indikator uspesnog dodavanja */
100    return 0;
101}
102
103/* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
104   glavi liste, onda se dodaje na pocetak liste */
105if ((*adresa_glave)->vrednost >= broj)
106    return dodaj_na_pocetak_liste(adresa_glave, broj);
107
108/* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
109   bude sortirana lista. */
110return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
111}
112
113Cvor *pretrazi_listu(Cvor * glava, int broj)
114{
115    /* U praznoj listi nema vrednosti */
116    if (glava == NULL)
117        return NULL;
118
119    /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
120    if (glava->vrednost == broj)
121        return glava;
122
123    /* Inace, pretraga se nastavlja u repu liste */
124    return pretrazi_listu(glava->sledeci, broj);
125}
126
127Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
128{
129    /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
130       vrednosti u glavi liste, jer je lista neopadajuće sortirana */
131    if (glava == NULL || glava->vrednost > broj)
132        return NULL;
133
134    /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
135    if (glava->vrednost == broj)
136        return glava;
137
138    /* Inace, pretraga se nastavlja u repu liste */
139    return pretrazi_listu(glava->sledeci, broj);
140}
141void obrisi_cvor(Cvor ** adresa_glave, int broj)
142{
143    /* U praznoj listi nema cvorova za brisanje. */
144    if (*adresa_glave == NULL)

```

```

145     return;
147 /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
148 obrisi_cvor(&(*adresa_glave)->sledeci, broj);
149
151 /* Preostaje provera da li glavu liste treba obrisati */
152 if ((*adresa_glave)->vrednost == broj) {
153     /* Pomocni pokazuje na cvor koji treba da se obrije */
154     Cvor *pomocni = *adresa_glave;
155     /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
156      brise se cvor koji je bio glava liste. */
157     *adresa_glave = (*adresa_glave)->sledeci;
158     free(pomocni);
159 }
161
163 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
164 {
165     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
166      od broja, kako je lista sortirana rastuce nema potrebe broj
167      traziti u repu liste i zato se funkcija prekida */
168     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
169         return;
171
172     /* Brisu se cvorovi iz repa koji imaju vrednost broj */
173     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
174
175     /* Preostaje provera da li glavu liste treba obrisati */
176     if ((*adresa_glave)->vrednost == broj) {
177         /* Pomocni pokazuje na cvor koji treba da se obrije */
178         Cvor *pomocni = *adresa_glave;
179         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
180          brise se cvor koji je bio glava liste */
181         *adresa_glave = (*adresa_glave)->sledeci;
182         free(pomocni);
183     }
184 }
186 void ispisi_vrednosti(Cvor * glava)
187 {
188     /* Prazna lista */
189     if (glava == NULL)
190         return;
191
192     /* Ispisuje se vrednost u glavi liste */
193     printf("%d", glava->vrednost);
194
195     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
196      se poziva ista funkcija za ispis ostalih. */
197     if (glava->sledeci != NULL) {
198         printf(", ");
199     }

```

```

197     ispis_i_vrednosti(glava->sledeci);
198 }
199 }
200
201 void ispis_i_listu(Cvor * glava)
202 {
203     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
204      jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
205      na glavu liste iz pozivajuce funkcije. Ona ispisuje samo
206      zgrade, a rekurzivno ispisivanje vrednosti u listi prepusta
207      rekurzivnoj pomocnoj funkciji ispis_i_vrednosti, koja ce ispisati
208      elemente razdvojene zapetom i razmakom. */
209     putchar('[');
210     ispis_i_vrednosti(glava);
211     printf("]\n");
212 }
```

### Rešenje 4.3

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX_DUZINA 20
6
7 /* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
8   broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
9 typedef struct _Cvor {
10     char etiketa[20];
11     unsigned broj_pojavljivanja;
12     struct _Cvor *sledeci;
13 } Cvor;
14
15 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
16   ili NULL ako alokacija nije uspesno izvrsena */
17 Cvor *napravi_cvor(unsigned br, char *etiketa)
18 {
19     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
20       alokacije */
21     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
22     if (novi == NULL)
23         return NULL;
24
25     /* Inicijalizacija polja strukture */
26     novi->broj_pojavljivanja = br;
27     strcpy(novi->etiketa, etiketa);
28     novi->sledeci = NULL;
29
30     /* Vraca se adresa novog cvora */
31     return novi;
32 }
```

```

33  /* Funkcija oslobođaja dinamicku memoriju zauzetu cvorovima liste */
35  void osloboodi_listu(Cvor ** adresa_glave)
{
37      Cvor *pomocni = NULL;

39      /* Sve dok lista ni bude prazna, brise se tekuća glava liste i
40          azurira se vrednost glave liste */
41      while (*adresa_glave != NULL) {
42          pomocni = (*adresa_glave)->sledeci;
43          free(*adresa_glave);
44          *adresa_glave = pomocni;
45      }
46      /* Nakon izlaska iz petlje pokazivac glava u main funkciji koji se
47          nalazi na adresi adresa_glave bice postavljen na NULL vrednost.
48          */
49 }
50
51 /* Funkcija dodaje novi cvor na pocetak liste. Povratna vrednost je 1
52     ako je doslo do greske pri alokaciji memorije za nov cvor, odnosno
53     0 */
54 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
55                             char *etiketa)
56 {
57     /* Kreira se novi cvor i proverava se uspesnost alokacije */
58     Cvor *novi = napravi_cvor(br, etiketa);
59     if (novi == NULL)
60         return 1;

61     /* Dodaje se novi cvor na pocetak liste */
62     novi->sledeci = *adresa_glave;
63     *adresa_glave = novi;

64     /* Vraca se indikator uspesnog dodavanja */
65     return 0;
66 }

67 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
68     NULL ako takav cvor ne postoji u listi */
69 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
70 {
71     Cvor *tekuci;

72     /* Obilazi se cvor po cvor liste */
73     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
74         /* Ako tekuci cvor sadrzi trazenu etiketu, vracamo njegovu
75             vrednost */
76         if (strcmp(tekuci->etiketa, etiketa) == 0)
77             return tekuci;

78     /* Cvor nije pronadjen */
79     return NULL;
80 }

81
82
83

```

```
85 }  
86 /* Funkcija ispisuje sadrzaj liste */  
87 void ispis_listu(Cvor * glava)  
{  
88     /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i  
     broj njihovog pojavljivanja u HTML datoteci. */  
89     for (; glava != NULL; glava = glava->sledeci)  
90         printf("%s - %u\n", glava->etiketa, glava->broj_pojavljenja);  
91 }  
92  
93 /* Glavni program */  
94 int main(int argc, char **argv)  
95 {  
96     /* Provera se da li je program pozvan sa ispravnim brojem  
     argumenata. */  
97     if (argc != 2) {  
98         fprintf(stderr,  
99                 "Greska! Program se poziva sa: ./a.out datoteka.html!\n")  
100        ;  
101        exit(EXIT_FAILURE);  
102    }  
103  
104    /* Priprema datoteke za citanje */  
105    FILE *in = NULL;  
106    in = fopen(argv[1], "r");  
107    if (in == NULL) {  
108        fprintf(stderr,  
109                 "Greska prilikom otvaranja datoteke %s!\n", argv[1]);  
110        exit(EXIT_FAILURE);  
111    }  
112  
113    char c;  
114    int i = 0;  
115    char procitana[MAX_DUZINA];  
116    Cvor *glava = NULL;  
117    Cvor *trazeni = NULL;  
118  
119    /* Cita se karakter po karakter datoteke sve dok se ne procita cela  
     datoteka */  
120    while ((c = fgetc(in)) != EOF) {  
121  
122        /* Proverava se da li se pocinje sa citanjem nove etikete */  
123        if (c == '<') {  
124            /* Proverava se da li se cita zatvarajuca etiketa */  
125            if ((c = fgetc(in)) == '/') {  
126                i = 0;  
127                while ((c = fgetc(in)) != '>')  
128                    procitana[i++] = c;  
129            }  
130            /* Cita se zatvarajuca etiketa */  
131        }  
132        else {  
133    }
```

```

135         i = 0;
137         procitana[i++] = c;
138         while ((c = fgetc(in)) != ' ' && c != '>')
139             procitana[i++] = c;
140     }
141     procitana[i] = '\0';
142
143     /* Trazi se procitana etiketa medju postojecim cvorovima liste.
144      Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
145      sa
146      brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
147      etikete */
148     trazeni = pretrazi_listu(glava, procitana);
149     if (trazeni == NULL) {
150         if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
151             fprintf(stderr, "Neuspela alokacija za nov cvor\n");
152             oslobodi_listu(&glava);
153             exit(EXIT_FAILURE);
154         }
155     } else
156         trazeni->broj_pojavljivanja++;
157 }
158
159 /* Zatvaranje datoteke */
160 fclose(in);
161
162 /* Ispisuje se sadrzaj cvorova liste */
163 ispisi_listu(glava);
164
165 /* Oslobadja se memorija zauzeta listom */
166 oslobodi_listu(&glava);
167
168 exit(EXIT_SUCCESS);
}

```

## Rešenje 4.4

```

#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 }

```

```

14     struct _Cvor *sledeci;
15 } Cvor;
16
17 /* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
18    novi cvor ili NULL ukoliko je doslo do greske */
19 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
21     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
22        alokacije */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
26
27     /* Inicijalizacija polja strukture */
28     strcpy(novi->broj_indeksa, broj_indeksa);
29     strcpy(novi->ime, ime);
30     strcpy(novi->prezime, prezime);
31     novi->sledeci = NULL;
32
33     /* Vraca se adresa novog cvora */
34     return novi;
35 }
36
37 /* Funkcija oslobadja memoriju zauzetu cvorovima liste */
38 void osloboodi_listu(Cvor **adresa_glave)
39 {
40     /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
41     if (*adresa_glave == NULL)
42         return;
43
44     /* Rekurzivnim pozivom se oslobadja rep liste */
45     osloboodi_listu(&(*adresa_glave)->sledeci);
46
47     /* Potom se oslobadja i glava liste */
48     free(*adresa_glave);
49
50     /* Proglasava se lista praznom */
51     *adresa_glave = NULL;
52 }
53
54 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
55    do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
56 int dodaj_na_pocetak_liste(Cvor **adresa_glave, char *broj_indeksa,
57                             char *ime, char *prezime)
58 {
59     /* Kreira se novi cvor i proverava se uspesnost alokacije */
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
61     if (novi == NULL)
62         return 1;
63
64     /* Dodaje se novi cvor na pocetak liste */
65     novi->sledeci = *adresa_glave;

```

```

66     *adresa_glave = novi;
68
69     /* Vraca se indikator uspesnog dodavanja */
70     return 0;
71 }
72
73 /* Funkcija ispisuje sadrzaj cvorova liste. */
74 void ispis_listu(Cvor * glava)
75 {
76     /* Pocevsi od glave liste */
77     for (; glava != NULL; glava = glava->sledeci)
78         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
79                glava->prezime);
80 }
81
82 /* Funkcija vraca cvor koji kao vrednost sadrzi trazen i broj indeksa.
83  U suprotnom funkcija vraca NULL */
84 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
85 {
86     /* Ako je lista prazna, ne postoji trazen cvor */
87     if (glava == NULL)
88         return NULL;
89
90     /* Poredi se trazen i broj indeksa sa brojem indeksa u glavi liste
91      */
92     if (!strcmp(glava->broj_indeksa, broj_indeksa))
93         return glava;
94
95     /* Ukoliko u glavi liste nije trazen indeks, pretraga se nastavlja
96      u repu liste */
97     return pretrazi_listu(glava->sledeci, broj_indeksa);
98 }
99
100 /* Glavni program */
101 int main(int argc, char **argv)
102 {
103     /* Argumenti komandne linije su neophodni jer se iz komandne linije
104      dobija ime datoteke sa informacijama o studentima */
105     if (argc != 2) {
106         fprintf(stderr,
107                 "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
108         exit(EXIT_FAILURE);
109     }
110
111     /* Priprema datoteke za citanje */
112     FILE *in = NULL;
113     in = fopen(argv[1], "r");
114     if (in == NULL) {
115         fprintf(stderr,
116                 "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
117         exit(EXIT_FAILURE);
118     }

```

```

118  /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
119   listu */
120  char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
121  char broj_indeksa[MAX_INDEKS];
122  Cvor *glava = NULL;
123  Cvor *trazeni = NULL;

124  /* Ucitavanje vrednosti u listu */
125  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
126    if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
127      fprintf(stderr, "Neuspela alokacija za nov cvor\n");
128      osloboodi_listu(&glava);
129      exit(EXIT_FAILURE);
130    }

132  /* Datoteka vise nije potrebna i zatvara se. */
133  fclose(in);

136  /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
137  while (scanf("%s", broj_indeksa) != EOF) {
138    trazeni = pretrazi_listu(glava, broj_indeksa);
139    if (trazeni == NULL)
140      printf("ne\n");
141    else
142      printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
143  }

144  /* Oslobadja se memorija zauzeta za cvorove liste. */
145  osloboodi_listu(&glava);

148  exit(EXIT_SUCCESS);
}

```

### 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
6   na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7   pokazivaca postojećih cvorova listi */
8  Cvor *objedini(Cvor **adresa_glave_1, Cvor **adresa_glave_2)
9  {
10   /* Pokazivaci na pocetne cvorove liste koje se prevezuju */
11   Cvor *lista1 = *adresa_glave_1;
12   Cvor *lista2 = *adresa_glave_2;

13   /* Pokazivac na pocetni cvor rezultujuće liste */
14   Cvor *rezultujuca = NULL;
15 }

```

```

17     Cvor *tekuci = NULL;
19
21     /* Ako su obe liste prazne i rezultat je prazna lista */
23     if (lista1 == NULL && lista2 == NULL)
24         return NULL;
26
28     /* Ako je prva lista prazna, rezultat je druga lista */
30     if (lista1 == NULL)
31         return lista2;
33
35     /* Ako je druga lista prazna, rezultat je prva lista */
37     if (lista2 == NULL)
38         return lista1;
40
42     /* Odredjuje se prvi cvor rezultujuće liste - to je ili prvi cvor
44         liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
46         koji sadrži manju vrednost */
48     if (lista1->vrednost < lista2->vrednost) {
49         rezultujuca = lista1;
50         lista1 = lista1->sledeci;
51     } else {
52         rezultujuca = lista2;
53         lista2 = lista2->sledeci;
54     }
56
58     /* Kako promenljiva rezultujuća pokazuje na početak nove liste, ne
59         sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
60         koji sadrži adresu trenutnog cvora rezultujuće liste */
61     tekuci = rezultujuca;
63
65     /* U svakoj iteraciji petlje rezultujućoj listi se dodaje novi cvor
66         tako da bude uređena neopadajuće. Pokazivac na listu iz koje se
67         uzima cvor se azurira tako da pokazuje na sledeći cvor */
68     while (lista1 != NULL && lista2 != NULL) {
69         if (lista1->vrednost < lista2->vrednost) {
70             tekuci->sledeci = lista1;
71             lista1 = lista1->sledeci;
72         } else {
73             tekuci->sledeci = lista2;
74             lista2 = lista2->sledeci;
75         }
76         tekuci = tekuci->sledeci;
77     }
79
80     /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
81         rezultujuću listu treba nadovezati ostatak druge liste */
83     if (lista1 == NULL)
84         tekuci->sledeci = lista2;
85     else
86         /* U suprotnom treba nadovezati ostatak prve liste */
87         tekuci->sledeci = lista1;
89
90     /* Preko adresa glava polaznih listi vrednosti pokazivaca u

```

```

69      pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
70      prethodnih listi nalaze negde unutar rezultujuće liste. Do njih
71      se može doći prateći pokazivace iz glave rezultujuće liste, tako
72      da stare pokazivace treba postaviti na NULL. */
73      *adresa_glave_1 = NULL;
74      *adresa_glave_2 = NULL;

75      return rezultujuca;
76  }

77 int main(int argc, char **argv)
78 {
79     /* Argumenti komandne linije su neophodni */
80     if (argc != 3) {
81         fprintf(stderr,
82             "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
83         exit(EXIT_FAILURE);
84     }

85     /* Otvaramo datoteke u kojima se nalaze elementi 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     FILE *in2 = NULL;
94     in2 = fopen(argv[2], "r");
95     if (in2 == NULL) {
96         fprintf(stderr,
97             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
98         exit(EXIT_FAILURE);
99     }

100    /* Liste su na početku prazne */
101    int broj;
102    Cvor *lista1 = NULL;
103    Cvor *lista2 = NULL;

104    /* Ucitavanje liste */
105    while (fscanf(in1, "%d", &broj) != EOF)
106        dodaj_na_kraj_liste(&lista1, broj);

107    while (fscanf(in2, "%d", &broj) != EOF)
108        dodaj_na_kraj_liste(&lista2, broj);

109    /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
110    fclose(in1);
111    fclose(in2);
112
113
114
115
116
117
118
119

```

```

121  /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
122  objedinjavanjem listi */
123  Cvor *rezultat = objedini(&list1, &list2);
124
125  /* Ispis rezultujuce liste. */
126  ispisi_listu(rezultat);
127
128  /* Lista rezultat dobijena je prevezivanjem cvorova polaznih listi.
129  Njenim oslobadjanjem bice oslobođena sva zauzeta memorija. */
130  osloboodi_listu(&rezultat);
131
132  exit(EXIT_SUCCESS);
}

```

### Rešenje 4.8

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
5   predstavlja zagradu koja se koristi i pokazivac na sledeci cvor
6   liste */
7 typedef struct cvor {
8     char zagrada;
9     struct cvor *sledeci;
10 } Cvor;
11
12 /* Funkcija koja oslobadja memoriju zauzetu stekom */
13 void osloboodi_stek(Cvor ** stek)
14 {
15   Cvor *tekuci;
16   Cvor *pomocni;
17
18   /* Oslobadja se cvor po cvor steka */
19   tekuci = *stek;
20   while (tekuci != NULL) {
21     pomocni = tekuci->sledeci;
22     free(tekuci);
23     tekuci = pomocni;
24   }
25
26   /* Stek se proglašava praznim */
27   *stek = NULL;
28 }
29
30 /* Glavni program */
31 int main()
32 {
33   /* Stek je na pocetku prazen */
34   Cvor *stek = NULL;
35   FILE *ulaz = NULL;

```

```

37     char c;
38     Cvor *pomocni = NULL;
39
40     /* Otvaranje datoteke za citanje izraza */
41     ulaz = fopen("izraz.txt", "r");
42     if (ulaz == NULL) {
43         fprintf(stderr,
44             "Greska prilikom otvaranja datoteke izraz.txt!\n");
45         exit(EXIT_FAILURE);
46     }
47
48     /* Cita se karakter po karakter iz datoteke dok se ne dodje do
49      kraja */
50     while ((c = fgetc(ulaz)) != EOF) {
51         /* Ako je ucitana otvorena zagrada, stavlja se na stek */
52         if (c == '(' || c == '{' || c == '[') {
53             /* Alocira se memorija za novi cvor liste i proverava se
54              uspesnost alokacije */
55             pomocni = (Cvor *) malloc(sizeof(Cvor));
56             if (pomocni == NULL) {
57                 fprintf(stderr, "Greska prilikom alokacije memorije!\n");
58                 /* Oslobadja se memorija zauzeta stekom */
59                 oslobodi_stek(&stek);
60                 /* I prekida se sa izvrsavanjem programa */
61                 exit(EXIT_FAILURE);
62             }
63
64             /* Inicijalizacija polja strukture */
65             pomocni->zagrada = c;
66
67             /* Promena vrha steka */
68             pomocni->sledeci = stek;
69             stek = pomocni;
70         }
71
72         /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
73          prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
74          otvorena zagrada */
75     else {
76         if (c == ')' || c == '}' || c == ']') {
77             if (stek != NULL && ((stek->zagrada == '(' && c == ')')
78                             || (stek->zagrada == '{' && c == '}')
79                             || (stek->zagrada == '[' && c == ']')))
80             {
81                 /* Sa vrha steka se uklanja otvorena zagrada */
82                 pomocni = stek->sledeci;
83                 free(stek);
84                 stek = pomocni;
85             } else {
86                 /* Inace, zaključujemo da zagrade u izrazu nisu ispravno
87                  uparene */
88                 break;
89             }
90         }
91     }
92 }
```

```
87         }
88     }
89 }
90
91 /* Procitana je cela datoteka i treba je zatvoriti */
92 fclose(ulaz);
93
94 /* Ako je stek prazan i procitana je cela datoteka, zagrade su
95    ispravno uparene */
96 if (stek == NULL && c == EOF)
97     printf("Zagrade su ispravno uparene.\n");
98 else {
99     /* U suprotnom se zaključuje da zagrade nisu ispravno uparene */
100    printf("Zagrade nisu ispravno uparene.\n");
101    /* Oslobadja se memorija koja je ostala zauzeta stekom */
102    osloboodi_stek(&stek);
103 }
104
105 exit(EXIT_SUCCESS);
106 }
```

#### Rešenje 4.9

Datoteka 4.8: *stek.h*

```
1 #ifndef _STEK_H_
2 #define _STEK_H_
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <ctype.h>
8
9 #define MAX 100
10
11 #define OTVORENA 1
12 #define ZATVORENA 2
13
14 #define VAN_ETIKETE 0
15 #define PROCITANO_MANJE 1
16 #define U_ETIKETI 2
17
18 /* Struktura kojim se predstavlja cvor liste sadrži ime etikete i
19    pokazivac na sledeći cvor */
20 typedef struct cvor {
21     char etiketa[MAX];
22     struct cvor *sledeci;
23 } Cvor;
24
25 /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
```

```

26     adresu ili NULL ako alokacija nije bila uspesna */
27 Cvor *napravi_cvor(char *etiketa);
28
29 /* Funkcija oslobadja memoriju zauzetu stekom */
30 void osloboodi_stek(Cvor ** adresa_vrha);
31
32 /* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske pri
   33   alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0 */
34 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);
35
36 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
   37   pokazivac razlicit od NULL, tada u niz karaktera na koji on
   38   pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
   39   suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
   40   samim tim nije bilo moguce skinuti vrednost sa steka) ili 1 u
   41   suprotnom */
42 int skinji_sa_steka(Cvor ** adresa_vrha, char *etiketa);
43
44 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
   45   steka. Ukoliko je stek prazan, vraca NULL */
46 char *vrh_steka(Cvor * vrh);
47
48 /* Funkcija prikazuje stek od vrha prema dnu */
49 void prikazi_stek(Cvor * vrh);
50
51 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
   52   etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
   53   Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
   54   procita etiketa. Vraca OTVORENA, ako je procitana otvorena
   55   etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
56 int uzmi_etiketu(FILE * f, char *etiketa);
57
58 #endif

```

Datoteka 4.9: *stek.c*

```

1 #include "stek.h"
2
3 Cvor *napravi_cvor(char *etiketa)
4 {
5     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
6       alokacije */
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8     if (novi == NULL)
9         return NULL;
10
11    /* Inicijalizacija polja u novom cvoru */
12    if (strlen(etiketa) >= MAX) {
13        fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
14        etiketa[MAX - 1] = '\0';
15    }

```

```

16    strcpy(novi->etiketa, etiketa);
17    novi->sledeci = NULL;
18
19    /* Vraca se adresa novog cvora */
20    return novi;
21}
22
23 void osloboodi_stek(Cvor ** adresa_vrha)
24{
25    Cvor *pomocni;
26
27    /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
28    while (*adresa_vrha != NULL) {
29        /* Potrebno je prvo zapamtitи adresu sledeceg cvora i onda
30         oslobooditi cvor koji predstavlja vrh steka */
31        pomocni = *adresa_vrha;
32        /* Sledeci cvor je novi vrh steka */
33        *adresa_vrha = (*adresa_vrha)->sledeci;
34        free(pomocni);
35    }
36
37    /* Nakon izlaska iz petlje stek je prazen i pokazivac na adresi
38     adresa_vrha ce pokazivati na NULL. */
39}
40
41 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
42{
43    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
44    Cvor *novi = napravi_cvor(etiketa);
45    if (novi == NULL)
46        return 1;
47
48    /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
49    novi->sledeci = *adresa_vrha;
50    *adresa_vrha = novi;
51    return 0;
52}
53
54 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
55{
56    Cvor *pomocni;
57
58    /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom i
59     vraca se 0 */
60    if (*adresa_vrha == NULL)
61        return 0;
62
63    /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
64     adresu kopira etiketa sa vrha steka */
65    if (etiketa != NULL)
66        strcpy(etiketa, (*adresa_vrha)->etiketa);

```

```

68  /* Element sa vrha steka se uklanja */
69  pomocni = *adresa_vrha;
70  *adresa_vrha = (*adresa_vrha)->sledeci;
71  free(pomocni);
72
73  /* Vraca se indikator uspesno izvrsene radnje */
74  return 1;
75 }
76
77 char *vrh_steka(Cvor * vrh)
78 {
79  /* Prazan stek nema cvor koji je vrh i vraca se NULL */
80  if (vrh == NULL)
81      return NULL;
82
83  /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
84  koji je na vrhu steka. */
85  return vrh->etiketa;
86 }
87
87 void prikazi_stek(Cvor * vrh)
88 {
89  /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
90  for (; vrh != NULL; vrh = vrh->sledeci)
91      printf("<%s>\n", vrh->etiketa);
92 }
93
94 int uzmi_etiketu(FILE * f, char *etiketa)
95 {
96  int c;
97  int i = 0;
98  /* Stanje predstavlja informaciju dokle se stalo sa citanjem
99  etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
100 nije zapoceto citanje. */
101 /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
102 OTVORENA ili ZATVORENA. */
103 int stanje = VAN_ETIKETE;
104 int tip;
105
106 /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
107 to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
108 samo malim slovima. */
109 while ((c = fgetc(f)) != EOF) {
110     switch (stanje) {
111     case VAN_ETIKETE:
112         if (c == '<')
113             stanje = PROCITANO_MANJE;
114         break;
115     case PROCITANO_MANJE:
116         if (c == '/') {
117             /* Cita se zatvorena etiketa */
118             tip = ZATVORENA;

```

```

120     } else {
121         if (isalpha(c)) {
122             /* Cita se otvorena etiketa */
123             tip = OTVORENA;
124             etiketa[i++] = tolower(c);
125         }
126     /* Od sada se cita etiketa i zato se menja stanje */
127     stanje = U_ETIKETI;
128     break;
129 case U_ETIKETI:
130     if (isalpha(c) && i < MAX - 1) {
131         /* Ako je procitani karakter slovo i nije prekoracena
132            dozvoljena duzina etikete, procitani karakter se smanjuje
133            i smesta u etiketu */
134         etiketa[i++] = tolower(c);
135     } else {
136         /* Inace, staje se sa citanjem etikete. Korektno se zavrsava
137            niska koja sadrzi procitanu etiketu i vraca se njen tip */
138         etiketa[i] = '\0';
139         return tip;
140     }
141     break;
142 }
143 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
144    etiketa i vraca se EOF. */
145 return EOF;
146 }
```

Datoteka 4.10: *main.c*

```

#include "stek.h"
2
/* Glavni program */
4 int main(int argc, char **argv)
{
6     /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
      nije procitana. */
8     Cvor *vrh = NULL;
10    char etiketa[MAX];
11    int tip;
12    int uparene = 1;
13    FILE *f = NULL;
14
15    /* Ime datoteke se preuzima iz komandne linije. */
16    if (argc < 2) {
17        fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
18        exit(EXIT_FAILURE);
19    }
```

```

20  /* Datoteka se otvara za citanje */
21  if ((f = fopen(argv[1], "r")) == NULL) {
22      fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
23              argv[1]);
24      exit(EXIT_FAILURE);
25  }
26
27  /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
28  while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
29      /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
30         etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
31         potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
32         koje nemaju sadrzaj (npr link). Zbog jednostavnosti
33         pretpostavlja se da njih nema u HTML dokumentu. */
34      if (tip == OTVORENA) {
35          if (strcmp(etiketa, "br") != 0
36              && strcmp(etiketa, "hr") != 0
37              && strcmp(etiketa, "meta") != 0)
38              if (potisni_na_stek(&vrh, etiketa) == 1) {
39                  fprintf(stderr, "Neuspela alokacija za nov cvor\n");
40                  oslobodi_stek(&vrh);
41                  exit(EXIT_FAILURE);
42              }
43      }
44      /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
45         u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
46         uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
47         je taj uslov ispunjen, skida se sa steka, jer je upravo
48         zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
49         nisu pravilno uparene. */
50      else if (tip == ZATVORENA) {
51          if (vrh_steka(vrh) != NULL
52              && strcmp(vrh_steka(vrh), etiketa) == 0)
53              skinji_sa_steka(&vrh, NULL);
54          else {
55              printf("Etikete nisu pravilno uparene\n");
56              printf("(nadjena je etiketa </%s>, etiketa);"
57                  if (vrh_steka(vrh) != NULL)
58                      printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh))
59              ;
60              else
61                  printf(" koja nije otvorena)\n");
62              uparene = 0;
63              break;
64          }
65      }
66      /* Završeno je citanje i datoteka se zatvara */
67      fclose(f);
68
69      /* Ako do sada nije pronadjen pogresno uparivanje, stek bi trebalo
70         da bude prazan. Ukoliko nije, tada postoje etikete koje su

```

```

    ostale otvorene */
72 if (uparene) {
    if (vrh_steka(vrh) == NULL)
        printf("Etikete su pravilno uparene!\n");
    else {
        printf("Etikete nisu pravilno uparene\n");
        printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
        /* Oslobođaja se memorija zauzeta stekom */
        osloboodi_stek(&vrh);
    }
}
82 exit(EXIT_SUCCESS);
84 }
```

**Rešenje 4.10**Datoteka 4.11: *red.h*

```

1 #ifndef _RED_H_
2 #define _RED_H_
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #define MAX 1000
8 #define JMBG_DUZINA 14
9
10 /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11   opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
18   korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
20     Zahtev nalog;
21     struct cvor *sledeci;
22 } Cvor;
23
24 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
25   poslate adresom i vraca adresu novog cvora ili NULL ako je doslo do
26   greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
27   treba smestiti u novi cvor zbog smestanja manjeg podatka na
28   sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
29   bajtovima(B) u odnosu na strukturu Zahtev. */
30 Cvor *napravi_cvor(Zahtev * zahtev);
31 }
```

```

1  /* Funkcija prazni red oslobadjajući memoriju koji je red zauzimao */
33 void osloboodi_red(Cvor ** pocetak, Cvor ** kraj);

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

39 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
41   pokazivac razlicit od NULL, tada se u strukturu na koju on
   pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
43   suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
   suprotnom. */
45 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                     Zahtev * zahtev);

47 /* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev korisnika
49   na pocetku reda. Ukoliko je red prazan funkcija vraca NULL. */
51 Zahtev *pocetak_reda(Cvor * pocetak);

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

55 #endif

```

Datoteka 4.12: *red.c*

```

1 #include "red.h"

3 Cvor *napravi_cvor(Zahtev * zahtev)
{
5  /* Alocira se memorija za novi cvor liste i proverava uspesnost
   alokacije */
7  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9  if (novi == NULL)
    return NULL;

11  /* Inicijalizacija polja strukture */
12  novi->nalog = *zahtev;
13  novi->sledeci = NULL;

15  /* Vraca se adresa novog cvora */
16  return novi;
17 }

19 void osloboodi_red(Cvor ** pocetak, Cvor ** kraj)
{
21  Cvor *pomocni = NULL;

23  /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
24  while (*pocetak != NULL) {

```

```

25     /* Potrebno je prvo zapamtitи adresу sledeceg cvora i onda
26         osloboeditи cvor sa pocetka reda */
27     pomocni = *pocetak;
28     *pocetak = (*pocetak)->sledeci;
29     free(pomocni);
30 }
31 /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
32     treba postaviti na NULL. */
33 *kraj = NULL;
34 }

35 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
36                   Zahtev * zahtev)
37 {
38     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39     Cvor *novi = napravi_cvor(zahtev);
40     if (novi == NULL)
41         return 1;
42
43     /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
44         kraj, to je podjednako efikasno kao dodavanje na pocetak liste
45         */
46     if (*adresa_kraja != NULL) {
47         (*adresa_kraja)->sledeci = novi;
48         *adresa_kraja = novi;
49     } else {
50         /* Ako je red bio ranije prazan */
51         *adresa_pocetka = novi;
52         *adresa_kraja = novi;
53     }
54
55     /* Vraca se indikator uspesnog dodavanja */
56     return 0;
57 }

58 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
59                     Zahtev * zahtev)
60 {
61     Cvor *pomocni = NULL;
62
63     /* Ako je red prazan */
64     if (*adresa_pocetka == NULL)
65         return 0;
66
67     /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
68         zahtev koji je na pocetku reda. */
69     if (zahtev != NULL)
70         *zahtev = (*adresa_pocetka)->nalog;
71
72     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac
73         na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor
74         u redu. */
75 }
```

```

77     pomocni = *adresa_pocetka;
78     *adresa_pocetka = (*adresa_pocetka)->sledeci;
79     free(pomocni);
80
81     /* Ukoliko red nakon oslobođanja početnog cvora ostane prazan,
82      potrebno je azurirati i vrednost pokazivaca na adresi
83      adresa_kraja na NULL */
84     if (*adresa_pocetka == NULL)
85         *adresa_kraja = NULL;
86
87     return 1;
88 }
89
90 Zahtev *pocetak_reda(Cvor * pocetak)
91 {
92     /* U praznom redu nema zahteva */
93     if (pocetak == NULL)
94         return NULL;
95
96     /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
97     return &(pocetak->nalog);
98 }
99
100 void prikazi_red(Cvor * pocetak)
101 {
102     /* Prikazuje se sadrzaj reda od pocetka prema kraju */
103     for (; pocetak != NULL; pocetak = pocetak->sledeci)
104         printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
105
106     printf("\n");
107 }
```

Datoteka 4.13: *main.c*

```

2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include "red.h"
6
7 #define VREME_ZA_PAUZU 5
8
9 /* Glavni program */
10 int main(int argc, char **argv)
11 {
12     /* Red je prazan. */
13     Cvor *pocetak = NULL, *kraj = NULL;
14     Zahtev nov_zahtev;
15     Zahtev *sledeci = NULL;
16     char odgovor[3];
17     int broj_usluzenih = 0;
```

```

18  /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog JMBG
   broja i opisa potrebe usluge. */
20 printf("Sluzbenik evidentira korisnicke zahteve:\n");
21 while (1) {
22
23     /* Ucitava se JMBG */
24     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
25     if (scanf("%s", nov_zahtev.jmbg) == EOF)
26         break;
27
28     /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
       JMBG broja procitao i da bi fgetse nakon toga procitala
       ispravan red sa opisom zahteva */
29     getchar();
30
31     /* Ucitava se opis problema */
32     printf("\tOpis problema: ");
33     fgets(nov_zahtev.opis, MAX - 1, stdin);
34     /* Ako je poslednji karakter nov red, eliminise se */
35     if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
36         nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
37
38     /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
39     if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
40         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
41         osloboodi_red(&pocetak, &kraj);
42         exit(EXIT_FAILURE);
43     }
44
45
46     /* Otvaranje datoteke za dopisivanje izvestaja */
47     FILE *izlaz = fopen("izvestaj.txt", "a");
48     if (izlaz == NULL) {
49         fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
50         exit(EXIT_FAILURE);
51     }
52
53     /* Dokle god ima korisnika u redu, treba ih usluziti */
54     while (1) {
55         sledeci = pocetak_reda(pocetak);
56         /* Ako nema nikog vise u redu, prekida se petlja */
57         if (sledeci == NULL)
58             break;
59
60         printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
61         printf("i zahtevom: %s\n", sledeci->opis);
62
63         skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
64
65         broj_usluzenih++;
66
67         printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
68     }
69
70 }

```

```

70     scanf("%s", odgovor);
72     if (strcmp(odgovor, "Da") == 0)
73         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
74     else
75         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
76                 nov_zahtev.opis);
78     if (broj_usluzenih == VREME_ZA_PAUZU) {
79         printf("\nDa li je kraj smene? [Da/Ne] ");
80         scanf("%s", odgovor);
82         if (strcmp(odgovor, "Da") == 0)
83             break;
84         else
85             broj_usluzenih = 0;
86     }
88 /**
90 * Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:
92
93     while (skinisi_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
94         printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
95                nov_zahtev.jmbg);
96         printf("sa zahtevom: %s\n", nov_zahtev.opis);
97         broj_usluzenih++;
98
99         printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
100        scanf("%s", odgovor);
101        if (strcmp(odgovor, "Da") == 0)
102            dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
103        else
104            fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
105                    nov_zahtev.jmbg, nov_zahtev.opis);
106
107        if (broj_usluzenih == VREME_ZA_PAUZU) {
108            printf("\nDa li je kraj smene? [Da/Ne] ");
109            scanf("%s", odgovor);
110            if (strcmp(odgovor, "Da") == 0)
111                break;
112            else
113                broj_usluzenih = 0;
114        }
115    }
116 /**
117 /* Datoteka vise nije potrebna i treba je zatvoriti. */
118 fclose(izlaz);
119
120 /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
121 neusluzenih korisnika, u tom slucaju treba osloboziti memoriju
122 koju zauzima red sa neobradjenim zahtevima korisnika. */

```

```

122     osloboodi_red(&pocetak, &kraj);
124     exit(EXIT_SUCCESS);
}

```

### Rešenje 4.11

Datoteka 4.14: *dvostruko\_povezana\_lista.h*

```

1 #ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
#define _DVOSTRUKO_POVEZANA_LISTA_H_

3 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
5    vrednost i pokazivace na sledeci i prethodni cvor liste. */
7 typedef struct cvor {
9     int vrednost;
10    struct cvor *sledeci;
12    struct cvor *prethodni;
13 } Cvor;

15 /* Funkcija kreira cvor, vrednost novog cvora inicializuje na broj,
17    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
19    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
21 Cvor *napravi_cvor(int broj);

23 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
25    ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na
27    adresi adresa_kraja. */
29 void osloboodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);

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

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

47 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
49    novi cvor sa vrednoscu broj. */
51 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

53 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
55    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
57 int dodaj_iza(Cvor * tekuci, int broj);

59 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
61    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
63    inace vraca 0. */
65

```

```

    inace vraca 0. */
43 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                      broj);

45
/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
47   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
48   NULL u slucaju da takav cvor ne postoji u listi. */
49 Cvor *pretrazi_listu(Cvor * glava, int broj);

51 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
52   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
53   neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
54   sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
55 */
55 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

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

61
/* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
63   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
64   obrije stara glava. */
65 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                      broj);

67
/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
69   oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
70   sortirana. Azurira pokazivac na glavu liste, koji moze biti
71   promenjen ukoliko se obrije stara glava liste. */
73 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                         adresa_kraja, int broj);

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

79 /* Funkcija prikazuje cvorove liste pocevsi od kraja ka glavi liste.
80   */
81 void ispisi_listu_unazad(Cvor * kraj);
81#endif

```

Datoteka 4.15: dvostruko\_povezana\_lista.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"
4
Cvor *napravi_cvor(int broj)

```

```

6  {
7      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8         alokacije */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;
12
13     /* Inicijalizacija polja strukture */
14     novi->vrednost = broj;
15     novi->sledeci = NULL;
16
17     /* Vraca se adresa novog cvora */
18     return novi;
19 }
20
21 void osloboodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
22 {
23     Cvor *pomocni = NULL;
24
25     /* Ako lista nije prazna, onda treba oslobooditi memoriju */
26     while (*adresa_glave != NULL) {
27         /* Potrebno je prvo zapamtitи adresu sledeceg cvora i onda
28             oslobooditi memoriju cvora koji predstavlja glavu liste */
29         pomocni = (*adresa_glave)->sledeci;
30         free(*adresa_glave);
31         /* Sledeci cvor je nova glava liste */
32         *adresa_glave = pomocni;
33     }
34     /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
35         treba postaviti na NULL */
36     *adresa_kraja = NULL;
37 }
38
39 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
40                             adresa_kraja, int broj)
41 {
42     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43     Cvor *novi = napravi_cvor(broj);
44     if (novi == NULL)
45         return 1;
46
47     /* Sledbenik novog cvora je glava stare liste */
48     novi->sledeci = *adresa_glave;
49
50     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
51         treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
52         krajnji */
53     if (*adresa_glave != NULL)
54         (*adresa_glave)->prethodni = novi;
55     else
56         *adresa_kraja = novi;

```

```

58  /* Novi cvor je nova glava liste */
59  *adresa_glave = novi;
60
61  /* Vraca se indikator uspesnog dodavanja */
62  return 0;
63 }
64
65 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
66                           int broj)
67 {
68  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
69  Cvor *novi = napravi_cvor(broj);
70  if (novi == NULL)
71      return 1;
72
73  /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
74   ujedno i cela lista. Azurira se vrednost na koju pokazuju
75   adresa_glave i adresa_kraja */
76  if (*adresa_glave == NULL) {
77      *adresa_glave = novi;
78      *adresa_kraja = novi;
79  } else {
80      /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
81       sledbenik poslednjeg cvora i azurira se samo pokazivac na kraj
82       liste */
83      (*adresa_kraja)->sledeci = novi;
84      novi->prethodni = (*adresa_kraja);
85      *adresa_kraja = novi;
86  }
87
88  /* Vraca se indikator uspesnog dodavanja */
89  return 0;
90 }

91 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
92 {
93  /* U praznoj listi nema takvog mesta i vraca se NULL */
94  if (glava == NULL)
95      return NULL;
96
97  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
98   pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
99   vrednost vecu ili jednaku od vrednosti novog cvora.
100
101   Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
102   provera da li se doslo do poslednjeg cvora liste pre nego sto se
103   proveri vrednost u sledecem cvoru jer u slucaju poslednjeg,
104   sledeci ne postoji pa ni njegova vrednost. */
105  while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
106      glava = glava->sledeci;
107
108  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do

```

```

110     poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeći ima
111     vrednost vecu od broj */
112     return glava;
113 }
114
115 int dodaj_iza(Cvor * tekuci, int broj)
116 {
117     /* Kreira se novi cvor i provera se uspesnost kreiranja */
118     Cvor *novi = napravi_cvor(broj);
119     if (novi == NULL)
120         return 1;
121
122     novi->sledeci = tekuci->sledeci;
123     novi->prethodni = tekuci;
124
125     /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
126      a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
127      na ispravne adrese */
128     if (tekuci->sledeci != NULL)
129         tekuci->sledeci->prethodni = novi;
130     tekuci->sledeci = novi;
131
132     /* Vraca se indikator uspesnog dodavanja */
133     return 0;
134 }
135
136 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
137                      broj)
138 {
139     /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste
140      */
141     if (*adresa_glave == NULL) {
142         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
143         Cvor *novi = napravi_cvor(broj);
144         if (novi == NULL)
145             return 1;
146
147         /* Azuriraju se vrednosti pocetka i kraja liste */
148         *adresa_glave = novi;
149         *adresa_kraja = novi;
150
151         /* Vraca se indikator uspesnog dodavanja */
152         return 0;
153     }
154
155     /* Ukoliko je vrednost glave liste veca ili jednaka od nove
156      vrednosti onda novi cvor treba staviti na pocetak liste */
157     if ((*adresa_glave)->vrednost >= broj) {
158         return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
159     }
160
161     /* Pronazi se cvor iza koga treba uvezati novi cvor */

```

```

162     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
163     /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
164     if (dodaj_iza(pomocni, broj) == 1)
165         return 1;
166     /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
167      novi cvor poslednji u listi. */
168     if (pomocni == *adresa_kraja)
169         *adresa_kraja = pomocni->sledeci;
170
171     return 0;
172 }
173
174 Cvor *pretrazi_listu(Cvor * glava, int broj)
175 {
176     /* Obilaze se cvorovi liste */
177     for (; glava != NULL; glava = glava->sledeci)
178         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
179          se obustavlja */
180         if (glava->vrednost == broj)
181             return glava;
182
183     /* Nema trazenog broja u listi i vraca se NULL */
184     return NULL;
185 }
186
187 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
188 {
189     /* Obilaze se cvorovi liste */
190     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
191     for (; glava != NULL && glava->vrednost <= broj;
192         glava = glava->sledeci)
193         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
194          se obustavlja */
195         if (glava->vrednost == broj)
196             return glava;
197
198     /* Nema trazenog broja u listi i bice vraceno NULL */
199     return NULL;
200 }
201
202 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
203 lako realizovati jer on sadrzi pokazivace na svog sledbenika i
204 prethodnika u listi. U funkciji se bise cvor zadat argumentom
205 tekuci */
206 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
207                      tekuci)
208 {
209     /* Ako je tekuci NULL pokazivac, nema sta da se brise */
210     if (tekuci == NULL)
211         return;
212
213     /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da

```

```

138     njegov sledbenik bude sledbenik tekuceg cvora */
139     if (tekuci->prethodni != NULL)
140         tekuci->prethodni->sledeci = tekuci->sledeci;
141
142     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
143      treba da bude prethodnik tekuceg cvora */
144     if (tekuci->sledeci != NULL)
145         tekuci->sledeci->prethodni = tekuci->prethodni;
146
147     /* Ako je glava cvor koji se brise, nova glava liste ce biti
148      sledbenik stare glave */
149     if (tekuci == *adresa_glave)
150         *adresa_glave = tekuci->sledeci;
151
152     /* Ako je cvor koji se brise poslednji u listi, azurira se i
153      pokazivac na kraj liste */
154     if (tekuci == *adresa_kraja)
155         *adresa_kraja = tekuci->prethodni;
156
157     /* Oslobadja se dinamicki alociran prostor za cvor tekuci */
158     free(tekuci);
159 }
160
161 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
162 )
163 {
164     Cvor *tekuci = *adresa_glave;
165
166     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broj, takvi
167      cvorovi se brisu iz liste. */
168     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
169         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
170 }
171
172 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
173                                     adresa_kraja, int broj)
174 {
175     Cvor *tekuci = *adresa_glave;
176
177     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
178      takvi cvorovi se brisu iz liste. */
179     while ((tekuci =
180             pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
181         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
182 }
183
184 void ispisi_listu(Cvor * glava)
185 {
186     putchar('[');
187     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
188      pocetka prema kraju liste */
189     for (; glava != NULL; glava = glava->sledeci) {

```

```

264     printf("%d", glava->vrednost);
265     if (glava->sledeci != NULL)
266         printf(", ");
267 }
268
269     printf("]\n");
270 }

271 void ispisi_listu_unazad(Cvor * kraj)
272 {
273     putchar('[');
274     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
275      prema pocetku liste */
276     for (; kraj != NULL; kraj = kraj->prethodni) {
277         printf("%d", kraj->vrednost);
278         if (kraj->prethodni != NULL)
279             printf(", ");
280     }
281     printf("]\n");
282 }
```

Datoteka 4.16: *main\_a.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"

5 /* 1) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku */
9     /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
10      da bi operacije poput dodavanja na kraj liste i ispisivanja
11      liste unazad bile efikasne poput dodavanja na pocetak liste i
12      ispisivanja liste od pocetnog do poslednjeg cvora. */
13     Cvor *glava = NULL;
14     Cvor *kraj = NULL;
15     Cvor *trazeni = NULL;
16     int broj;

17     /* Testira se funkcija za dodavanja novog broja na pocetak liste */
18     printf("Unesite brojeve: (za kraj CTRL+D)\n");
19     while (scanf("%d", &broj) > 0) {
20         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
21            memorije za novi cvor. Memoriju alociranu za cvorove liste
22            treba osloboditi pre napustanja programa */
23         if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
24             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
25             osloboodi_listu(&glava, &kraj);
26             exit(EXIT_FAILURE);
27     }
```

```

29     printf("\tLista: ");
30     ispisi_listu(glava);
31 }
32
33 /* Testira se funkcija za pretragu liste */
34 printf("\nUnesite broj koji se trazi u listi: ");
35 scanf("%d", &broj);
36
37 /* Pokazivac traženi dobija vrednost rezultata pretrage */
38 traženi = pretrazi_listu(glava, broj);
39 if (traženi == NULL)
40     printf("Broj %d se ne nalazi u listi!\n", broj);
41 else
42     printf("Traženi broj %d je u listi!\n", traženi->vrednost);
43
44 /* Ispisuje se lista unazad */
45 printf("\nLista ispisana u nazad: ");
46 ispisi_listu_unazad(kraj);
47
48 /* Oslobadja se memorija zauzeta za cvorove liste */
49 oslobodi_listu(&glava, &kraj);
50
51 exit(EXIT_SUCCESS);
52 }
```

Datoteka 4.17: *main\_b.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"
4
5 /* 2) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku. */
9     Cvor *glava = NULL;
10    Cvor *kraj = NULL;
11    int broj;
12
13    /* Testira se funkcija za dodavanja novog broja na kraj liste */
14    printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
15    while (scanf("%d", &broj) > 0) {
16        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17           memorije za novi cvor. Memoriju alociranu za cvorove liste
18           treba osloboditi pre napustanja programa */
19        if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
20            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21            oslobodi_listu(&glava, &kraj);
22            exit(EXIT_FAILURE);
23        }
24        printf("\tLista: ");
```

```

15     ispisi_listu(glava);
26 }
27
28 /* Testira se funkcija za brisanje elemenata iz liste */
29 printf("\nUnesite broj koji se brise iz liste: ");
30 scanf("%d", &broj);
31
32 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
33   procitanom sa ulaza */
34 obrisi_cvor(&glava, &kraj, broj);
35
36 printf("Lista nakon brisanja: ");
37 ispisi_listu(glava);
38
39 /* Ispisuje se lista unazad */
40 printf("\nLista ispisana u nazad: ");
41 ispisi_listu_unazad(kraj);
42
43 /* Oslobadja se memorija zauzeta za cvorove liste */
44 osloboodi_listu(&glava, &kraj);
45
46 exit(EXIT_SUCCESS);
}

```

Datoteka 4.18: *main\_c.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"
4
5 /* 3) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku */
9     Cvor *glava = NULL;
10    Cvor *kraj = NULL;
11    Cvor *trazenii = NULL;
12    int broj;
13
14    /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
15       bude uredjena neopadajuce */
16    printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
17    while (scanf("%d", &broj) > 0) {
18        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
19           memorije za novi cvor. Memoriju alociranu za cvorove liste
20           treba oslobooditi pre napustanja programa */
21        if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
22            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23            osloboodi_listu(&glava, &kraj);
24            exit(EXIT_FAILURE);
25        }
}

```

```

27     printf("\tLista: ");
28     ispisi_listu(glava);
29 }
30
31 /* Testira se funkcija za pretragu liste */
32 printf("\nUnesite broj koji se trazi u listi: ");
33 scanf("%d", &broj);
34
35 /* Pokazivac traženi dobija vrednost rezultata pretrage */
36 traženi = pretrazi_listu(glava, broj);
37 if (traženi == NULL)
38     printf("Broj %d se ne nalazi u listi!\n", broj);
39 else
40     printf("Traženi broj %d je u listi!\n", traženi->vrednost);
41
42 /* Testira se funkcija za brisanje elemenata iz liste */
43 printf("\nUnesite broj koji se brise iz liste: ");
44 scanf("%d", &broj);
45
46 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
47   procitanom sa ulaza */
48 obrisi_cvor_sortirane_liste(&glava, &kraj, broj);
49
50 printf("Lista nakon brisanja: ");
51 ispisi_listu(glava);
52
53 /* Ispisuje se lista unazad */
54 printf("\nLista ispisana u nazad: ");
55 ispisi_listu_unazad(kraj);
56
57 /* Oslobadja se memorija zauzeta za cvorove liste */
58 oslobođi_listu(&glava, &kraj);
59
60 exit(EXIT_SUCCESS);
}

```

### Rešenje 4.14

Datoteka 4.19: *stabla.h*

```

#ifndef _STABLA_H_
#define _STABLA_H_ 1

/* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
6 typedef struct cvor {
7     int broj;
8     struct cvor *levo;
9     struct cvor *desno;
10} Cvor;

```

```

12 /* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
   funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
   do greske */
18 int dodaj_u_stablo(Cvor ** adresa_korena, int broj);

20 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
22 Cvor *pretrazi_stablo(Cvor * koren, int broj);

24 /* e) Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
   stablu */
26 Cvor *pronadji_najmanji(Cvor * koren);

28 /* f) Funkcija koja pronalazi cvor koji sadrzi najvecu vrednost u
   stablu */
30 Cvor *pronadji_najveci(Cvor * koren);

32 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
34 void obrisi_element(Cvor ** adresa_korena, int broj);

36 /* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
   postablo - Koren - Desno podstablo ) */
38 void ispisi_stablo_infiksno(Cvor * koren);

40 /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
   Levo podstablo - Desno podstablo ) */
42 void ispisi_stablo_prefiksno(Cvor * koren);

44 /* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
   podstablo - Desno postablo - Koren) */
46 void ispisi_stablo_postfiksno(Cvor * koren);

48 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
50 void oslobodi_stablo(Cvor ** adresa_korena);

52 #endif

```

Datoteka 4.20: *stabla.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 Cvor *napravi_cvor(int broj)
6 {
7     /* Alocira se memorija za novi cvor i proverava se uspesnost
8     alokacije. */

```

```

10    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
11    if (novi == NULL)
12        return NULL;
13
14    /* Inicijalizuju se polja novog cvora. */
15    novi->broj = broj;
16    novi->levo = NULL;
17    novi->desno = NULL;
18
19    /* Vraca se adresa novog cvora. */
20    return novi;
21}
22
23int dodaj_u_stablo(Cvor **adresa_korena, int broj)
24{
25    /* Ako je stablo prazno */
26    if (*adresa_korena == NULL) {
27
28        /* Kreira se novi cvor */
29        Cvor *novi_cvor = napravi_cvor(broj);
30
31        /* Proverava se uspesnost kreiranja */
32        if (novi_cvor == NULL) {
33
34            /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
35             */
36            return 1;
37        }
38        /* Inace ... */
39
40        /* Novi cvor se proglašava korenom stabla */
41        *adresa_korena = novi_cvor;
42
43        /* I vraca se indikator uspesnosti kreiranja */
44        return 0;
45    }
46
47    /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati
48     broj */
49
50    /* Ako je zadata vrednost manja od vrednosti korena */
51    if (broj < (*adresa_korena)->broj)
52
53        /* Broj se dodaje u levo podstablo */
54        return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
55
56    else
57        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
58         dodaje u desno podstablo */
59        return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
60}

```

```

62 Cvor *pretrazi_stablo(Cvor * koren, int broj)
63 {
64     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
65     if (koren == NULL)
66         return NULL;
67
68     /* Ako je trazena vrednost sadrzana u korenu */
69     if (koren->broj == broj) {
70
71         /* Prekidamo pretragu */
72         return koren;
73     }
74
75     /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
76     if (broj < koren->broj)
77
78         /* Pretraga se nastavlja u levom podstablu */
79         return pretrazi_stablo(koren->levo, broj);
80
81     else
82         /* U suprotnom, pretraga se nastavlja u desnom podstablu */
83         return pretrazi_stablo(koren->desno, broj);
84 }
85
86 Cvor *pronadji_najmanji(Cvor * koren)
87 {
88
89     /* Ako je stablo prazno, prekida se pretraga */
90     if (koren == NULL)
91         return NULL;
92
93     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
94      levo od njega */
95
96     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
97      najmanju vrednost */
98     if (koren->levo == NULL)
99         return koren;
100
101    /* Inace, pretragu treba nastaviti u levom podstablu */
102    return pronadji_najmanji(koren->levo);
103 }
104
105 Cvor *pronadji_najveci(Cvor * koren)
106 {
107
108     /* Ako je stablo prazno, prekida se pretraga */
109     if (koren == NULL)
110         return NULL;
111
112     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
113      desno od njega */
114 }
```

```

114  /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
115   najvecu vrednost */
116  if (koren->desno == NULL)
117    return koren;
118
119  /* Inace, pretragu treba nastaviti u desnom podstablu */
120  return pronadji_najveci(koren->desno);
121 }
122
123 void obrisi_element(Cvor ** adresa_korena, int broj)
124 {
125   Cvor *pomocni_cvor = NULL;
126
127   /* Ako je stablo prazno, brisanje nije primenljivo */
128   if (*adresa_korena == NULL)
129     return;
130
131   /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
132    stabla, ona se eventualno nalazi u levom podstablu, pa treba
133    rekurzivno primeniti postupak na levo podstablu. Koren ovako
134    modifikovanog stabla je nepromenjen. */
135   if (broj < (*adresa_korena)->broj) {
136     obrisi_element(&(*adresa_korena)->levo, broj);
137     return;
138   }
139
140   /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
141    stabla, ona se eventualno nalazi u desnom podstablu pa treba
142    rekurzivno primeniti postupak na desno podstablu. Koren ovako
143    modifikovanog stabla je nepromenjen. */
144   if ((*adresa_korena)->broj < broj) {
145     obrisi_element(&(*adresa_korena)->desno, broj);
146     return;
147   }
148
149   /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
150    jednaka broju koji se brise (tj. slucaj kada treba obrisati
151    koren) */
152
153   /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
154    prazno stablo (vraca se NULL) */
155   if ((*adresa_korena)->levo == NULL
156       && (*adresa_korena)->desno == NULL) {
157     free(*adresa_korena);
158     *adresa_korena = NULL;
159     return;
160   }
161
162   /* Ako koren ima samo levog sina, tada se brisanje vrsti tako sto se
163    brise koren, a novi koren postaje levi sin */
164   if ((*adresa_korena)->levo != NULL

```

```

166     && (*adresa_korena)->desno == NULL) {
167     pomocni_cvor = (*adresa_korena)->levo;
168     free(*adresa_korena);
169     *adresa_korena = pomocni_cvor;
170     return;
171 }
172 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako što
173  se brise koren, a novi koren postaje desni sin */
174 if ((*adresa_korena)->desno != NULL
175     && (*adresa_korena)->levo == NULL) {
176     pomocni_cvor = (*adresa_korena)->desno;
177     free(*adresa_korena);
178     *adresa_korena = pomocni_cvor;
179     return;
180 }
181 /* Slučaj kada koren ima oba sina - najpre se potrazi sledbenik
182  korena (u smislu poretka) u stablu. To je upravo po vrednosti
183  najmanji cvor u desnom podstablju. On se može pronaci npr.
184  funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
185  vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
186  broj koji se brise). Zatim se prosto rekurzivno pozove funkcija
187  za brisanje na desno podstablo. S obzirom da u njemu treba
188  obrisati najmanji element, a on zasigurno ima najviše jednog
189  potomka, jasno je da će njegovo brisanje biti obavljen na jedan
190  od jednostavnijih nacija koji su gore opisani. */
191 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
192 (*adresa_korena)->broj = pomocni_cvor->broj;
193 pomocni_cvor->broj = broj;
194 obrisi_element(&(*adresa_korena)->desno, broj);
195 }
196 }
197 void ispisi_stablo_infiksno(Cvor * koren)
198 {
199 /* Ako stablo nije prazno */
200 if (koren != NULL) {
201
202     /* Prvo se ispisuju svi cvorovi levo od korena */
203     ispisi_stablo_infiksno(koren->levo);
204
205     /* Zatim se ispisuje vrednost u korenu */
206     printf("%d ", koren->broj);
207
208     /* Na kraju se ispisuju cvorovi desno od korena */
209     ispisi_stablo_infiksno(koren->desno);
210 }
211 }
212 }
213 void ispisi_stablo_prefiksno(Cvor * koren)
214 {
215 /* Ako stablo nije prazno */

```

```

218 if (koren != NULL) {
219     /* Prvo se ispisuje vrednost u korenu */
220     printf("%d ", koren->broj);
221
222     /* Zatim se ispisuju svi cvorovi levo od korena */
223     ispisi_stablo_prefiksno(koren->levo);
224
225     /* Na kraju se ispisuju svi cvorovi desno od korena */
226     ispisi_stablo_prefiksno(koren->desno);
227 }
228
229 void ispisi_stablo_postfiksno(Cvor * koren)
230 {
231     /* Ako stablo nije prazno */
232     if (koren != NULL) {
233
234         /* Prvo se ispisuju svi cvorovi levo od korena */
235         ispisi_stablo_postfiksno(koren->levo);
236
237         /* Zatim se ispisuju svi cvorovi desno od korena */
238         ispisi_stablo_postfiksno(koren->desno);
239
240         /* Na kraju se ispisuje vrednost u korenu */
241         printf("%d ", koren->broj);
242     }
243 }
244
245 void osloboodi_stablo(Cvor ** adres_a_korena)
246 {
247     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
248     if (*adres_a_korena == NULL)
249         return;
250
251     /* Inace ... */
252     /* Oslobadja se memorija zauzeta levim podstablom */
253     osloboodi_stablo(&(*adres_a_korena)->levo);
254
255     /* Oslobadja se memorija zauzeta desnim podstablom */
256     osloboodi_stablo(&(*adres_a_korena)->desno);
257
258     /* Oslobadja se memorija zauzeta korenom */
259     free(*adres_a_korena);
260
261     /* Proglasava se stablo praznim */
262     *adres_a_korena = NULL;
263 }

```

Datoteka 4.21: *main.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"

5 int main()
{
7     Cvor *koren;
8     int n;
9     Cvor *trazeni_cvor;

11    /* Proglasava se stablo praznim */
12    koren = NULL;
13
15    /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
16       dodavanja */
17    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
18    while (scanf("%d", &n) != EOF) {
19        if (dodaj_u_stablo(&koren, n) == 1) {
20            fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
21            oslobodi_stablo(&koren);
22            return 0;
23        }
24    }

25    /* Generisu se trazeni ispisi: */
26    printf("\nInfiksni ispis: ");
27    ispisi_stablo_infiksno(koren);
28    printf("\nPrefiksni ispis: ");
29    ispisi_stablo_prefiksno(koren);
30    printf("\nPostfiksni ispis: ");
31    ispisi_stablo_postfiksno(koren);

32    /* Demonstrira se rad funkcije za pretragu */
33    printf("\nTrazi se broj: ");
34    scanf("%d", &n);
35    trazeni_cvor = pretrazi_stablo(koren, n);
36    if (trazeni_cvor == NULL)
37        printf("Broj se ne nalazi u stablu!\n");
38
39    else
40        printf("Broj se nalazi u stablu!\n");

41    /* Demonstrira se rad funkcije za brisanje */
42    printf("Brise se broj: ");
43    scanf("%d", &n);
44    obrisi_element(&koren, n);
45    printf("Rezultujuce stablo: ");
46    ispisi_stablo_infiksno(koren);
47    printf("\n");

48    /* Oslobadja se memorija zauzeta stablom */
49
50
51    /* Oslobadja se memorija zauzeta stablom */

```

```

53     oslobodi_stablo(&koren);
54
55 }

```

### Rešenje 4.15

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 #define MAX 50
7
8 /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
9
10 typedef struct cvor {
11     char *rec;
12     int brojac;
13     struct cvor *levo;
14     struct cvor *desno;
15 } Cvor;
16
17 /* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec)
19 {
20     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
21     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
22     if (novi_cvor == NULL)
23         return NULL;
24
25     /* Alocira se memorija za zadatu rec: potrebno je rezervisati
       memoriju za svaki karakter reci uključujući i terminirajući nulu
       */
26     novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
27     if (novi_cvor->rec == NULL) {
28         free(novi_cvor);
29         return NULL;
30     }
31
32     /* Inicijalizuju se polja u novom cvoru */
33     strcpy(novi_cvor->rec, rec);
34     novi_cvor->brojac = 1;
35     novi_cvor->levo = NULL;
36     novi_cvor->desno = NULL;
37
38     /* Vraca se adresa novog cvora */
39     return novi_cvor;
40 }
41
42
43
44

```

```

46  /* Funkcija koja dodaje novu rec u stablo - ukoliko je dodavanje
47    uspesno povratna vrednost je 0, u suprotnom povratna vrednost je 1
48 */
49 int dodaj_u_stablo(Cvor ** adresu_korena, char *rec)
50 {
51     /* Ako je stablo prazno */
52     if (*adresu_korena == NULL) {
53         /* Kreira se cvor koji sadrzi zadatu rec */
54         Cvor *novi_cvor = napravi_cvor(rec);
55         /* Proverava se uspesnost kreiranja novog cvora */
56         if (novi_cvor == NULL) {
57             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
58             */
59             return 1;
60         }
61         /* Inace... */
62         /* Novi cvor se proglašava korenom stabla */
63         *adresu_korena = novi_cvor;
64
65         /* I vraca se indikator uspesnog dodavanja */
66         return 0;
67     }
68
69     /* Ako stablo nije prazno, trazi odgovarajuca pozicija za novu rec
70     */
71
72     /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
73      levo podstablo */
74     if (strcmp(rec, (*adresu_korena)->rec) < 0)
75         return dodaj_u_stablo(&(*adresu_korena)->levo, rec);
76
77     else
78         /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
79          desno podstablo */
80     if (strcmp(rec, (*adresu_korena)->rec) > 0)
81         return dodaj_u_stablo(&(*adresu_korena)->desno, rec);
82
83     else
84         /* Ako je rec jednaka reci u korenu, uvecava se njen broj
85          pojavljivanja */
86         (*adresu_korena)->brojac++;
87     }
88
89     /* Funkcija koja oslobođa memoriju zauzetu stablom */
90 void osloboodi_stablo(Cvor ** adresu_korena)
91 {
92     /* Ako je stablo prazno, nepotrebno je oslobođati memoriju */
93     if (*adresu_korena == NULL)
94         return;
95
96     /* Inace ... */
97     /* Oslobođaja se memorija zauzeta levim podstablom */

```

```

96    osloboodi_stablo(&(*adresa_korena)->levo);

98    /* Oslobadja se memorija zauzeta desnim podstablom */
100   osloboodi_stablo(&(*adresa_korena)->desno);

102   /* Oslobadja se memorija zauzeta korenom */
103   free((*adresa_korena)->rec);
104   free(*adresa_korena);

106   /* Stablo se proglašava praznim */
107   *adresa_korena = NULL;
108 }

110   /* Funkcija koja pronalazi cvor koji sadrži najfrekventniju rec (rec
111    sa najvećim brojem pojavljivanja) */
112 Cvor *nadji_najfrekventniju_rec(Cvor * koren)
113 {
114     Cvor *max, *max_levo, *max_desno;

116     /* Ako je stablo prazno, prekida se sa pretragom */
117     if (koren == NULL)
118         return NULL;

120     /* Pronalazi se najfrekventnija rec u levom podstablu */
121     max_levo = nadji_najfrekventniju_rec(koren->levo);

122     /* Pronalazi se najfrekventnija rec u desnom podstablu */
123     max_desno = nadji_najfrekventniju_rec(koren->desno);

125     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
126      podstabla, korena i desnog podstabla */
127     max = koren;
128     if (max_levo != NULL && max_levo->brojac > max->brojac)
129         max = max_levo;
130     if (max_desno != NULL && max_desno->brojac > max->brojac)
131         max = max_desno;

133     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
134     return max;
135 }

137   /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
138    pracene brojem pojavljivanja */
139 void prikazi_stablo(Cvor * koren)
140 {
141     /* Ako je stablo prazno, zavrsava se sa ispisom */
142     if (koren == NULL)
143         return;

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

```

```

148     /* Zatim rec iz korena */
150     printf("%s: %d\n", koren->rec, koren->brojac);
152     /* I nastavlja se sa ispisom reci iz desnog podstabla */
153     prikazi_stablo(koren->desno);
154 }
155
156 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
157 niz rec. Maksimalna duzina reci je odredjena argumentom max.
158 Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
159 suprotnom. Rec je niz malih ili velikih slova. */
160 int procitaj_rec(FILE * f, char rec[], int max)
161 {
162     /* Karakter koji se cita */
163     int c;
164
165     /* Indeks pozicije na koju se smesta procitani karakter */
166     int i = 0;
167
168     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
169      nije stiglo do kraja datoteke... */
170     while (i < max - 1 && (c = fgetc(f)) != EOF) {
171         /* Proverava se da li je procitani karakter slovo */
172         if (isalpha(c))
173             /* Ako jeste, smesta se u niz - pritom se vrsti konverzija u
174              mala slova jer program treba da bude neosetljiv na razliku
175              izmedju malih i velikih slova */
176             rec[i++] = tolower(c);
177
178         else
179             /* Ako nije, proverava se da li je procitano barem jedno slovo
180               nove reci */
181             /* Ako jeste, prekida se sa citanjem */
182             if (i > 0)
183                 break;
184
185         /* U suprotnom se ide na sledecu iteraciju */
186     }
187
188     /* Dodaje se na rec terminirajuca nula */
189     rec[i] = '\0';
190
191     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
192     return i > 0 ? 0 : EOF;
193 }
194
195 int main(int argc, char **argv)
196 {
197     Cvor *koren = NULL, *max;
198     FILE *f;
199     char rec[MAX];

```

```

200  /* Provera da li je navedeno ime datoteke prilikom pokretanja
202   programa */
204   if (argc < 2) {
205     fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
206     exit(EXIT_FAILURE);
207   }
208
209   /* Priprema datoteke za citanje */
210   if ((f = fopen(argv[1], "r")) == NULL) {
211     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
212             argv[1]);
213     exit(EXIT_FAILURE);
214   }
215
216   /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
217    pretrage uz proveru uspesnosti dodavanja */
218   while (procitaj_rec(f, rec, MAX) != EOF) {
219     if (dodaj_u_stablo(&koren, rec) == 1) {
220       fprintf(stderr, "Neuspelo dodavanje reci %s\n", rec);
221       oslobodi_stablo(&koren);
222       exit(EXIT_FAILURE);
223     }
224   }
225
226   /* Posto je citanjem reci zavrseno, zatvara se datoteka */
227   fclose(f);
228
229   /* Prikazuju se sve reci iz teksta i brojevi njihovih
230    pojavljivanja. */
231   prikazi_stablo(koren);
232
233   /* Pronalazi se najfrekventnija rec */
234   max = nadji_najfrekventniju_rec(koren);
235
236   /* Ako takve reci nema... */
237   if (max == NULL)
238
239     /* Ispisuje se odgovarajuce obavestenje */
240     printf("U tekstu nema reci!\n");
241
242   else
243     /* Inace, ispisuje se broj pojavljivanja reci */
244     printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
245           max->rec, max->brojac);
246
247   /* Oslobadja se dinamicki alociran prostor za stablo */
248   oslobodi_stablo(&koren);
249
250   exit(EXIT_SUCCESS);
251 }
```

## Rešenje 4.16

```

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

6 #define MAX_IME_DATOTEKE 50
7 #define MAX_CIFARA 13
8 #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrži ime i prezime, broj
    telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
13     char ime_i_prezime[MAX_IME_I_PREZIME];
14     char telefon[MAX_CIFARA];
15     struct cvor *levo;
16     struct cvor *desno;
17 } Cvor;

18 /* Funkcija koja kreira novi cvora stabla */
19 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
20 {
21     /* Alocira se memorija za novi cvor i proverava se uspesnost
        alokacije. */
22     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
23     if (novi_cvor == NULL)
24         return NULL;

25     /* Inicijalizuju se polja novog cvora */
26     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
27     strcpy(novi_cvor->telefon, telefon);
28     novi_cvor->levo = NULL;
29     novi_cvor->desno = NULL;

30     /* Vraca se adresa novog cvora */
31     return novi_cvor;
32 }

33 /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
    povratna vrednost je 1 */
34 int
35 dodaj_u_stablo(Cvor **adresa_korena, char *ime_i_prezime,
36                  char *telefon)
37 {
38     /* Ako je stablo prazno */
39     if (*adresa_korena == NULL) {
40         /* Kreira se novi cvor */
41         Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
42         /* Proverava se uspesnost kreiranja novog cvora */
43         if (novi_cvor == NULL) {
44             /* Povrati 1 jer je stablo prazno */
45             return 1;
46         }
47         /* Ukoliko je novi cvor uspesno kreiran, postavlja ga na
            poziciju koju je zadat u parametru */
48         *adresa_korena = novi_cvor;
49     }
50 }
```

```

52     /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
53     */
54     return 1;
55 }
56 /* Inace... */
57 /* Novi cvor se proglašava korenom stabla */
58 *adresa_korena = novi_cvor;
59
60 /* I vraca se indikator uspesnog dodavanja */
61 return 0;
62 }
63
64 /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
65   unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
66   treba da bude pretrazivacko po ovom polju */
67
68 /* Ako je zadato ime i prezime leksikografski manje od imena i
69   prezimena sadrzanog u korenju, podaci se dodaju u levo podstablo
70   */
71 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
72     < 0)
73     return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
74                           telefon);
75
76 else
77     /* Ako je zadato ime i prezime leksikografski vece od imena i
78       prezimena sadrzanog u korenju, podaci se dodaju u desno
79       podstablo */
80 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
81     return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
82                           telefon);
83 }
84
85 /* Funkcija koja oslobadja memoriju zauzetu stablom */
86 void osloboodi_stablo(Cvor ** adresakorena)
87 {
88     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
89     if (*adresa_korena == NULL)
90         return;
91
92     /* Inace ... */
93     /* Oslobadja se memorija zauzeta levim podstablom */
94     osloboodi_stablo(&(*adresa_korena)->levo);
95
96     /* Oslobadja se memorija zauzeta desnim podstablom */
97     osloboodi_stablo(&(*adresa_korena)->desno);
98
99     /* Oslobadja se memorija zauzeta korenom */
100    free(*adresa_korena);
101
102    /* Stablo se proglašava praznim */
103    *adresa_korena = NULL;

```

```

102 }
104 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
105 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
106   koristiti za proveru da li je stablo lepo kreirano ili ne */
107 void prikazi_stablo(Cvor * koren)
108 {
109   /* Ako je stablo prazno, zavrsava se sa ispisom */
110   if (koren == NULL)
111     return;
112
113   /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
114   podstabla */
115   prikazi_stablo(koren->levo);
116
117   /* Zatim se ispisuju podaci iz korena */
118   printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
119
120   /* I nastavlja se sa ispisom podataka iz desnog podstabla */
121   prikazi_stablo(koren->desno);
122 }

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

```

```

154     ime_i_prezime[i++] = c;
156
157     else if (i > 0)
158         break;
159
160     /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
161        blanko karaktera */
162     ime_i_prezime[--i] = '\0';
163
164     /* I pocinje se sa citanjem broja telefona */
165     i = 0;
166
167     /* Upisuje se cifra koja je vec procitana */
168     telefon[i++] = c;
169
170     /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
171        karaktera cije prisustvo nije dozvoljeno u broju telefona */
172     while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
173         if (c == '/' || c == '-' || !isdigit(c))
174             telefon[i++] = c;
175         else
176             break;
177
178     /* Upisuje se terminirajuca nula */
179     telefon[i] = '\0';
180
181     /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
182     return !feof(f) ? 0 : EOF;
183 }
184
185 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
186 */
187 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
188 {
189     /* Ako je imenik prazan, zavrsava se sa pretragom */
190     if (koren == NULL)
191         return NULL;
192
193     /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
194        zavrsava sa pretragom */
195     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
196         return koren;
197
198     /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
199        korenu pretraga se nastavlja levo */
200     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
201         return pretrazi_imenik(koren->levo, ime_i_prezime);
202
203     else
204         /* u suprotnom, pretraga se nastavlja desno */
205         return pretrazi_imenik(koren->desno, ime_i_prezime);

```

```

206 }
208 int main(int argc, char **argv)
{
210     char ime_datoteke[MAX_IME_DATOTEKE];
211     Cvor *koren = NULL;
212     Cvor *trazeni;
213     FILE *f;
214     char ime_i_prezime[MAX_IME_I_PREZIME];
215     char telefon[MAX_CIFARA];
216     char c;
217     int i;
218
219     /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
220     printf("Unesite ime datoteke: ");
221     scanf("%s", ime_datoteke);
222     getchar();
223     if ((f = fopen(ime_datoteke, "r")) == NULL) {
224         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
225                 ime_datoteke);
226         exit(EXIT_FAILURE);
227     }
228
229     /* Citaju se podaci iz datoteke i smestanju u binarno stablo
230      pretrage uz proveru uspesnosti dodavanja */
231     while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
232         if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
233             fprintf(stderr, "Neuspelo dodavanje podataka za osobu %s\n",
234                     ime_i_prezime);
235             oslobodi_stablo(&koren);
236             exit(EXIT_FAILURE);
237         }
238
239     /* Zatvara se datoteka */
240     fclose(f);
241
242     /* Omogucava se pretraga imenika */
243     while (1) {
244         /* Ucitavaja se ime i prezime */
245         printf("Unesite ime i prezime: ");
246         i = 0;
247         while ((c = getchar()) != '\n')
248             ime_i_prezime[i++] = c;
249             ime_i_prezime[i] = '\0';
250
251         /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
252            funkcionalnost */
253         if (strcmp(ime_i_prezime, "KRAJ") == 0)
254             break;
255
256         /* Inace se ispisuje rezultat pretrage */
257         trazeni = pretrazi_imenik(koren, ime_i_prezime);

```

```

258     if (trazeni == NULL)
259         printf("Broj nije u imeniku!\n");
260     else
261         printf("Broj je: %s \n", trazeni->telefon);
262     }
263
264     /* Oslobadja se memorija zauzeta imenikom */
265     oslobodi_stablo(&koren);
266
267     exit(EXIT_SUCCESS);
268 }
```

**Rešenje 4.17**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 51
6
7 /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8    studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9    jezika i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX];
12     char prezime[MAX];
13     double uspeh;
14     double matematika;
15     double jezik;
16     struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
18 } Cvor;
19
20 /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
22                     double matematika, double jezik)
23 {
24     /* Alocira se memorija za novi cvor i proverava se uspesnost
25        alokacije. */
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;
29
30     /* Inicijalizuju se polja strukture */
31     strcpy(novi->ime, ime);
32     strcpy(novi->prezime, prezime);
33     novi->uspeh = uspeh;
34     novi->matematika = matematika;
35     novi->jezik = jezik;
36     novi->levo = NULL;
37     novi->desno = NULL;
```

```

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

43 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo -
44    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
45    povratna vrednost je 1 */
46 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
47                      double uspeh, double matematika, double jezik)
48 {
49  /* Ako je stablo prazno */
50  if (*koren == NULL) {
51    /* Kreira se novi cvor */
52    Cvor *novi_cvor =
53      napravi_cvor(ime, prezime, uspeh, matematika, jezik);
54    /* Proverava se uspesnost kreiranja novog cvora */
55    if (novi_cvor == NULL) {
56      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
57      */
58      return 1;
59    }
60    /* Inace... */
61    /* Novi cvor se proglašava korenom stabla */
62    *koren = novi_cvor;
63
64    /* I vraca se indikator uspesnog dodavanja */
65    return 0;
66  }
67
68  /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
69    sortirano po ukupnom broju poena */
70  if (uspeh + matematika + jezik >
71      (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
72    return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
73                          matematika, jezik);
74  else
75    return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
76                          matematika, jezik);
77 }

79
80 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
81 void oslobodi_stablo(Cvor ** koren)
82 {
83  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
84  if (*koren == NULL)
85    return;
86
87  /* Inace ... */
88  /* Oslobadja se memorija zauzeta levim podstablom */
89  oslobodi_stablo(&(*koren)->levo);

```

```

91  /* Oslobadja se memorija zauzeta desnim podstabom */
93  oslobadji_stablo(&(*koren)->desno);
95
96  /* Oslobadja se memorija zauzeta korenom */
98  free(*koren);
99
100 /* Stablo se proglašava praznim */
101 *koren = NULL;
102 }

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

```

```

143     if (koren->matematika + koren->jezik < 10)
144         return 1 + nisu_polozili(koren->levo) +
145             nisu_polozili(koren->desno);
146
147     return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
148 }
149
150 int main(int argc, char **argv)
151 {
152     FILE *in;
153     Cvor *koren;
154     char ime[MAX], prezime[MAX];
155     double uspeh, matematika, jezik;
156
157     /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
158     in = fopen("prijemni.txt", "r");
159     if (in == NULL) {
160         fprintf(stderr,
161                 "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
162         exit(EXIT_FAILURE);
163     }
164
165     /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
166      dodavanja */
167     koren = NULL;
168     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
169                   &matematika, &jezik) != EOF) {
170         if (dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik)
171             == 1) {
172             fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
173                     prezime);
174             oslobodi_stablo(&koren);
175             exit(EXIT_FAILURE);
176         }
177     }
178
179     /* Zatvaranje datoteke */
180     fclose(in);
181
182     /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
183     stampaj(koren, 1);
184
185     /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
186      prijemni */
186     if (nisu_polozili(koren) != 0)
187         printf("-----\n");
188
189     /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
190     stampaj(koren, 0);
191
192     /* Oslobadja se memorija zauzeta stablom */

```

```

193     oslobodi_stablo(&koren);
195     exit(EXIT_SUCCESS);
}

```

### Rešenje 4.18

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_NISKA 51
6
7 /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
8    osobe, dan i mesec rodjenja i redom pokazivace na levo i desno
9    podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX_NISKA];
12     char prezime[MAX_NISKA];
13     int dan;
14     int mesec;
15     struct cvor_stabla *levo;
16     struct cvor_stabla *desno;
17 } Cvor;
18
19 /* Funkcija koja kreira novi cvor */
20 Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
21 {
22     /* Alocira se memorija */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
26
27     /* Inicijalizuju se polja strukture */
28     strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
30     novi->dan = dan;
31     novi->mesec = mesec;
32     novi->levo = NULL;
33     novi->desno = NULL;
34
35     /* Vraca se adresa novog cvora */
36     return novi;
37 }
38
39 /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
40    uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
41    dodavanje uspesno povratna vrednost je 0, u suprotnom povratna
42    vrednost je 1 */
43 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
44                     int dan, int mesec)

```

```

45 {
46     /* Ako je stablo prazno */
47     if (*koren == NULL) {
48
49         /* Kreira se novi cvor */
50         Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
51         /* Proverava se uspesnost kreiranja novog cvora */
52         if (novi_cvor == NULL) {
53             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
54             */
55             return 1;
56         }
57         /* Inace... */
58         /* Novi cvor se proglašava korenom stabla */
59         *koren = novi_cvor;
60
61         /* I vraca se indikator uspesnog dodavanja */
62         return 0;
63     }
64
65     /* Stablo se uređuje po mesecu, a zatim po danu u okviru istog
66      meseca */
67     if (mesec < (*koren)->mesec)
68         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
69     else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
70         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
71     else
72         return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec);
73 }
74
75 /* Funkcija vrsti pretragu stabla i vraca cvor sa traženim datumom */
76 Cvor *pretrazi(Cvor * koren, int dan, int mesec)
77 {
78     /* Stablo je prazno, obustavlja se pretraga */
79     if (koren == NULL)
80         return NULL;
81
82     /* Ako je traženi datum u korenu */
83     if (koren->dan == dan && koren->mesec == mesec)
84         return koren;
85
86     /* Ako je mesec traženog datuma manji od meseca sadrzanog u korenu
87      ili ako su meseci isti ali je dan traženog datuma manji od
88      aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
89      svakako proverava da li leva grana postoji - ako ne postoji
90      treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
91      */
92     if (mesec < koren->mesec
93         || (mesec == koren->mesec && dan < koren->dan)) {
94         if (koren->levo == NULL)
95             return koren;

```

```

95     else
96         return pretrazi(koren->levo, dan, mesec);
97     }
98
99     /* Inace se nastavlja pretraga u desnom delu */
100    return pretrazi(koren->desno, dan, mesec);
101}
102
103/* Funkcija koja pronalazi najmanji datum u stablu */
104Cvor *pronadji_najmanji_datum(Cvor * koren)
105{
106    /* Stablo je prazno, obustavlja se pretraga */
107    if (koren == NULL)
108        return NULL;
109
110    /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
111     sadrzi najmanji datum */
112    if (koren->levo == NULL)
113        return koren;
114    else
115        /* Inace, trazimo manji datum u levom podstablu */
116        return pronadji_najmanji_datum(koren->levo);
117}
118
119/* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
120 */
121void datum_u_nisku(int dan, int mesec, char datum[])
122{
123    if (dan < 10) {
124        datum[0] = '0';
125        datum[1] = dan + '0';
126    } else {
127        datum[0] = dan / 10 + '0';
128        datum[1] = dan % 10 + '0';
129    }
130    datum[2] = '.';
131
132    if (mesec < 10) {
133        datum[3] = '0';
134        datum[4] = mesec + '0';
135    } else {
136        datum[3] = mesec / 10 + '0';
137        datum[4] = mesec % 10 + '0';
138    }
139    datum[5] = '.';
140    datum[6] = '\0';
141}
142
143/* Funkcija koja oslobadja memoriju zauzetu stablom */
144void osloboodi_stablo(Cvor ** adresa_korena)
145{
146    /* Stablo je prazno */

```

```

147     if (*adresa_korena == NULL)
148         return;
149
150     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
151     if ((*adresa_korena)->levo)
152         osloboodi_stablo(&(*adresa_korena)->levo);
153
154     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
155     if ((*adresa_korena)->desno)
156         osloboodi_stablo(&(*adresa_korena)->desno);
157
158     /* Oslobadja se memorija zauzeta korenom */
159     free(*adresa_korena);
160
161     /* Proglasava se stablo praznim */
162     *adresa_korena = NULL;
163 }
164
165 int main(int argc, char **argv)
166 {
167     FILE *in;
168     Cvor *koren;
169     Cvor *slavljenik;
170     char ime[MAX_NISKA], prezime[MAX_NISKA];
171     int dan, mesec;
172     char datum[7];
173
174     /* Provera da li je zadato ime ulazne datoteke */
175     if (argc < 2) {
176         /* Ako nije, ispisuje se poruka i prekida se sa izvrsavanjem
177            programa */
178         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
179         exit(EXIT_FAILURE);
180     }
181
182     /* Inace, priprema se datoteka za citanje */
183     in = fopen(argv[1], "r");
184     if (in == NULL) {
185         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
186                 argv[1]);
187         exit(EXIT_FAILURE);
188     }
189
190     /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja
191        */
192     koren = NULL;
193     while (fscanf(
194             (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
195             if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
196                 fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
197                         prezime);
198                 osloboodi_stablo(&koren);
199             }
200         }
201     }
202 }
```

```

197     exit(EXIT_FAILURE);
198 }
199
200 /* Datoteka se zatvara */
201 fclose(in);
202
203 /* Omogucuje se pretraga podataka */
204 while (1) {
205
206     /* Ucitava se novi datum */
207     printf("Unesite datum: ");
208     if (scanf("%d.%d.", &dan, &mesec) == EOF)
209         break;
210
211     /* Pretrazuje se stablo */
212     slavljenik = pretrazi(koren, dan, mesec);
213
214     /* Ispisuju se pronadjeni podaci */
215
216     /* Ako slavljenik nije pronadjen, to moze znaci da: */
217     /* 1. drvo je prazno */
218     if (slavljenik == NULL && koren == NULL) {
219         printf("Nema podataka o ovom ni o sledecem rođendanu.\n");
220         continue;
221     }
222     /* 2. posle datuma koji je unesen, nema podataka u stablu - u
223      ovom slučaju se pretraga vrši pocevsi od naredne godine i
224      ispisuje se najmanji datum */
225     if (slavljenik == NULL) {
226         slavljenik = pronadji_najmanji_datum(koren);
227         datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
228         printf("Slavljenik: %s %s %s\n", slavljenik->ime,
229               slavljenik->prezime, datum);
230         continue;
231     }
232
233     /* Ako je slavljenik pronadjen, razlikuju se slučajevi: */
234     /* 1. Pronadjeni su tacni podaci */
235     if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
236         printf("Slavljenik: %s %s\n", slavljenik->ime,
237               slavljenik->prezime);
238         continue;
239     }
240
241     /* 2. Pronadjeni su podaci o prvom sledecem rođendanu */
242     datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
243     printf("Slavljenik: %s %s %s\n", slavljenik->ime,
244           slavljenik->prezime, datum);
245 }
246
247 /* Oslobadja se memorija zauzeta stablom */
248 osloboodi_stablo(&koren);

```

```

249     exit(EXIT_SUCCESS);
251 }
```

### Rešenje 4.19

```

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

4 /* Uključuje se biblioteka za rad sa stablima */
# include "stabla.h"
6
/* Funkcija koja proverava da li su dva stabla koja sadrže cele
8 brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
odnosno 0 ako nisu */
10 int identitet(Cvor * koren1, Cvor * koren2)
{
12     /* Ako su oba stabla prazna, jednaka su */
13     if (koren1 == NULL && koren2 == NULL)
14         return 1;
16     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednak */
17     if (koren1 == NULL || koren2 == NULL)
18         return 0;
20     /* Ako su oba stabla neprazna i u korenu se nalaze razlike
      vrednosti, može se zaključiti da se razlikuju */
22     if (koren1->broj != koren2->broj)
23         return 0;
24     /* Inace, proverava se da li vazi i jednakost levih i desnih
      podstabala */
26     return (identitet(koren1->levo, koren2->levo)
27             && identitet(koren1->desno, koren2->desno));
28 }
30
int main()
31 {
32     int broj;
33     Cvor *koren1, *koren2;
35
/* Ucitavaju se elementi prvog stabla */
36     koren1 = NULL;
37     printf("Prvo stablo: ");
38     scanf("%d", &broj);
39     while (broj != 0) {
40         if (dodaj_u_stablo(&koren1, broj) == 1) {
41             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
42             osloboodi_stablo(&koren1);
43             return 0;
44     }
```

```

46     scanf("%d", &broj);
47 }
48
49 /* Ucitavaju se elementi drugog stabla */
50 koren2 = NULL;
51 printf("Drugo stablo: ");
52 scanf("%d", &broj);
53 while (broj != 0) {
54     if (dodaj_u_stablo(&koren2, broj) == 1) {
55         fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
56         oslobodi_stablo(&koren2);
57         return 0;
58     }
59     scanf("%d", &broj);
60 }
61
62 /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
63 njen rezultat. */
64 if (identitet(koren1, koren2))
65     printf("Stabla jesu identicna.\n");
66 else
67     printf("Stabla nisu identicna.\n");
68
69 /* Oslobadja se memorija zauzeta stablima */
70 oslobodi_stablo(&koren1);
71 oslobodi_stablo(&koren2);
72
73 return 0;
74 }
```

### Rešenje 4.20

```

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

/* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija kreira novo stablo identично stablu koje je dato korenom.
Povratna vrednost funkcije je 0 ukoliko je kopiranje uspesno,
odnosno 1 ukoliko je doslo do greske */
int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
{
    /* Izlaz iz rekurzije */
    if (koren == NULL) {
        *duplikat = NULL;
        return 0;
    }

    /* Duplira se koren stabla i postavlja da bude koren novog stabla
    */
}
```

```

15     *duplikat = napravi_cvor(koren->broj);
16     if (*duplikat == NULL) {
17         return 1;
18     }
19
20
21     /* Rekurzivno se dupliraju levo i desno podstablo i njihove adrese
22      se cuvaju redom u pokazivacima na levo i desno podstablo korena
23      duplikata */
24     int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
25     int kopija_desno =
26         kopiraj_stablo(koren->desno, &(*duplikat)->desno);
27
28     /* Ako je uspesno duplirano i levo i desno podstablo */
29     if (kopija_levo == 0 && kopija_desno == 0)
30         /* Uspesno je duplirano i celo stablo */
31         return 0;
32     /* Inace, prijavljuje se da je doslo do greske */
33     return 1;
34
35 }
36
37 /* Funkcija izracunava uniju dva skupa predstavljena stablima -
38   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
39   Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
40   uspesno, odnosno 1 ukoliko je doslo do greske */
41 int kreiraj_uniju(Cvor ** adresu_korena1, Cvor * koren2)
42 {
43
44     /* Ako drugo stablo nije prazno */
45     if (koren2 != NULL) {
46         /* 1. Dodaje se njegov koren u prvo stablo */
47         if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
48             return 1;
49         }
50
51         /* 2. Rekurzivno se racuna unija levog i desnog podstabala drugog
52           stabla sa prvim stablom */
53         int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
54         int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);
55
56         /* Ako je unija podstabala uspesno kreirana */
57         if (unija_levo == 0 && unija_desno == 0)
58             /* Uspesno je kreirana i unija stabala */
59             return 0;
60
61         /* U suprotnom se prijavljuje da je doslo do greske */
62         return 1;
63     }
64
65     /* Ako je drugo stablo prazno, nista se ne preduzima */
66     return 0;
67 }
68
69
70

```

```

72  /* Funkcija izracunava presek dva skupa predstavljana stablima -  

73   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.  

74   Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka  

75   uspesno, odnosno 1 ukoliko je doslo do greske */  

76  int kreiraj_presek(Cvor ** adres_a_koren1, Cvor * koren2)  

77  {  

78    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */  

79    if (*adresa_koren1 == NULL)  

80      return 0;  

81  

82    /* Inace... */  

83    /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.  

84     iz levog i desnog podstabla prvog stabla brisu se svi oni  

85     elementi koji ne postoje u drugom stablu */  

86    int presek_levo = kreiraj_presek(&(*adresa_koren1)->levo, koren2);  

87    int presek_desno =  

88      kreiraj_presek(&(*adresa_koren1)->desno, koren2);  

89    if (presek_levo == 0 && presek_desno == 0) {  

90      /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on  

91       uklanja iz prvog stabla */  

92      if (pretrazi_stablo(koren2, (*adresa_koren1)->broj) == NULL)  

93        obrisi_element(adresa_koren1, (*adresa_koren1)->broj);  

94  

95      /* Presek stabala je uspesno kreiran */  

96      return 0;  

97    }  

98    /* Inece, prijavljuje se da je doslo do greske */  

99    return 1;  

100  }  

101  

102  /* Funkcija izracunava razliku dva skupa predstavljana stablima -  

103   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.  

104   Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike  

105   uspesno, odnosno 1 ukoliko je doslo do greske */  

106  int kreiraj_razliku(Cvor ** adres_a_koren1, Cvor * koren2)  

107  {  

108    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */  

109    if (*adresa_koren1 == NULL)  

110      return 0;  

111  

112    /* Inace... */  

113    /* Kreira se razlika levog i desnog podstabla sa drugim stablom,  

114     tj. iz levog i desnog podstabla prvog stabla se brisu svi oni  

115     elementi koji postoje i u drugom stablu */  

116    int razlika_levo =  

117      kreiraj_razliku(&(*adresa_koren1)->levo, koren2);  

118    int razlika_desno =  

119      kreiraj_razliku(&(*adresa_koren1)->desno, koren2);  

120    if (razlika_levo == 0 && razlika_desno == 0) {  

121      /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on  

122       uklanja se iz prvog stabla */  

123      if (pretrazi_stablo(koren2, (*adresa_koren1)->broj) != NULL)

```

```

124     obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
126     /* Razlika stabala je uspesno kreirana */
127     return 0;
128 }
129
130 /* Ineće, prijavljuje se da je doslo do greske */
131 return 1;
132 }
133
134 int main()
135 {
136     Cvor *skup1;
137     Cvor *skup2;
138     Cvor *pomocni_skup = NULL;
139     int n;
140
141     /* Ucitavaju se elementi prvog skupa */
142     skup1 = NULL;
143     printf("Prvi skup: ");
144     while (scanf("%d", &n) != EOF) {
145         if (dodaj_u_stablo(&skup1, n) == 1) {
146             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
147             osloboodi_stablo(&skup1);
148             return 0;
149         }
150     }
151
152     /* Ucitavaju se elementi drugog skupa */
153     skup2 = NULL;
154     printf("Drugi skup: ");
155     while (scanf("%d", &n) != EOF) {
156         if (dodaj_u_stablo(&skup2, n) == 1) {
157             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
158             osloboodi_stablo(&skup2);
159             return 0;
160         }
161     }
162
163     /* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
164      bi se isti mogao iskoristiti i za preostale operacije */
165     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
166         osloboodi_stablo(&skup1);
167         osloboodi_stablo(&pomocni_skup);
168         return 0;
169     }
170     if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
171         osloboodi_stablo(&pomocni_skup);
172         osloboodi_stablo(&skup2);
173         return 0;
174     }
175     printf("Unija: ");

```

```

176     ispisi_stablo_infiksno(pomocni_skup);
177     putchar('\n');

178     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
179      operacije */
180     oslobodi_stablo(&pomocni_skup);

182     /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
183      kako bi se isti mogao iskoristiti i za preostale operacije */
184     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
185         oslobodi_stablo(&skup1);
186         oslobodi_stablo(&pomocni_skup);
187         return 0;
188     }
189     if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
190         oslobodi_stablo(&pomocni_skup);
191         oslobodi_stablo(&skup2);
192         return 0;
193     }
194     printf("Presek: ");
195     ispisi_stablo_infiksno(pomocni_skup);
196     putchar('\n');

198     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
199      operacije */
200     oslobodi_stablo(&pomocni_skup);

202     /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
203      kako bi se isti mogao iskoristiti i za preostale operacije */
204     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
205         oslobodi_stablo(&skup1);
206         oslobodi_stablo(&pomocni_skup);
207         return 0;
208     }
209     if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
210         oslobodi_stablo(&pomocni_skup);
211         oslobodi_stablo(&skup2);
212         return 0;
213     }
214     printf("Razlika: ");
215     ispisi_stablo_infiksno(pomocni_skup);
216     putchar('\n');

218     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
219      operacije */
220     oslobodi_stablo(&pomocni_skup);

222     /* Oslobadja se memorija zauzeta polaznim skupovima */
223     oslobodi_stablo(&skup1);
224     oslobodi_stablo(&skup2);

226     return 0;

```

```
| }
```

### Rešenje 4.21

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje 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     /* Tekuća vrednost promenljive r je broj elemenata koji su upisani
24     u niz i na osnovu nje se može 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 što najpre elemente niza smesti u
37 stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
38 desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
39 kreiran i sortiran, a 1 ukoliko je doslo do greske.
40
41 Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
42 sto je to slučaj sa ostalim opisanim algoritmima sortiranja jer se
43 sortiranje vrši u pomocnoj dinamickoj strukturi, a ne razmenom
44 elemenata niza. */
45 int sortiraj(int a[], int n)
46 {
47     int i;
```

```

49     Cvor *koren;
50
51     /* Kreira se stablo smestanjem elemenata iz niza u stablo */
52     koren = NULL;
53     for (i = 0; i < n; i++) {
54         if (dodaj_u_stablo(&koren, a[i]) == 1) {
55             osloboodi_stablo(&koren);
56             return 1;
57         }
58     }
59     /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
60      a */
61     kreiraj_niz(koren, a);
62
63     /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
64      */
65     osloboodi_stablo(&koren);
66
67     /* Vraca se indikator uspesnog sortiranja */
68     return 0;
69 }
70
71 int main()
72 {
73     int a[MAX];
74     int n, i;
75
76     /* Ucitavaju se dimenzija i elementi niza */
77     printf("n: ");
78     scanf("%d", &n);
79     if (n < 0 || n > MAX) {
80         printf("Greska: pogresna dimenzija niza!\n");
81         return 0;
82     }
83
84     printf("a: ");
85     for (i = 0; i < n; i++)
86         scanf("%d", &a[i]);
87
88     /* Poziva se funkcija za sortiranje */
89     if (sortiraj(a, n) == 0) {
90         /* Ako je niz uspesno sortiran, ispisuje se rezultujuci niz */
91         for (i = 0; i < n; i++)
92             printf("%d ", a[i]);
93             printf("\n");
94     } else {
95         /* Inace, obavestava se korisnik da je doslo do greske */
96         printf("Greska: problem prilikom sortiranja niza!\n");
97     }
98
99     return 0;
}

```

## Rešenje 4.22

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* a) Funkcija koja izracunava broj cvorova stabla */
8 int broj_cvorova(Cvor * koren)
9 {
10    /* Ako je stablo prazno, broj cvorova je nula */
11    if (koren == NULL)
12        return 0;
13
14    /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
15       levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
16       zato sto treba racunati i koren */
17    return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
18 }
19
20 /* b) Funkcija koja izracunava broj listova stabla */
21 int broj_listova(Cvor * koren)
22 {
23    /* Ako je stablo prazno, broj listova je nula */
24    if (koren == NULL)
25        return 0;
26
27    /* Proverava se da li je tekuci cvor list */
28    if (koren->levo == NULL && koren->desno == NULL)
29        /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
30           uvecati broj listova za 1 */
31        return 1;
32
33    /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
34     */
35    return broj_listova(koren->levo) + broj_listova(koren->desno);
36 }
37
38 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
39 void pozitivni_listovi(Cvor * koren)
40 {
41    /* Slučaj kada je stablo prazno */
42    if (koren == NULL)
43        return;
44
45    /* Ako je cvor list i sadrži pozitivnu vrednost */
46    if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
47        /* Stampa se */
48        printf("%d ", koren->broj);
49
50    /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */

```

```

52     pozitivni_listovi(koren->levo);
53     pozitivni_listovi(koren->desno);
54 }
55 /* d) Funkcija koja izracunava zbir cvorova stabla */
56 int zbir_svih_cvorova(Cvor * koren)
57 {
58     /* Ako je stablo prazno, zbir cvorova je 0 */
59     if (koren == NULL)
60         return 0;
61
62     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
63      elemenata u podstablima */
64     return koren->broj + zbir_svih_cvorova(koren->levo) +
65            zbir_svih_cvorova(koren->desno);
66 }
67
68 /* e) Funkcija koja izracunava najveci element stabla */
69 Cvor *najveci_element(Cvor * koren)
70 {
71     /* Ako je stablo prazno, obustavlja se pretraga */
72     if (koren == NULL)
73         return NULL;
74
75     /* Zbog prirode pretrazivackog stabla, vrednosti vece od korena se
76      nalaze u desnom podstablu */
77
78     /* Ako desnog podstabla nema */
79     if (koren->desno == NULL)
80         /* Najveca vrednost je koren */
81         return koren;
82
83     /* Inace, najveca vrednost se trazi desno */
84     return najveci_element(koren->desno);
85 }
86
87 /* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
89 {
90     /* Dubina praznog stabla je 0 */
91     if (koren == NULL)
92         return 0;
93
94     /* Izracunava se dubina levog podstabla */
95     int dubina_levo = dubina_stabla(koren->levo);
96
97     /* Izracunava se dubina desnog podstabla */
98     int dubina_desno = dubina_stabla(koren->desno);
99
100    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
101       jer se racuna i koren */
102    return dubina_levo >

```

```

104         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
105     }
106
107     /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
108     int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
109     {
110         /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazenog
111            nivoa */
112
113         /* Ako nema vise cvorova, nema spustanja niz stablo */
114         if (koren == NULL)
115             return 0;
116
117         /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
118            rekurzivnih poziva uvecati broj cvorova za 1 */
119         if (i == 0)
120             return 1;
121
122         /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
123            */
124         return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
125             + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
126     }
127
128     /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
129     void ispis_nivo(Cvor * koren, int i)
130     {
131         /* Ideja je slicna ideji iz prethodne funkcije */
132
133         /* Nema vise cvorova, nema spustanja kroz stablo */
134         if (koren == NULL)
135             return;
136
137         /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
138         if (i == 0) {
139             printf("%d ", koren->broj);
140             return;
141         }
142         /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
143            desnom podstablu */
144         ispis_nivo(koren->levo, i - 1);
145         ispis_nivo(koren->desno, i - 1);
146     }
147
148     /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
149        stabla */
150     Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
151     {
152         /* Ako je stablo prazno, obustavlja se pretraga */
153         if (koren == NULL)
154             return NULL;

```

```

154  /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
155  if (i == 0)
156      return koren;
157
158  /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
159  Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
160
161  /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
162  Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);
163
164  /* Trazi se i vraca maksimum izracunatih vrednosti */
165  if (a == NULL && b == NULL)
166      return NULL;
167  if (a == NULL)
168      return b;
169  if (b == NULL)
170      return a;
171  return a->broj > b->broj ? a : b;
172 }
173
174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
175 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
177     /* Ako je stablo prazno, zbir je nula */
178     if (koren == NULL)
179         return 0;
180
181     /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
182     if (i == 0)
183         return koren->broj;
184
185     /* Inace, spustanje se nastavlja za jedan nivo nize i traže se sume
186      iz levog i desnog podstabla */
187     return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
188         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
189 }
190
191
192 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
193    manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
195 {
196     /* Ako je stablo prazno, zbir je nula */
197     if (koren == NULL)
198         return 0;
199
200     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
201        prirode pretrazivackog stabla treba obici i levo i desno
202        podstablo */
203     if (koren->broj <= x)
204         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
205             zbir_manjih_od_x(koren->desno, x);

```

```

206     /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
208      medju njima jedino moze biti onih koje zadovoljavaju uslov */
210      return zbir_manjih_od_x(koren->levo, x);
211  }
212
213 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_za_pretragu\n");
220         return 1;
221     }
222     int i = atoi(argv[1]);
223     int x = atoi(argv[2]);
224
225     /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
226     */
227     Cvor *koren = NULL;
228     int broj;
229     while (scanf("%d", &broj) != EOF) {
230         if (dodaj_u_stablo(&koren, broj) == 1) {
231             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
232             oslobodi_stablo(&koren);
233             return 0;
234         }
235     }
236
237     /* ispisuju se rezultati rada funkcija */
238     printf("Broj cvorova: %d\n", broj_cvorova(koren));
239     printf("Broj listova: %d\n", broj_listova(koren));
240     printf("Pozitivni listovi: ");
241     pozitivni_listovi(koren);
242     printf("\n");
243     printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
244     if (najveci_element(koren) == NULL)
245         printf("Najveci element: ne postoji\n");
246     else
247         printf("Najveci element: %d\n", najveci_element(koren)->broj);
248
249     printf("Dubina stabla: %d\n", dubina_stabla(koren));
250
251     printf("Broj cvorova na %d. nivou: %d\n", i,
252           broj_cvorova_na_itom_nivou(koren, i));
253     printf("Elementi na %d. nivou: ", i);
254     ispis_nivo(koren, i);
255     printf("\n");
256     if (najveci_element_na_itom_nivou(koren, i) == NULL)
257         printf("Nema elemenata na %d. nivou!\n", i);
258     else

```

```

258     printf("Maksimalni element na %d. nivou: %d\n", i,
259           najveci_element_na_itom_nivou(koren, i)->broj);
260
260     printf("Zbir elemenata na %d. nivou: %d\n", i,
261           zbir_cvorova_na_itom_nivou(koren, i));
262     printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
263           zbir_manjih_od_x(koren, x));
264
264     /* Oslobadja se memorija zauzeta stablom */
265     osloboodi_stablo(&koren);
266
268     return 0;
}

```

### Rešenje 4.23

```

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

/* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
    /* Dubina praznog stabla je 0 */
    if (koren == NULL)
        return 0;

    /* Izracunava se dubina levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

    /* Izracunava se dubina desnog podstabla */
    int dubina_desno = dubina_stabla(koren->desno);

    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
       jer se racuna i koren */
    return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
{
    /* Ideja je slicna ideji iz prethodne funkcije */
    /* Nema vise cvorova, nema spustanja niz stablo */
    if (koren == NULL)
        return;

    /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
    if (i == 0) {

```

```

36     printf("%d ", koren->broj);
37     return;
38 }
39 /* Inace, vrsti se spustanje za jedan nivo nize i u levom i u desnom
40    podstablu */
41 ispisi_nivo(koren->levo, i - 1);
42 ispisi_nivo(koren->desno, i - 1);
43 }
44
45 /* Funkcija koja ispisuje stablo po nivoima */
46 void ispisi_stablo_po_nivoima(Cvor * koren)
47 {
48     int i;
49
50     /* Prvo se izracunava dubina stabla */
51     int dubina;
52     dubina = dubina_stabla(koren);
53
54     /* Ispisuje se nivo po nivo stabla */
55     for (i = 0; i < dubina; i++) {
56         printf("%d. nivo: ", i);
57         ispisi_nivo(koren, i);
58         printf("\n");
59     }
60 }
61
62 int main(int argc, char **argv)
63 {
64     Cvor *koren;
65     int broj;
66
67     /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
68        uspesnosti dodavanja */
69     koren = NULL;
70     while (scanf("%d", &broj) != EOF) {
71         if (dodaj_u_stablo(&koren, broj) == 1) {
72             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
73             osloboodi_stablo(&koren);
74             return 0;
75         }
76     }
77
78     /* Ispisuje se stablo po nivoima */
79     ispisi_stablo_po_nivoima(koren);
80
81     /* Oslobadja se memorija zauzeta stablom */
82     osloboodi_stablo(&koren);
83
84     return 0;
85 }

```

## Rešenje 4.25

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija koja izracunava dubinu stabla */
8 int dubina_stabla(Cvor * koren)
9 {
10    /* Dubina praznog stabla je 0 */
11    if (koren == NULL)
12        return 0;
13
14    /* Izracunava se dubina levog podstabla */
15    int dubina_levo = dubina_stabla(koren->levo);
16
17    /* Izracunava se dubina desnog podstabla */
18    int dubina_desno = dubina_stabla(koren->desno);
19
20    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
21       jer se racuna i koren */
22    return dubina_levo >
23           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24}
25
26 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
27   stablo */
28 int avl(Cvor * koren)
29 {
30    int dubina_levo, dubina_desno;
31
32    /* Ako je stablo prazno, zaustavlja se brojanje */
33    if (koren == NULL) {
34        return 0;
35    }
36
37    /* Izracunava se dubina levog podstabla korena */
38    dubina_levo = dubina_stabla(koren->levo);
39
40    /* Izracunava se dubina desnog podstabla korena */
41    dubina_desno = dubina_stabla(koren->desno);
42
43    /* Ako je uslov za AVL stablo ispunjen */
44    if (abs(dubina_desno - dubina_levo) <= 1) {
45        /* Racuna se broj AVL cvorova u levom i desnom podstablu i
46           uvecava za jedan iz razloga sto koren ispunjava uslov */
47        return 1 + avl(koren->levo) + avl(koren->desno);
48    } else {
49        /* Inace, racuna se samo broj AVL cvorova u podstablima */
50        return avl(koren->levo) + avl(koren->desno);
51    }
52}

```

```

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 uz proveru
59      uspesnosti dodavanja */
60     koren = NULL;
61     while (scanf("%d", &broj) != EOF) {
62         if (dodaj_u_stablo(&koren, broj) == 1) {
63             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
64             osloboodi_stablo(&koren);
65             return 0;
66         }
67     }
68
69     /* Racuna se i ispisuje broj AVL cvorova */
70     printf("%d\n", avl(koren));
71
72     /* Oslobadja se memorija zauzeta stablom */
73     osloboodi_stablo(&koren);
74
75     return 0;
76 }
77 }
```

### Rešenje 4.26

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Ukljucuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
8  brojeva hip. Ideja koja ce biti implementirana u osnovi ima
9   pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
10  desnog podstabla - ako je vrednost u korenu veca od izracunatih
11  vrednosti, uoceni fragment stabla zadovoljava uslov za hip. Zato
12  ce funkcija vracati maksimalne vrednosti iz uocenog podstabala ili
13  vrednost -1 ukoliko se zakljuci da stablo nije hip. */
14 int heap(Cvor * koren)
15 {
16     int max_levo, max_desno;
17
18     /* Prazno stablo je hip - kao rezultat se vraca 0 kao najmanji
19      pozitivan broj */
20     if (koren == NULL) {
21         return 0;
22     }
23
24     max_levo = heap(koren->levi);
25     max_desno = heap(koren->desni);
26
27     if (max_levo < 0 || max_desno < 0 || koren->vrednost < max_levo || koren->vrednost < max_desno) {
28         return -1;
29     }
30
31     return max(max_levo, max_desno);
32 }
```



```
74     printf("Zadato stablo nije hip!\n");
75 } else {
76     printf("Zadato stablo je hip!\n");
77 }
78 /* Oslobadja se memorija zauzeta stablom */
79 oslobodi_stablo(&koren);
80
81     return 0;
82 }
```

# Glava 5

## Ispitni rokovi

### 5.1 Programiranje 2, praktični deo ispita, jun 2015.

**Zadatak 5.1** Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu za grešku ispisati vrednost **-1** i prekinuti izvršavanje programa.

*Test 1*

```
POZIV: ./a.out ulaz.txt
ULAZ.TXT
5
Programiranje
Matematika
12345
dInAmicNArEc
Ispit

IZLAC:
```

*Test 2*

```
Poziv: ./a.out ulaz.txt
ULAZ.TXT
2
maksimalano
poena

IZLAC:
```

*Test 3*

```
|| POZIV: ./a.out ulaz.txt
|| DATOTEKA ULAZ.TXT NE POSTOJI
|| IZLAZ:
||   -1
```

*Test 4*

```
|| POZIV: ./a.out
|| IZLAZ ZA GREŠKU:
||   -1
```

[Rešenje 5.1]

**Zadatak 5.2** Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumiraj_n (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 `sumiraj_n` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

*Test 1*

```
|| ULAZ:
||   2 8 10 3 6 14 13 7 4 0
|| IZLAZ:
||   20
```

*Test 2*

```
|| ULAZ:
||   0 50 14 5 2 4 56 8 52 7 1 0
|| IZLAZ:
||   50
```

[Rešenje 5.2]

**Zadatak 5.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

*Test 1*

```
|| ULAZ:  
4 5  
1 2 3 4 5  
-1 2 -3 4 -5  
-5 -4 -3 -2 1  
-1 0 0 0 0  
IZLAZ:  
0
```

*Test 2*

```
|| ULAZ:  
2 3  
0 0 -5  
1 2 -4  
IZLAZ:  
2
```

*Test 3*

```
|| ULAZ:  
-2  
IZLAZ ZA GREŠKU:  
-1
```

[Rešenje [5.3](#)]

## 5.2 Programiranje 2, praktični deo ispita, jul 2015.

**Zadatak 5.4** Napisati program koji kao prvi argument komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podnische `needle` u nisci `haystack`, i vraća pokazivač na početak podnische, ili `NULL` ako podniska nije pronađena.

*Test 1*

```
|| POZIV: ./a.out ulaz.txt test  
  
ULAZ.TXT  
Ovo je test primer.  
U njemu se rec test javlja  
vise puta. testtesttest  
  
IZLAZ:  
5
```

*Test 2*

```
|| POZIV: ./a.out  
  
IZLAZ ZA GREŠKU:  
-1
```

*Test 3*

```
|| POZIV: ./a.out ulaz.txt foo  
  
DATOTEKA ULAZ.TXT NE POSTOJI  
  
IZLAZ ZA GREŠKU:  
-1
```

*Test 4*

```
|| POZIV: ./a.out ulaz.txt .  
  
DATOTEKA ULAZ.TXT JE PRAZNA  
  
IZLAZ:  
0
```

[Rešenje 5.4]

**Zadatak 5.5** Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji 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 ZA GREŠKU:
-1
```

Test 3

```
DATOTEKA TROUGLOVI.TXT NE POSTOJI

IZLAZ ZA GREŠKU:
-1
```

Test 4

```
TROUGLOVI.TXT
0

IZLAZ:
```

[Rešenje 5.5]

**Zadatak 5.6** Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeva. Napisati funkciju

`int prebroj_n(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>
ÜLAZ:	ÜLAZ:
3	-1 4 5 1 7
IZLAZ:	IZLAZ:
0	0

[Rešenje 5.6]

## 5.3 Programiranje 2, praktični deo ispita, septembar 2015.

**Zadatak 5.7** Sa standardnog ulaza se učitavaju neoznačeni celi brojevi x i n. Na standardni izlaz ispisati neoznačen ceo broj koji se dobija od broja x kada se njegov binarni zapis rotira za n mesta udesno (na primer, ako je binarni zapis broja x jednak 00000000000000000000000000000001111, i ako je n=1 tada na standardni izlaz treba ispisati neožnačen broj čiji je binarni zapis jednak 10000000000000000000000000000000111).

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ:	ULAZ:	ULAZ:
6 1	15 3	31 100
IZLAZ:	IZLAZ:	IZLAZ:
3	3758096385	4026531841

<i>Test 4</i>	<i>Test 5</i>
ULAZ:	ULAZ:
4 0	0 5

<i>IZLAZ:</i>	<i>IZLAZ:</i>
4	0

[Rešenje 5.7]

**Zadatak 5.8** Napisati funkciju `int dopuni_listu(Cvor ** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Povratna vrednost funkcije treba da bude 1 ukoliko je došlo greške pri alokaciji memorije, inače 0. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

*Test 1*

	ULAZ:	
	1 2 3 4 5	
	IZLAZ:	1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00

*Test 2*

	ULAZ:	
	12	
	IZLAZ:	12.00

*Test 3*

	ULAZ:	
	prazna lista	
	IZLAZ:	

*Test 4*

	ULAZ:	
	13.3 15.8	
	IZLAZ:	13.30 14.55

[Rešenje 5.8]

**Zadatak 5.9** Sa standardnog ulaza se učitava dimenzija n kvadratne celobrojne matrice A ( $n > 0$ ), a zatim i elementi matrice A. Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje su sume brojeva u svim redovima i kolonama međusobno jednake). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati -. Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati -1 na standardni izlaz za grešku. NAPOMENA: Rešenje koristi biblioteku za rad sa matricama iz zadatka 2.19.

*Test 1*

	ULAZ:	
	4	
	1 2 3 4	
	2 1 4 3	
	3 4 2 1	
	4 3 1 2	
	IZLAZ:	10

*Test 2*

	ULAZ:	
	3	
	1 1 1	
	1 1 1	
	IZLAZ:	3

*Test 3*

	ULAZ:	
	2	
	1 1	
	IZLAZ:	-

*Test 4*

	ULAZ:	
	2	
	1 2	
	1 2	
	IZLAZ:	-

*Test 5*

	ULAZ:	
	1	
	5	

*Test 6*

	ULAZ:	
	0	
	IZLAZ ZA GREŠKU:	-1

[Rešenje 5.9]

## 5.4 Rešenja

### Rešenje 5.1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define MAKS 50

6 /* Funkcija vrsti dinamicku alokaciju memorije potrebne n linija tj.
7   n niski od kojih nijedna nije duza od MAKS karaktera. */
8 char **alociranje_memorije(int n)
{
10    char **linije = NULL;
11    int i, j;
12    /* Alocira se prostor za niz vrsti matrice */
13    linije = (char **) malloc(n * sizeof(char *));
14    /* U slucaju neuspesnog otvaranja ispisuje se -1 na stderr i
15       program zavrsava. */
16    if (linije == NULL)
17        return NULL;
18    /* Alocira se prostor za svaku vrstu matrice. Niska nije duza od
19       MAKS karaktera, a 1 se dodaje zbog terminirajuce nule. */
20    for (i = 0; i < n; i++) {
21        linije[i] = malloc((MAKS + 1) * sizeof(char));
22        /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
23           prethodno alocirani resursi, i povratna vrednost je NULL */
24        if (linije[i] == NULL) {
25            for (j = 0; j < i; j++)
26                free(linije[j]);
27            free(linije);
28            return NULL;
29        }
30    }
31    return linije;
}
32
33 char **oslobadjanje_memorije(char **linije, int n)
{
34    int i;
35    /* Oslobadja se prostor rezervisan za svaku vrstu */
36    for (i = 0; i < n; i++) {
37        free(linije[i]);
38    }
39    /* Oslobadja se memorija za niz pokazivaca na vrste */
40    free(linije);
41
42    /* Matrica postaje prazna, tj. nealocirana */
43    return NULL;
}

```

```

48 }
49
50 int main(int argc, char *argv[])
51 {
52     FILE *ulaz;
53     char **linije;
54     int i, j, n;
55
56     /* Proverava argumenata komandne linije. */
57     if (argc != 2) {
58         fprintf(stderr, "-1\n");
59         exit(EXIT_FAILURE);
60     }
61
62     /* Otvaranje datoteke cije ime je navedeno kao argument komandne
63      linije neposredno nakon imena programa koji se poziva. U slučaju
64      neuspelog otvaranja ispisuje se -1 na stderr i program završava
65      izvršavanje. */
66     ulaz = fopen(argv[1], "r");
67     if (ulaz == NULL) {
68         fprintf(stderr, "-1\n");
69         exit(EXIT_FAILURE);
70     }
71     /* Ucitavanje broja linija. */
72     fscanf(ulaz, "%d", &n);
73
74     /* Alociranje memorije na osnovu ucitanog broja linija. */
75     linije = alociranje_memorije(n);
76
77     /* U slučaju neuspesne alokacije ispisuje se -1 na stderr i program
78      završava. */
79     if (linije == NULL) {
80         fprintf(stderr, "-1\n");
81         exit(EXIT_FAILURE);
82     }
83
84     /* Ucitavanje svih n linija iz datoteke. */
85     for (i = 0; i < n; i++) {
86         fscanf(ulaz, "%s", linije[i]);
87     }
88
89     /* Ispisivanje u odgovarajućem poretku ucitane linije koje
90      zadovoljavaju kriterijum. */
91     for (i = n - 1; i >= 0; i--) {
92         if (isupper(linije[i][0])) {
93             printf("%s\n", linije[i]);
94         }
95     }
96     /* Oslobadanje memorije koja je dinamicki alocirana. */
97     linije = oslobadanje_memorije(linije, n);
98
99     /* Zatvaranje datoteku. */

```

```

100    fclose(ulaz);
      exit(EXIT_SUCCESS);
}

```

### Rešenje 5.2

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Datoteka 5.1: *main.c*

```

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

int sumiraj_n(Cvor * koren, int n)
{
    /* Ako je stablo prazno, suma je nula */
    if (koren == NULL)
        return 0;
    /* Inace ... */
    /* Ako je n jednako nula, vraca se broj iz korena */
    if (n == 0)
        return koren->broj;
    /* Inace, izracunava se suma na (n-1)-om nivou u levom podstablu,
       kao i suma na (n-1)-om nivou u desnom podstablu i vraca se zbir
       te dve izracunate vrednosti jer predstavlja zbir svih cvorova na
       n-tom nivou u pocetnom stablu */
    return sumiraj_n(koren->levo, n - 1) + sumiraj_n(koren->desno, n -
        1);
}

int main()
{
    Cvor *koren = NULL;
    int n;
    int nivo;

    /* Ucitava se vrednost nivoa */
    scanf("%d", &nivo);
    while (1) {
        scanf("%d", &n);
        /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
        if (n == 0)
            break;
        /* Ako nije, dodaje se procitani broj u stablo. */
        if (dodaj_u_stablo(&koren, n) == 1) {
            fprintf(stderr, "-1\n", n);
            osloboodi_stablo(&koren);
            exit(EXIT_FAILURE);
        }
    }
}

```

```

40     }
41 }
42 /* Ispisuje se rezultat rada trazene funkcije */
43 printf("%d\n", sumiraj_n(koren, nivo));
44 /* Oslobadja se memorija */
45 oslobodi_stablo(&koren);
46
47 exit(EXIT_SUCCESS);
48 }

```

### Rešenje 5.3

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAKS 50
4
5 /* Funkcija ucitava elemenate matrice sa standardnog ulaza */
6 void ucitaj_matricu(int m[][MAKS], int v, int k)
7 {
8     int i, j;
9     for (i = 0; i < v; i++) {
10         for (j = 0; j < k; j++) {
11             scanf("%d", &m[i][j]);
12         }
13     }
14 }
15
16 /* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
17  koja ima v vrsta */
18 int broj_negativnih_u_koloni(int m[][MAKS], int v, int k)
19 {
20     int broj_negativnih = 0;
21     int i;
22     for (i = 0; i < v; i++) {
23         if (m[i][k] < 0)
24             broj_negativnih++;
25     }
26     return broj_negativnih;
27 }
28
29 int MAKS_indeks(int m[][MAKS], int v, int k)
30 {
31     int i, j;
32     int broj_negativnih;
33     /* Inicijalizujemo na nulu indeks kolone sa maksimalnim brojem
34     negativnih (maks_indeks_kolone), kao i maksimalni broj
35     negativnih elemenata u nekoj koloni (maks_broj_negativnih) */
36     int maks_indeks_kolone = 0;
37     int maks_broj_negativnih = 0;

```

```

38     for (j = 0; j < k; j++) {
40         /* Racunamo broj negativnih u j-oj koloni */
41         broj_negativnih = broj_negativnih_u_koloni(m, v, j);
42         /* Ukoliko broj negativnih u j-toj koloni veci od trenutnog
43            maksimalnog, azuriramo trenutni maksimalni broj negativnih
44            kao i indeks kolone sa maksimalnim brojem negativnih */
45         if (maks_broj_negativnih < broj_negativnih) {
46             maks_indeks_kolone = j;
47             maks_broj_negativnih = broj_negativnih;
48         }
49     }
50     return maks_indeks_kolone;
51 }
52
53 int main()
54 {
55     int m[MAKS][MAKS];
56     int v, k;
57
58     /* Ucitavanje dimenzije matrice */
59     scanf("%d", &v);
60     if (v < 0 || v > MAKS) {
61         fprintf(stderr, "-1\n");
62         exit(EXIT_FAILURE);
63     }
64     scanf("%d", &k);
65     if (k < 0 || k > MAKS) {
66         fprintf(stderr, "-1\n");
67         exit(EXIT_FAILURE);
68     }
69     /* Ucitavanje elemenata matrice */
70     ucitaj_matricu(m, v, k);
71
72     /* Pronalazenje kolone koja sadrzi najveci broj negativnih
73        elemenata i ispisivanje trazenog rezultata */
74     printf("%d\n", MAKS_indeks(m, v, k));
75     exit(EXIT_SUCCESS);
76 }
```

### Rešenje 5.4

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define MAKS 128
5
6 int main(int argc, char **argv)
7 {
8     FILE *f;
9     int brojac = 0;
```

```

10  char linija[MAKS], *p;
12  /* Provera da li je broj argumenata komandne linije 3 */
13  if (argc != 3) {
14      fprintf(stderr, "-1\n");
15      exit(EXIT_FAILURE);
16  }
17  /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18  linije */
19  if ((f = fopen(argv[1], "r")) == NULL) {
20      fprintf(stderr, "-1\n");
21      exit(EXIT_FAILURE);
22  }
23  /* Ucitavanje iz otvorene datoteke - liniju po liniju */
24  while (fgets(linija, MAKS, f) != NULL) {
25      p = linija;
26      while (1) {
27          p = strstr(p, argv[2]);
28
29          /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
30          if (p == NULL)
31              break;
32          /* Inace se uvecava brojac */
33          brojac++;
34          /* p se pomera da bi se u sledecoj iteraciji posmatra ostatak
35             linije nakon uocene podniske */
36          p = p + strlen(argv[2]);
37      }
38  }
39  /* Zatvaranje datoteke */
40  fclose(f);
41  /* Ispisivanje vrednosti brojaca */
42  printf("%d\n", brojac);
43
44  exit(EXIT_SUCCESS);
}

```

### Rešenje 5.5

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 /* Struktura trougao */
6 typedef struct _trougao {
7     double xa, ya, xb, yb, xc, yc;
8 } trougao;
9
10 /* Funkcija racuna duzinu duzi */
11 double duzina(double x1, double y1, double x2, double y2)
12 {

```

```

13     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
14 }
15 /* Funkcija racuna povrsinu trougla koristeci Heronov obrazac */
16 double povrsina(trougao t)
17 {
18     /* Racunanje duzina stranica trougla */
19     double a = duzina(t.xb, t.yb, t.xc, t.yc);
20     double b = duzina(t.xa, t.ya, t.xc, t.yc);
21     double c = duzina(t.xa, t.ya, t.xb, t.yb);
22     /* Poluobim */
23     double s = (a + b + c) / 2;
24     /* Primena Heronovog obrasca */
25     return sqrt(s * (s - a) * (s - b) * (s - c));
26 }
27

28 /* Funkcija poredi dva trougla: ukoliko je povrsina trougla koji je
29    prvi argument funkcije manja od povrsine trougla koji je drugi
30    element funkcije funkcija vraca 1, ukoliko je veca -1, a ukoliko
31    su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
32    napisana tako da se moze proslediti funkciji qsort da se niz
33    trouglova sortira po povrsini opadajuce. */
34 int poredi(const void *a, const void *b)
35 {
36     trougao x = *(trougao *) a;
37     trougao y = *(trougao *) b;
38     double xp = povrsina(x);
39     double yp = povrsina(y);
40     if (xp < yp)
41         return 1;
42     if (xp > yp)
43         return -1;
44     return 0;
45 }
46

47 int main()
48 {
49     FILE *f;
50     int n, i;
51     trougao *niz;
52

53     /* Otvaranje datoteke ciji je naziv trouglovi.txt */
54     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
55         fprintf(stderr, "-1\n");
56         exit(EXIT_FAILURE);
57     }
58

59     /* Ucitavanje podataka o broju trouglova iz datoteke */
60     if (fscanf(f, "%d", &n) != 1) {
61         fprintf(stderr, "-1\n");
62         exit(EXIT_FAILURE);
63     }

```

```

65  /* Dinamicka alokacija memotije za niz trouglova duzine n */
67  if ((niz = malloc(n * sizeof(trougao))) == NULL) {
68      fprintf(stderr, "-1\n");
69      exit(EXIT_FAILURE);
70  }
71
72  /* Ucitavanje podataka u niz iz otvorene datoteke */
73  for (i = 0; i < n; i++) {
74      if (fscanf(f, "%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
75                  &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
76      {
77          fprintf(stderr, "-1\n");
78          exit(EXIT_FAILURE);
79      }
80
81  /* Pozivanje funkcije qsort da sortira niz na osnovu funkcije
82   * poredi */
83  qsort(niz, n, sizeof(trougao), &poredi);
84
85  /* Ispisivanje sortiranog niza na standardni izlaz */
86  for (i = 0; i < n; i++)
87      printf("%g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
88             niz[i].yb, niz[i].xc, niz[i].yc);
89
90  /* Oslobadjanje dinamicki alocirane memorije */
91  free(niz);
92
93  /* Zatvranje datoteke */
94  fclose(f);
95  exit(EXIT_SUCCESS);
}

```

### Rešenje 5.6

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Datoteka 5.2: *main.c*

```

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

/* Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
 * stablo. Funkcija vraca 1 u slucaju neuspesnog dodavanja elementa u
 * stablo, a inace 0. */
int ucitaj_stablo(Cvor ** koren)
{

```

```

10    *koren = NULL;
11    int x;
12    /* Sve dok ima brojeva na standardnom ulazu, ucitani brojevi se
13       dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
14       je i povratna vrednost iz funkcije ucitaj_stablo 1. */
15    while (scanf("%d", &x) == 1)
16        if (dodaj_u_stablo(koren, x) == 1)
17            return 1;
18    return 0;
19 }

20 /* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
21 int prebroj_n(Cvor * koren, int n)
22 {
23    /* Ukoliko je stablo prazno, rezultat je nula. Takodje, ako je n
24       negativan broj, na tom nivou nema cvorova (rezultat je nula). */
25    if (koren == NULL || n < 0)
26        return 0;
27    /* Ukoliko je n = 0, na tom nivou je samo koren. Ukoliko ima jednog
28       potomka funkcija vraca 1, inace 0 */
29    if (n == 0) {
30        if (koren->levo == NULL && koren->desno != NULL)
31            return 1;
32        if (koren->levo != NULL && koren->desno == NULL)
33            return 1;
34        return 0;
35    }
36    /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
37       (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om nivou
38       levog podstabla */
39    return prebroj_n(koren->levo, n - 1) + prebroj_n(koren->desno, n -
40                1);
41 }

42 int main()
43 {
44    Cvor *koren;
45    int n;
46    scanf("%d", &n);

47    /* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
48       oslobodja se alocirana memorija i izlazi se iz programa. */
49    if (ucitaj_stablo(&koren) == 1) {
50        fprintf(stderr, "-1\n");
51        oslobodi_stablo(&koren);
52        exit(EXIT_FAILURE);
53    }

54    /* Ispisivanje rezultata */
55    printf("%d\n", prebroj_n(koren, n));
56

57    /* Oslobadjanje dinamicki alociranog stabla */
58
59
60

```

```

62     osloboodi_stablo(&koren);
63     exit(EXIT_SUCCESS);
64 }
```

### Rešenje 5.7

```

#include <stdio.h>
2
/* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
   zapis argumenta x rotira za n mesta udesno */
4
unsigned int rotiraj(unsigned int x, unsigned int n)
6 {
7     int i;
8     unsigned int maska = 1;
9     /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
10    izgledala: 000...00001111 */
11    /* Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
12    nastavku je drugi nacin. */
13    for (i = 1; i < n; i++)
14        maska = (maska << 1) | 1;
15    /* Kada se x poremeri za n mesta udesno x >> n, poslednjih n bitova
16    binarne reprezentacije broja x ce "ispasti". Za rotaciju je
17    potrebno da se tih n bitova postavi na pocetak broja. Kreirana
18    maska omogucava da se tih n bitova izdvoji sa (maska & x), a
19    zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta uлево
20    tih n bitova postavlja na pocetak. Primenom logicke disjunkcije
21    dobija se rotirani broj. */
22    return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
23 }
24
int main()
25 {
26     unsigned int x, n;
27
28     /* Ucitavanje brojeva sa standardnog ulaza */
29     scanf("%u%u", &x, &n);
30
31     /* Ispisivanje rezultata */
32     printf("%u\n", rotiraj(x, n));
33
34     return 0;
35 }
```

### Rešenje 5.8

Datoteka 5.3: *liste.h*

```
#ifndef _LISTE_H_
```

```

2 #define _LISTE_H_ 1

4 /* Struktura koja predstavlja cvor liste */
5 typedef struct cvor {
6     double vrednost;
7     struct cvor *sledeci;
8 }
9 Cvor;

10 /* Pomocna funkcija koja kreira cvor. */
11 Cvor * napravi_cvor(double broj);

14 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
15    liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
16 void osloboidi_listu(Cvor ** adresa_glave);

18 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
19    ili NULL kao je lista prazna */
20 Cvor * pronadji_poslednji(Cvor * glava);

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

26 /* Funkcija prikazuje sve elemente liste pocev od glave ka kraju
27    liste. */
28 void ispisi_listu(Cvor * glava);

30#endif

```

Datoteka 5.4: liste.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "liste.h"

4 /* Pomocna funkcija koja kreira cvor. */
5 Cvor *napravi_cvor(double broj)
6 {
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8     if (novi == NULL)
9         return NULL;
10    /* inicializacija polja u novom cvoru */
11    novi->vrednost = broj;
12    novi->sledeci = NULL;
13    return novi;
14 }

18 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
19    ciji se pocetni cvor nalazi na adresi adresa_glave. */

```

```

20 void osloboodi_listu(Cvor ** adresa_glave)
21 {
22     Cvor *pomocni = NULL;
23     while (*adresa_glave != NULL) {
24         pomocni = (*adresa_glave)->sledeci;
25         free(*adresa_glave);
26         *adresa_glave = pomocni;
27     }
28 }
29
30 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
31 ili NULL kao je lista prazna */
32 Cvor *pronadji_poslednji(Cvor * glava)
33 {
34     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
35      funkcija vraca NULL. */
36     if (glava == NULL)
37         return NULL;
38     while (glava->sledeci != NULL)
39         glava = glava->sledeci;
40     return glava;
41 }
42
43 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
44 greske pri alokaciji memorije, inace vraca 0. */
45 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
46 {
47     Cvor *novi = napravi_cvor(broj);
48     if (novi == NULL)
49         return 1;
50     if (*adresa_glave == NULL) {
51         *adresa_glave = novi;
52         return 0;
53     }
54     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
55     poslednji->sledeci = novi;
56 }
57
58 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
59 */
60 void ispisi_listu(Cvor * glava)
61 {
62     for (; glava != NULL; glava = glava->sledeci)
63         printf("%.2lf ", glava->vrednost);
64     putchar('\n');
65 }
66
67
68
69
70

```

Datoteka 5.5: *main.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "liste.h"
4
5 /* Funkcija umeće novi cvor iza tekuceg u listi */
6 void dodaj_iza(Cvor * tekuci, Cvor * novi)
7 {
8     /* Novi cvor se dodaje iza tekuceg cvora. */
9     novi->sledeci = tekuci->sledeci;
10    tekuci->sledeci = novi;
11 }
12
13 /* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka.
14    Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace vraca
15    0. */
16 int dopuni_listu(Cvor ** adresa_glave)
17 {
18     Cvor *tekuci;
19     Cvor *novi;
20     double aritmeticka_sredina;
21     /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
22     if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
23         return 1;
24     /* Promenljiva tekuci se inicializacuje da pokazuje na pocetni
25        cvor */
26     tekuci = *adresa_glave;
27     /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
28        vrednosti u susednim cvorovims i kreira cvor sa tom vrednoscu. U
29        slucaju neupele alokacije novog cvora, funkcija vraca 1. Inace,
30        novi cvor se umeće izmedju dva cvora za koje racunata
31        aritmeticka sredina */
32     while (tekuci->sledeci != NULL) {
33         aritmeticka_sredina =
34             ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
35         novi = napravi_cvor(aritmeticka_sredina);
36         if (novi == NULL)
37             return 1;
38         /* Poziva se funkcija koja umeće novi cvor iza tekuceg cvora */
39         dodaj_iza(tekuci, novi);
40         /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
41            cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz
42            polazne liste */
43         tekuci = tekuci->sledeci;
44         tekuci = tekuci->sledeci;
45     }
46     return 0;
47 }
48 }
```

```

50 int main()
51 {
52     Cvor *glava = NULL;
53     double broj;
54
55     /* Ucitavanje se vrsti do kraja ulaza. Elementi se dodaju na kraj
56      liste! */
57     while (scanf("%lf", &broj) > 0) {
58         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
59          memorije za nov cvor. Memoriju alociranu za cvorove liste
60          treba oslobooditi. */
61         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
62             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
63             osloboodi_listu(&glava);
64             exit(EXIT_FAILURE);
65         }
66     }
67
68     /* Pozivanje funkcije da dopuni listu. Ako je funkcija vratila 1,
69      onda je bilo greske pri alokaciji memorije za nov cvor. Memoriju
70      alociranu za cvorove liste treba oslobooditi. */
71     if (dopuni_listu(&glava) == 1) {
72         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
73         osloboodi_listu(&glava);
74         exit(EXIT_FAILURE);
75     }
76
77     /* Ispisivanje liste */
78     ispisi_listu(glava);
79
80     /* Oslobadjanje liste */
81     osloboodi_listu(&glava);
82
83     exit(EXIT_SUCCESS);
84 }
```

### Rešenje 5.9

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 /* Funkcija racuna zbir elemenata v-te vrste */
6 int zbir_vrste(int **M, int n, int v)
7 {
8     int j, zbir = 0;
9     for (j = 0; j < n; j++)
10        zbir += M[v][j];
11    return zbir;
12 }
```

```

14 /* Funkcija racuna zbir elemenata k-te kolone */
15 int zbir_kolone(int **M, int n, int k)
16 {
17     int i, zbir = 0;
18     for (i = 0; i < n; i++)
19         zbir += M[i][k];
20     return zbir;
21 }
22
23 /* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
24    argument magican. Ukoliko jeste magican funkcija vraca 1, inace 0.
25    Argument funkcije zbir ce sadrzati zbir elemenata neke vrste ili
26    kolone ukoliko je kvadrat magican. */
27 int magicni_kvadrat(int **M, int n, int *zbirmagicnog)
28 {
29     int i, j;
30     int zbir = 0, zbir_pom;
31     /* Promenljivu zbir inicijalizujemo na zbir 0-te vreste */
32     zbir = zbir_vrste(M, n, 0);
33
34     /* Racunaju se zbirovi u ostalim vrstama i ako neki razlikuje od
35        vrednosti promeljive zbir funkcija vraca 1 */
36     for (i = 1; i < n; i++) {
37         zbir_pom = zbir_vrste(M, n, i);
38         if (zbir_pom != zbir)
39             return 0;
40     }
41     /* Racunaju se zbirovi u svim kolonama i ako neki razlikuje od
42        vrednosti promeljive zbir funkcija vraca 1 */
43     for (j = 0; j < n; j++) {
44         zbir_pom = zbir_kolone(M, n, j);
45         if (zbir_pom != zbir)
46             return 0;
47     }
48     /* Inace su zbirovi svih vrsta i kolona jednaki, postavlja se
49        vresnost u zbirmagicnog i funkcija vraca 1 */
50     *zbirmagicnog = zbir;
51     return 1;
52 }
53
54 int main()
55 {
56     int n, i, j;
57     int **matrica = NULL;
58     int zbir = 0, zbirmagicnog = 0;
59     scanf("%d", &n);
60
61     /* Provera da li je n strogoo pozitivan */
62     if (n <= 0) {
63         printf("-1\n");
64         exit(EXIT_FAILURE);
65     }

```

```
66  /* Dinamicka alokacija matrice dimenzije nxn */
68  matrica = alociraj_matricu(n);
69  if (matrica == NULL) {
70      printf("-1\n");
71      exit(EXIT_FAILURE);
72  }

74  /* Ucitavanje elemenata matrice sa standardnog ulaza */
75  ucitaj_matricu(matrica, n);

76  /* Ispisivanje rezultata na osnovu funkcije magicni_kvadrat */
77  if (magicni_kvadrat(matrica, n, &zbirmagicnog)) {
78      printf("%d\n", zbirmagicnog);
79  } else
80      printf("-\n");

82  /* Oslobojanje dinamicki alocirane memorije */
83  matrica = dealociraj_matricu(matrica, n);

86  exit(EXIT_SUCCESS);
}
```