

# PROGRAMIRANJE 2

ZBIRKA ZADATAKA SA REŠENJIMA

MILENA VUJOŠEVIĆ JANIČIĆ

ANA SPASIĆ	JELENA GRAOVAC
---------------	----------------

MIRKO SPASIĆ	NINA RADOJIČIĆ
-----------------	----------------

MIRKO SPASIĆ	ANĐELKA ZEČEVIĆ
-----------------	-----------------



MATEMATIČKI FAKULTET, BEOGRAD 2016.

## PROGRAMIRANJE 2

Elektronsko izdanje (2016)



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

**PROGRAMIRANJE 2  
Zbirka zadataka sa rešenjima**

**Beograd  
2016.**

Autori:

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

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

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Andelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu. Studentski trg 16, 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: Copy Centar, Beograd. Tiraž 200.

СIP Каталогизација у публикацији  
Народна библиотека Србије, Београд

004.4(075.8)(076)

004.432.2C(075.8)(076)

PROGRAMIRANJE 2 : zbirka zadataka sa rešenjima / Milena Vujošević

Jančić ... [et al.]. - Beograd : Matematički fakultet, 2016

(Beograd : Copy Centar). - VII, 361 str. ; 24 cm

Tiraž 200.

ISBN 978-86-7589-107-9

1. Вујошевић Јаничић, Милена 1980- [аутор]

a) Програмирање - Задаци b) Програмски језик "C"- Задаци

COBISS.SR-ID 221508876

©2016. Milena Vujošević Jančić, Jelena Graovac, Nina Radojičić, Ana Spasić, Mirko Spasić, Andelka Zečević

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.



# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>1</b>
1.1	Podela koda po datotekama . . . . .	1
1.2	Algoritmi za rad sa bitovima . . . . .	5
1.3	Rekurzija . . . . .	11
1.4	Rešenja . . . . .	20
<b>2</b>	<b>Pokazivači</b>	<b>69</b>
2.1	Pokazivačka aritmetika . . . . .	69
2.2	Višedimenzioni nizovi . . . . .	73
2.3	Dinamička alokacija memorije . . . . .	78
2.4	Pokazivači na funkcije . . . . .	84
2.5	Rešenja . . . . .	86
<b>3</b>	<b>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</b>	<b>Dinamičke strukture podataka</b>	<b>221</b>
4.1	Liste . . . . .	221
4.2	Stabla . . . . .	232
4.3	Rešenja . . . . .	241
<b>A</b>	<b>Ispitni rokovi</b>	<b>337</b>
A.1	Praktični deo ispita, jun 2015. . . . .	337
A.2	Praktični deo ispita, jul 2015. . . . .	339
A.3	Praktični deo ispita, septembar 2015. . . . .	341
A.4	Praktični deo ispita, januar 2016. . . . .	343
A.5	Rešenja . . . . .	345



# Predgovor

U okviru kursa *Programiranje 2* na Matematičkom fakultetu vežbaju se zadaci koji imaju za cilj da studente nauče rekurzivnom pristupu rešavanja problema, ispravnom radu sa pokazivačima i dinamički alociranom memorijom, osnovnim algoritmima pretraživanja i sortiranja, kao i radu sa dinamičkim strukturama podataka, poput listi i stabala. Zadaci koji se nalaze u ovoj zbirci predstavljaju objedinjen skup zadataka sa vežbi i praktikuma ovog kursa, kao i primere zadataka sa održanih ispita. Elektronska verzija zbirke i prpratna rešenja u elektronskom formatu, dostupna su besplatno u okviru strane kursa [www.programiranje2.matf.bg.ac.rs](http://www.programiranje2.matf.bg.ac.rs) u skladu sa navedenom licencom.

U prvom poglavlju zbirke obrađene su uvodne teme koje obuhvataju osnovne tehnike koje se koriste u rešavanju svih ostalih zadataka u zbirci: podela koda po datotekama i rekurzivni pristup rešavanju problema. Takođe, u okviru ovog poglavlja dati su i osnovni algoritmi za rad sa bitovima. Drugo poglavlje je posvećeno pokazivačima: pokazivačkoj aritmetici, višedimenzionim nizovima, dinamičkoj alokaciji memorije i radu sa pokazivačima na funkcije. Treće poglavlje obrađuje algoritme pretrage i sortiranja, a četvrto dinamičke strukture podataka: liste i stabla. Dodatak sadrži najvažnije ispitne rokove iz jedne akademske godine. Većina zadataka je rešena, a teži zadaci su obeleženi zvezdicom.

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. Zbog toga smatramo da je naš osnovni doprinos što smo objedinili, precizno formulisali, rešili i detaljno iskomentarisali sve najvažnije zadatke koji su potrebni za uspešno savlađivanje koncepata koji se obrađuju u okviru kursa. Takođe, formulacije zadataka smo obogatili primerima koji upotpunjuju razumevanje zahteva zadataka i koji omogućavaju čitaocu zbirke da proveri sopstvena rešenja. Primeri su dati u obliku testova i interakcija sa programom. Testovi su svedene prirode i obuhvataju samo jednostavne ulaze i izlaze iz programa. Interakcija sa programom obuhvata naizmeničnu interakciju čovek-računar u kojoj su ulazi i izlazi isprepletani. U zadacima koji zahtevaju



rad sa argumentima komandne linije, navedeni su i primeri poziva programa, a u zadacima koji demonstriraju rad sa datotekama, i primeri ulaznih ili izlaznih datoteka. Test primeri koji su navedeni uz ispitne zadatke u dodatku su oni koji su korišćeni za početno testiranje (koje prethodi ocenjivanju) studentskih radova na ispitima.

Neizmerno zahvaljujemo recenzentima, Gordani Pavlović Lažetić i Draganu Uroševiću, na veoma pažljivom čitanju rukopisa i na brojnim korisnim sugestijama. Takođe, zahvaljujemo studentima koji su svojim aktivnim učešćem u nastavi pomogli i doprineli uobličavanju ovog materijala.

Svi komentari i sugestije na sadržaj zbirke su dobrodošli i osećajte se slobodnim da ih pošaljete elektronskom poštom bilo kome od autora<sup>1</sup>.

*Autori*

---

<sup>1</sup>Adrese autora su: milena, jgraovac, nina, aspasic, mirko, andjelkaz, sa nastavkom @matf.bg.ac.rs

# 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 zadat njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `void ucitaj_kompleksan_broj(KompleksanBroj * z)` koja učitava kompleksan broj `z` sa standardnog ulaza.
- (c) Napisati funkciju `void ispisi_kompleksan_broj(KompleksanBroj z)` koja ispisuje kompleksan broj `z` na standardni izlaz u odgovarajućem formatu.
- (d) Napisati funkciju `float realan_deo(KompleksanBroj z)` koja vraća vrednost realnog dela broja `z`.
- (e) Napisati funkciju `float imaginaran_deo(KompleksanBroj z)` koja vraća vrednost imaginarnog dela broja `z`.
- (f) Napisati funkciju `float moduo(KompleksanBroj z)` koja vraća moduo kompleksnog broja `z`.
- (g) Napisati funkciju `KompleksanBroj konjugovan(KompleksanBroj z)` koja vraća konjugovano-kompleksni broj broja `z`.
- (h) Napisati funkciju `KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća zbir dva kompleksna broja `z1` i `z2`.

## 1 Uvodni zadaci

---

- (i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća razliku dva kompleksna broja  $z_1$  i  $z_2$ .
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća proizvod dva kompleksna broja  $z_1$  i  $z_2$ .
- (k) Napisati funkciju `float argument(KompleksanBroj z)` koja vraća argument kompleksnog broja  $z$ .

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza uneti dva kompleksna broja  $z_1$  i  $z_2$ , a zatim ispisati realni deo, imaginarni deo, moduo, konjugovano-kompleksan broj i argument broja koji se dobija kao zbir, razlika ili proizvod brojeva  $z_1$  i  $z_2$  u zavisnosti od znaka ('+', '-', '\*') koji se unosi sa standardnog ulaza.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite realni i imaginarni deo kompleksnog broja: 1 -3
(1.00 - 3.00 i)
Unesite realni 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)
Realni_deo: 2
Imaginarni_deo: -7.000000
Moduo: 7.280110
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. Napisati program koji testira ovu biblioteku. Sa standardnog ulaza uneti kompleksan broj, a zatim na standardni izlaz ispisati njegov polarni oblik.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite realni i imaginarni deo kompleksnog broja: -5 2
Polarni oblik kompleksnog broja je 5.39 * ei * 2.76
```

[Rešenje 1.2]

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

- Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20 koji je zadat nizom svojih koeficijenata tako da se na  $i$ -toj poziciji u nizu nalazi koeficijent uz  $i$ -ti stepen polinoma.
- Napisati funkciju `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz, od najvišeg ka najnižem stepenu. Ipisati samo koeficijente koji su različiti od nule.
- Napisati funkciju `Polinom ucitaj()` koja učitava polinom sa standardnog ulaza. Za polinom najpre uneti stepen, a zatim njegove koeficijente.
- Napisati funkciju `double izracunaj(const Polinom * p, double x)` koja vraća vrednosti polinoma `p` u datoj tački `x` koristeći Hornerov algoritam.
- Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja vraća zbir dva polinoma `p` i `q`.
- Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja vraća proizvod dva polinoma `p` i `q`.
- Napisati funkciju `Polinom izvod(const Polinom * p)` koja vraća izvod polinoma `p`.
- Napisati funkciju `Polinom n_izvod(const Polinom * p, int n)` koja vraća  $n$ -ti izvod polinoma `p`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati polinome `p` i `q`, a zatim ih ispisati na standardni izlaz u odgovarajućem formatu. Izračunati i ispisati zbir `z` i proizvod `r` unetih polinoma `p` i `q`. Sa standardnog ulaza učitati realni broj `x`, a zatim na standardni izlaz ispisati vrednost polinoma `z` u tački `x` zaokruženu na dve decimale. Na kraju, sa standardnog ulaza učitati broj `n` i na izlaz ispisati  $n$ -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 polinom z:
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 z:
0
Vrednost polinoma z u tacki 0.00 je 0.30
Unesite izvod polinoma koji zelite:
3
3. izvod polinoma r je: 151.20x^2+176.40x-1.62
```

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

- (a) Definisati strukturu `Razlomak` koja opisuje razlomak.
- (b) Napisati funkciju `Razlomak ucitaj()` za učitavanje razlomka.
- (c) Napisati funkciju `void ispisi(const Razlomak * r)` koja ispisuje razlomak `r`.
- (d) Napisati funkciju `int brojilac(const Razlomak * r)` koja vraćaja brojilac razlomka `r`.
- (e) Napisati funkciju `int imenilac(const Razlomak * r)` koja vraćaja imenilac razlomka `r`.
- (f) Napisati funkciju `double realna_vrednost(const Razlomak * r)` koja vraća odgovarajuću realnu vrednost razlomka `r`.
- (g) Napisati funkciju `double recipročna_vrednost(const Razlomak * r)` koja vraća recipročnu vrednost razlomka `r`.
- (h) Napisati funkciju `Razlomak skрати(const Razlomak * r)` koja vraća skraćenu vrednost datog razlomka `r`.
- (i) Napisati funkciju `Razlomak saberi(const Razlomak * r1, const Razlomak * r2)` koja vraća zbir dva razlomka `r1` i `r2`.
- (j) Napisati funkciju `Razlomak oduzmi(const Razlomak * r1, const Razlomak * r2)` koja vraća razliku dva razlomka `r1` i `r2`.
- (k) Napisati funkciju `Razlomak pomnozi(const Razlomak * r1, const Razlomak * r2)` koja vraća proizvod dva razlomka `r1` i `r2`.
- (l) Napisati funkciju `Razlomak podeli(const Razlomak * r1, const Razlomak * r2)` koja vraća količnik dva razlomka `r1` i `r2`.

Napisati program koji testira prethodne funkcije. Sa standardnog ulaza učitati dva razlomka `r1` i `r2`. Na standardni izlaz ispisati skraćene vrednosti zbira, razlike, proizvoda i količnika razlomaka `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: 2 3
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. Biblioteka treba da sadrži funkcije `stampanje_bitova`, `stampanje_bitova_short` i `stampanje_bitova_char` za štampanje bitova u binarnom zapisu celog broja tipa `int`, `short` i `char`, koji se zadaje kao argument funkcije. Napisati program koji testira napisanu biblioteku. Sa standardnog ulaza učitati u heksadekadnom formatu cele brojeve tipa `int`, `short` i `char` i na standardni izlaz ispisati njihovu binarnu reprezentaciju.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj tipa int: 0x4f4f4f4f
Binarna reprezentacija: 01001111010011110100111101001111
Unesite broj tipa short: 0x4f4f
Binarna reprezentacija: 0100111101001111
Unesite broj tipa char: 0x4f
Binarna reprezentacija: 01001111
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkcije `_bitove_1` i `prebroj_bitove_2` koje vraćaju broj jedinica u binarnom zapisu označenog celog broja  $x$  koji se zadaje kao argument funkcije. Prebrojavanje bitova ostvariti na dva načina:

- formiranjem odgovarajuće maske i njenim pomeranjem (funkcija `prebroj_bitove_1`)
- formiranjem odgovarajuće maske i pomeranjem promenljive  $x$  (funkcija `prebroj_bitove_2`).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati ceo broj u heksadekadnom formatu i redni broj funkcije koju treba primeniti (1 ili 2), a zatim na standardni izlaz ispisati broj jedinica u binarnom zapisu učitano broja pozivom izabrane funkcije. Ukoliko korisnik ne unese ispravnu vrednost 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
Poziva se funkcija prebroj_bitove_1
Broj jedinica u zapisu je 7
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj: -0x7F
Unesite redni broj funkcije: 2
Poziva se funkcija prebroj_bitove_2
Broj jedinica u zapisu je 26
```

## 1 Uvodni zadaci

---

### Primer 3

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

### Primer 4

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj: 0x00FF00FF
Unesite redni broj funkcije: 3
IZLAZ ZA GREŠKE:
Greska: Neodgovarajući redni broj funkcije.
```

[Rešenje 1.6]

**Zadatak 1.7** Napisati funkcije `unsigned najveci(unsigned x)` i `unsigned najmanji(unsigned x)` koje vraćaju najveći, odnosno najmanji neoznačen ceo broj koji se može zapisati istim binarnim ciframa kao broj `x`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati neoznačen ceo broj u heksadekadnom formatu, a zatim ispisati binarnu reprezentaciju najvećeg i najmanjeg broja koji se može zapisati istim binarnim ciframa kao učitani broj.

### Test 1

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

### Test 2

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

### Test 3

```
ULAZ:
0x00FF00FF
IZLAZ:
Najveci:
11111111111111111000000000000000
Najmanji:
00000000000000000111111111111111
```

### Test 4

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

[Rešenje 1.7]

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

- Napisati funkciju `unsigned postavi_0(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se `n` bitova datog broja `x`, počevši od pozicije `p`, postave na 0.
- Napisati funkciju `unsigned postavi_1(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se `n` bitova datog broja `x`, počevši od pozicije `p`, postave na 1.





### Primer 2

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

x = 2882398951                = 10101011110011011110101011100111
postavi_1(2882398951, 5, 10) = 10101011110011011110111111100111

x = 2882398951                = 10101011110011011110101011100111
vrati_bitove(2882398951, 5, 10) = 00000000000000000000000000001011

x = 2882398951                = 10101011110011011110101011100111
y = 35156526                  = 0000010000110000111001000101110
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 bitova ulevo podrazumeva se pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najveće težine pomera na poziciju najmanje težine. Analogno, rotiranje bitova udesno podrazumeva pomeranje svih bitova za jednu poziciju udesno, s tim što se bit sa pozicije najmanje težine pomera na poziciju najveće težine.

- Napisati funkciju `unsigned rotiraj_ulevo(unsigned x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta ulevo datog celog neoznačenog broja `x`.
- Napisati funkciju `unsigned rotiraj_udesno(unsigned x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta udesno datog celog neoznačenog broja `x`.
- Napisati funkciju `int rotiraj_udesno_oznaceni(int x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta udesno datog celog broja `x`.

Napisati program koji sa standardnog ulaza učitava neoznačene cele brojeve `x` i `n` koji se unose u heksadekaskom formatu, zatim ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem tri prethodno napisane funkcije sa argumentima `x` i `n`, a na kraju ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem funkcije `rotiraj_udesno_oznaceni` za argumente `-x` i `n`.

### Primer 1

```

INTERAKCIJA SA PROGRAMOM:
Unesite neoznaceni ceo broj x: ba11a7
Unesite neoznaceni ceo broj n: 5
x = 00000000101110100001000110100111
rotiraj_ulevo(ba11a7, 5) = 00010111010000100011010011100000
rotiraj_udesno(ba11a7, 5) = 00111000000001011101000010001101
rotiraj_udesno_oznaceni(ba11a7, 5) = 00111000000001011101000010001101
rotiraj_udesno_oznaceni(-ba11a7, 5) = 11000111111110100010111101110010
    
```

[Rešenje 1.9]

**Zadatak 1.10** Napisati funkciju `unsigned ogledalo(unsigned x)` koja vraća ceo broj čiji binarni zapis predstavlja sliku u ogledalu binarnog zapisa broja `x`. Napisati program koji testira datu funkciju za broj koji se sa standardnog ulaza zadaje u heksadekadnom formatu. Najpre ispisati binarnu reprezentaciju unetog broja, a zatim i binarnu reprezentaciju broja dobijenog kao njegova slika u ogledalu.

#### Test 1

```

ULAZ:
255
IZLAZ:
0000000000000000000000001001010101
10101010010000000000000000000000
    
```

#### Test 2

```

ULAZ:
-15
IZLAZ:
1111111111111111111111111111101011
11010111111111111111111111111111
    
```

[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 `int broj_parova(unsigned int x)` koja vraća broj pojava dve uzastopne jedinice u binarnom zapisu celog neoznačenog

## 1 Uvodni zadaci

---

broja  $x$ . Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Tri uzastopne jedinice sadrže dve uzastopne jedinice dva puta.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ:    11    IZLAZ:    1	ULAZ:    1024    IZLAZ:    0	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  $j$  učitati kao parametre komandne linije. Pri rešavanju nije dozvoljeno koristiti ni pomoćni niz ni aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ .

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 2</i>
POKRETANJE: ./a.out 1 2    INTERAKCIJA SA PROGRAMOM:    ULAZ:    11    IZLAZ:    13	POKRETANJE: ./a.out 1 2    INTERAKCIJA SA PROGRAMOM:    ULAZ:    1024    IZLAZ:    1024	POKRETANJE: ./a.out 12 12    INTERAKCIJA SA PROGRAMOM:    ULAZ:    12345    IZLAZ:    12345

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ:    11    IZLAZ:    0000000B	ULAZ:    1024    IZLAZ:    00000400	ULAZ:    12345    IZLAZ:    00003039

[Rešenje 1.14]

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

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

**Zadatak 1.16** Napisati funkciju koja vraća broj petica u oktalnom zapisu neoznačenog celog broja  $x$ . Napisati program koji testira tu funkciju 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>
<pre> ULAZ:  123 IZLAZ: 0 </pre>	<pre> ULAZ:  3245 IZLAZ: 2 </pre>	<pre> ULAZ:  100328 IZLAZ: 1 </pre>

## 1.3 Rekurzija

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

- tako da rešenje bude linearne složenosti,
- tako da rešenje bude logaritamske složenosti.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1' ili '2'), ceo broj  $x$  i prirodan broj  $k$ , a zatim na standardni izlaz ispisati rezultat primene izabrane funkcije na unete brojeve. Ukoliko se na ulazu unese pogrešan redni broj funkcije, ispisati odgovarajuću poruku o grešci na standardni izlaz i prekinuti izvršavanje programa.

## 1 Uvodni zadaci

---

### Primer 1

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

### Primer 2

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

[Rešenje 1.17]

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

- funkciju `unsigned paran(unsigned n)` koja proverava da li je broj cifara broja `x` paran i vraća 1 ako jeste, a 0 inače;
- i funkciju `unsigned neparan(unsigned n)` koja proverava da li je broj cifara broja `x` neparan i vraća 1 ako jeste, 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: Gornja vrednost za  $n$  je postavljena na 12 zbog ograničenja veličine broja koji može da stane u promenljivu tipa `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 vraća  $n$ -ti element u nizu Fibonačijevih brojeva. Elementi niza Fibonačijevih brojeva  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \end{aligned}$$

Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati prirodan broj  $n$  i na standardni izlaz ispisati rezultat primene napisane funkcije na prirodan broj  $n$ .

*Primer 1*

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

*Primer 2*

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

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

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= a \cdot F(n-1) + b \cdot F(n-2) \end{aligned}$$

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1', '2', '3'), vrednosti koeficijenata  $a$  i  $b$  i prirodan broj  $n$ . Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim podacima, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa. NAPOMENA: Niz  $F$  definisan na ovaj način predstavlja uopštenje Fibonačijevih brojeva.

## 1 Uvodni zadaci

---

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije:
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:
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

- sabirajući elemente počev od početka niza ka kraju niza,
- 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. Elementi niza se unose sve do kraja ulaza (EOF). Pretpostaviti da niz neće imati više od 256 elemenata.

*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 vektora celih brojeva. Napisati program koji testira ovu funkciju za nizove (vektore) koji se unose sa standardnog ulaza. Prvo treba uneti dimenziju nizova, a zatim i njihove elemente. Na standardni izlaz ispisati skalarni proizvod unetih nizova. Pretpostaviti da 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 vraća broj pojavljivanja elementa  $x$  u nizu  $a$  dužine  $n$ . Napisati program koji testira ovu funkciju za broj



## 1 Uvodni zadaci

---

$x$  i niz  $a$  koji se unose sa standardnog ulaza. Prvo se unosi  $x$ , a zatim elementi niza sve do kraja ulaza. Pretpostaviti da nizovi 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 data cela broja uzastopni članovi datog celobrojnog niza. Sa standardnog ulaza učitati tri broja, a zatim elemente niza sve do kraja ulaza. Na standardni izlaz ispisati rezultat primene funkcije nad učitanim podacima. 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 jesu uzastopni  
clanovi niza.
```

[Rešenje 1.27]

**Zadatak 1.28** Napisati rekurzivnu funkciju `int prebroj(int x)` koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja  $x$ . Napisati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza u heksadekadnom formatu.



## 1 Uvodni zadaci

[Rešenje 1.31]

**Zadatak 1.32** Napisati rekurzivnu funkciju `int palindrom(char s[], int n)` koja ispituje da li je data niska `s` palindrom. Napisati program koji testira ovu funkciju za nisku koja se zadaje sa standardnog ulaza. Pretpostaviti da niska neće imati više od 31 karaktera.

```
Test 1          Test 2          Test 3
|| ULAZ:        || ULAZ:        || ULAZ:
|| a            || aa           || aba
|| IZLAZ:       || IZLAZ:       || IZLAZ:
|| da           || da           || da

Test 4          Test 5
|| ULAZ:        || ULAZ:
|| programiranje || anavolimilovana
|| IZLAZ:       || IZLAZ:
|| ne           || 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          Test 2          Test 3
|| ULAZ:        || ULAZ:        || ULAZ:
|| 2            || 3            || -5
|| IZLAZ:       || IZLAZ:       || IZLAZ ZA GREŠKE:
|| 1 2          || 1 2 3        || Greska: Duzina
|| 2 1          || 1 3 2        || permutacije
||              || 2 1 3        || mora biti
||              || 2 3 1        || broj iz
||              || 3 1 2        || intervala
||              || 3 2 1        || [0, 15]!
```

[Rešenje 1.33]

\* **Zadatak 1.34** Paskalov trougao sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbira 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 elementa u tom redu (na toj hipotenuzi) odgovara binomnom koeficijentu  $\binom{n}{k}$ , pri čemu brojanje počinje od nule. Na primer, vrednost polja  $(4, 2)$  je 6.

- (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:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
8

```

```

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

```

[Rešenje 1.34]

\* **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:
a a
a b
b a
b b

```

```

Test 2
ULAZ:
3
IZLAZ:
a a a
a a b
a b a
a b b
b a a
b a b
b b a
b b b

```

\* **Zadatak 1.36 Hanojske kule:** Data su tri vertikalna štapa. Na jednom od njih se nalazi  $n$  diskova poluprečnika 1, 2, 3, ... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je premestiti diskove sa jednog na drugi štap tako da budu u istom redosledu, pri čemu se ni

u jednom trenutku ne sme staviti veći disk preko manjeg. Preostali štap koristiti 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. Preostala dva štapa koristiti kao pomoćne štapove 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
14    argument funkcije */
15 void ucitaj_kompleksan_broj(KompleksanBroj * z)
16 {
17     /* Ucitava se vrednost sa standardnog ulaza */
18     printf("Unesite realni i imaginarni 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 */
```

```
printf("(");
30
/* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
32 razlicit od nule: tada se realni deo ispisuje na standardni
izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li
34 je imaginarni deo pozitivan ili negativan, a potom i apsolutna
vrednost imaginarnog dela kompleksnog broja 2) realni deo
36 kompleksnog broja je nula: tada se samo ispisuje imaginaran
deo, s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
38 decimalnih mesta */
if (z.real != 0) {
40     printf("%.2f", z.real);

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

52 /* Završava se sa ispisom */
54 printf(")");
}

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

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

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

74 /* Funkcija vraća vrednost konjugovano kompleksnog broja koji
76 odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z)
78 {
80     /* Konjugovano kompleksan broj z se dobija tako sto se promeni
znak imaginarnom delu kompleksnog broja */
```

## 1 Uvodni zadaci

---

```
82     KomplexsanBroj z1 = z;
      z1.imag *= -1;
84     return z1;
      }
86
      /* Funkcija vraća kompleksan broj čija vrednost je jednaka zbiru
88     argumenata funkcije */
      KomplexsanBroj saberi(KomplexsanBroj z1, KomplexsanBroj z2)
90     {
92         /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
          broj čiji je realan deo zbir realnih delova kompleksnih
          brojeva z1 i z2, a imaginaran deo zbir imaginarnih delova
94         kompleksnih brojeva z1 i z2 */
          KomplexsanBroj z = z1;
96         z.real += z2.real;
          z.imag += z2.imag;
98
          return z;
100     }
102
      /* Funkcija vraća kompleksan broj čija vrednost je jednaka razlici
104     argumenata funkcije */
      KomplexsanBroj oduzmi(KomplexsanBroj z1, KomplexsanBroj z2)
106     {
108         /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
          broj čiji je realan deo razlika realnih delova kompleksnih
          brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
110         kompleksnih brojeva z1 i z2 */
          KomplexsanBroj z = z1;
          z.real -= z2.real;
112         z.imag -= z2.imag;
114
          return z;
116     }
118
      /* Funkcija vraća kompleksan broj čija vrednost je jednaka
      proizvodu argumenata funkcije */
      KomplexsanBroj mnozi(KomplexsanBroj z1, KomplexsanBroj z2)
120     {
122         /* Rezultat množenja dva kompleksna broja z1 i z2 je kompleksan
          broj čiji se realan i imaginaran deo računaju po formuli za
          množenje kompleksnih brojeva z1 i z2 */
124         KomplexsanBroj z;
          z.real = z1.real * z2.real - z1.imag * z2.imag;
126         z.imag = z1.real * z2.imag + z1.imag * z2.real;
128
          return z;
130     }
132
      /* Funkcija vraća argument zadatog kompleksnog broja */
      float argument(KomplexsanBroj z)
```

```
134 {
    /* Argument kompleksnog broja z se racuna pozivanjem funkcije
       atan2 iz biblioteke math.h */
136 return atan2(z.imag, z.real);
138 }

138 int main()
140 {
    char c;

142     /* Deklaracija 3 promenljive tipa KompleksanBroj */
144     KompleksanBroj z1, z2, z;

146     /* Ucitava se prvi kompleksni broj, koji se potom ispisuje na
       standardni izlaz */
148     ucitaj_kompleksan_broj(&z1);
       ispisi_kompleksan_broj(z1);
150     printf("\n");

152     /* Ucitava se drugi kompleksni broj, koji se potom ispisuje na
       standardni izlaz */
154     ucitaj_kompleksan_broj(&z2);
       ispisi_kompleksan_broj(z2);
156     printf("\n");

158     /* Ucitavase znak na osnovu koga korisnik bira aritmeticku
       operaciju koja ce se izvršiti nad kompleksnim brojevima, a
160     zatim se vrsi provera da li je unet neki od dozvoljenih
       aritmetickih znakova. */
162     getchar();
       printf("Unesite znak (+,-,*): ");
164     scanf("%c", &c);
       if (c != '+' && c != '-' && c != '*') {
166         fprintf(stderr, "Greska: Nedoovoljena vrednost operatora.\n");
           exit(EXIT_FAILURE);
168     }

170     /* Analizira se uneti operator */
       if (c == '+') {
172         /* Racuna se zbir */
           z = saberi(z1, z2);
174     } else if (c == '-') {
           /* Racuna se razlika */
176         z = oduzmi(z1, z2);
       } else {
178         /* Racuna se proizvod */
           z = mnozi(z1, z2);
180     }

182     /* Ispisuje se rezultat */
       ispisi_kompleksan_broj(z1);
184     printf(" %c ", c);
```



## 1 Uvodni zadaci

```
186     ispisi_kompleksan_broj(z2);
187     printf(" = ");
188     ispisi_kompleksan_broj(z);
189
190     /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
191        broja */
192     printf("\nRealni_deo: %.f\nImaginarni_deo: %.f\nModuo: %.f\n",
193           realan_deo(z), imaginaran_deo(z), moduo(z));
194
195     /* Izracunava se i ispisuje konjugovano kompleksan broj drugog
196        kompleksnog broja */
197     printf("Konjugovano kompleksan broj: ");
198     ispisi_kompleksan_broj(konjugovan(z));
199     printf("\n");
200
201     /* Testira se funkcija koja racuna argument kompleksnog broja */
202     printf("Argument kompleksnog broja: %.f\n", argument(z));
203
204     exit(EXIT_SUCCESS);
205 }
```

### Rešenje 1.2

*kompleksan\_broj.h*

```
1  /* Zaglavlje kompleksan_broj.h sadrzi definiciju tipa
2     KompleksanBroj i deklaracije funkcija za rad sa kompleksnim
3     brojevima. Zaglavlje nikada ne treba da sadrzi definicije
4     funkcija. Da bi neki program mogao da koristi ove brojeve i
5     funkcije iz ove biblioteke, neophodno je da ukljuci ovo
6     zaglavlje. */
7
8     /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i
9        onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
10       posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi
11       u narednoj pretprocesorskoj define direktivi. */
12     #ifndef _KOMPLEKSAN_BROJ_H
13     #define _KOMPLEKSAN_BROJ_H
14
15     /* Struktura KompleksanBroj */
16     typedef struct {
17         float real;
18         float imag;
19     } KompleksanBroj;
20
21     /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
22        definisane u kompleksan_broj.c */
23
24     /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
25        kompleksnog broja i smesta ih u strukturu cija je adresa
```

```

    argument funkcije */
27 void ucitaj_kompleksan_broj(KompleksanBroj * z);
29 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
    obliku (x + i y) */
31 void ispisi_kompleksan_broj(KompleksanBroj z);
33 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
    float realan_deo(KompleksanBroj z);
35
37 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
    float imaginaran_deo(KompleksanBroj z);
39 /* Funkcija vraca vrednost modula zadanog kompleksnog broja */
    float moduo(KompleksanBroj z);
41
43 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
    odgovara kompleksnom broju argumentu */
    KompleksanBroj konjugovan(KompleksanBroj z);
45
47 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
    argumenata funkcije */
    KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
49
51 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
    argumenata funkcije */
    KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
53
55 /* Funkcija vraca kompleksan broj cija vrednost je jednaka
    proizvodu argumenata funkcije */
    KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
57
59 /* Funkcija vraca argument zadanog kompleksnog broja */
    float argument(KompleksanBroj z);
61 /* Kraj zakljucanog dela */
    #endif

```

*kompleksan\_broj.c*

```

#include <stdio.h>
2 #include <math.h>
4 /* Ukljucuje se zaglavlje za rad sa kompleksnim brojevima, jer je
    neophodno da bude poznata definicija tipa KompleksanBroj. */
6 #include "kompleksan_broj.h"
8 void ucitaj_kompleksan_broj(KompleksanBroj * z)
    {
10     /* Ucitavanje vrednosti sa standardnog ulaza */
        printf("Unesite realan i imaginaran deo kompleksnog broja: ");

```

## 1 Uvodni zadaci

---

```
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 slucaja: 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
24  je 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 imaginaran
27  deo, 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  /* Zavrшава se sa ispisom */
44  printf("");
45  }

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

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

56  float moduo(KompleksanBroj z)
57  {
58  /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
59  return sqrt(z.real * z.real + z.imag * z.imag);
60  }
61  }
```

```
64 KompleksanBroj konjugovan(KompleksanBroj z)
65 {
66     /* Konjugovano kompleksan broj se dobija od datog broja z tako
67     sto se promeni znak imaginarnom delu kompleksnog broja */
68     KompleksanBroj z1 = z;
69     z1.imag *= -1;
70     return z1;
71 }
72
73
74 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
75 {
76     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
77     broj ciji je realan deo zbir realnih delova kompleksnih
78     brojeva z1 i z2, a imaginaran deo zbir imaginarnih delova
79     kompleksnih brojeva z1 i z2 */
80     KompleksanBroj z = z1;
81     z.real += z2.real;
82     z.imag += z2.imag;
83
84     return z;
85 }
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
98     return z;
99 }
100
101
102 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
103 {
104     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
105     broj ciji se realan i imaginaran deo racunaju po formuli za
106     mnozenje kompleksnih brojeva z1 i z2 */
107     KompleksanBroj z;
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
115 float argument(KompleksanBroj z)
116 {
117     /* Argument kompleksnog broja z se racuna pozivanjem funkcije
118     atan2 iz biblioteke math.h */
```

## 1 Uvodni zadaci

---

```
116     return atan2(z.imag, z.real);
    }
```

*main.c*

```
1  /*****
2  Ovaj program koristi korektno definisanu biblioteku kompleksnih
3  brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
4  kompleksnog broja i popis deklaracija podrzanih funkcija, a u
5  kompleksan_broj.c se nalaze njihove definicije.
6
7  Kompilacija programa se najjednostavnije postize naredbom
8  gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c
9
10 Kompilacija se moze uraditi i postupno, na sledeci nacin:
11 gcc -Wall -c -o kompleksan_broj.o kompleksan_broj.c
12 gcc -Wall -c -o main.o main.c
13 gcc -lm -o kompleksan_broj kompleksan_broj.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 kompleksan_broj.c kompleksan_broj.h) nalazi u direktorijumu sa imenom
18 header_dir prevodjenje se vrši dodavanjem opcije -I header_dir
19 gcc -I header_dir -Wall -lm -o kompleksan_broj kompleksan_broj.c
20     main.c
21 *****/
22
23 #include <stdio.h>
24 /* Ukljucuje se zaglavlje za rad sa kompleksnim brojevima */
25 #include "kompleksan_broj.h"
26
27 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
28     polarni oblik */
29 int main()
30 {
31     KompleksanBroj z;
32
33     /* Ucitava se kompleksan broj */
34     ucitaj_kompleksan_broj(&z);
35
36     /* Ispisuje se polarni oblik kompleksnog broja */
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

*polinom.h*

```
1 #ifndef _POLINOM_H
2 #define _POLINOM_H
3
4 /* Maksimalni stepen polinoma */
5 #define MAKS_STEPEN 20
6
7 /* Polinomi se predstavljaju strukturom koja cuva koeficijente
8    (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
9 typedef struct {
10     double koef[MAKS_STEPEN + 1];
11     int stepen;
12 } Polinom;
13
14 /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
15    obliku. Polinom se prenosi po adresi da bi se uštedela memorija:
16    ne kopira se cela struktura, vec se samo prenosi adresa na kojoj
17    se nalazi polinom koji ispisujemo */
18 void ispis(const Polinom * p);
19
20 /* Funkcija koja ucitava polinom sa standardnog ulaza */
21 Polinom ucitaj();
22
23 /* Funkcija racuna i vraća vrednost polinoma p u tacki x Hornerovim
24    algoritmom */
25 double izracunaj(const Polinom * p, double x);
26
27 /* Funkcija koja sabira dva polinoma */
28 Polinom saberi(const Polinom * p, const Polinom * q);
29
30 /* Funkcija koja mnozi dva polinoma p i q */
31 Polinom pomnozi(const Polinom * p, const Polinom * q);
32
33 /* Funkcija koja racuna izvod polinoma p */
34 Polinom izvod(const Polinom * p);
35
36 /* Funkcija koja racuna n-ti izvod polinoma p */
37 Polinom n_izvod(const Polinom * p, int n);
38 #endif
```

*polinom.c*

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

## 1 Uvodni zadaci

---

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

38 Polinom ucitaj()
  {
40   int i;
   Polinom p;
42
   printf("Unesite polinom p (prvo stepen, pa zatim koeficijente");
44   printf(" od najveceg stepena do nultog):\n");

46   /* Ucitava se stepena polinoma */
   scanf("%d", &p.stepen);
48

   /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
   dozvoljenog opsega */
50   while (p.stepen > MAKS_STEPEN || p.stepen < 0) {
52     printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
     scanf("%d", &p.stepen);
54   }

56   /* Unose se koeficijenti polinoma */
   for (i = p.stepen; i >= 0; i--)
```

```

58     scanf("%lf", &p.koef[i]);
60     /* Vraca se procitani polinom */
61     return p;
62 }
64 double izracunaj(const Polinom * p, double x)
65 {
66     /* Primer: Hornerov algoritam za polinom x^4+2x^3+3x^2+2x+1:
67     x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 2)*x + 1 */
68
69     double rezultat = 0;
70     int i = p->stepen;
71
72     /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
73     svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
74     vrednost odgovarajuceg koeficijenta */
75     for (; i >= 0; i--)
76         rezultat = rezultat * x + p->koef[i];
77     return rezultat;
78 }
80 Polinom saberi(const Polinom * p, const Polinom * q)
81 {
82     Polinom r;
83     int i;
84
85     /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
86     stepenom */
87     r.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
88
89     /* Racunaju se svi koeficijenti rezultujuceg polinoma tako sto se
90     sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
91     sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
92     od stepena nekog od polaznih polinoma podrazumeva se da je
93     koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
94     polinoma */
95     for (i = 0; i <= r.stepen; i++)
96         r.koef[i] =
97             (i > p->stepen ? 0 : p->koef[i]) +
98             (i > q->stepen ? 0 : q->koef[i]);
99
100     /* Vraca se izracunati polinom */
101     return r;
102 }
104 Polinom pomnozi(const Polinom * p, const Polinom * q)
105 {
106     int i, j;
107     Polinom r;
108
109     /* Stepen rezultata odgovara zbiru stepena polaznih polinoma */

```



## 1 Uvodni zadaci

---

```
110 r.stepen = p->stepen + q->stepen;
111 if (r.stepen > MAKS_STEPEN) {
112     fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
113     exit(EXIT_FAILURE);
114 }

116 /* Svi koeficijenti rezultujućeg polinoma se inicijalizuju na 0 */
117 for (i = 0; i <= r.stepen; i++)
118     r.koef[i] = 0;

120 /* U svakoj iteraciji odgovarajući koeficijent rezultata se
121    uvecava za proizvod odgovarajućih koeficijenata iz polaznih
122    polinoma */
123 for (i = 0; i <= p->stepen; i++)
124     for (j = 0; j <= q->stepen; j++)
125         r.koef[i + j] += p->koef[i] * q->koef[j];

126 /* Vraca se izracunati polinom */
127 return r;
128 }

130 Polinom izvod(const Polinom * p)
131 {
132     int i;
133     Polinom r;

134     /* Izvod polinoma ce imati stepen za jedan stepen manji od
135        stepena polaznog polinoma. Ukoliko je stepen polinoma p vec
136        nula, onda je rezultujući polinom nula (izvod od konstante je
137        nula). */
138     if (p->stepen > 0) {
139         r.stepen = p->stepen - 1;

140         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
141            polaznog polinoma */
142         for (i = 0; i <= r.stepen; i++)
143             r.koef[i] = (i + 1) * p->koef[i + 1];
144     } else
145         r.koef[0] = r.stepen = 0;

146     /* Vraca se izracunati polinom */
147     return r;
148 }

150 Polinom n_izvod(const Polinom * p, int n)
151 {
152     int i;
153     Polinom r;

154     /* Provera da li je n nenegativna vrednost */
155     if (n < 0) {
156         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
157     }
158 }
```

```

162     exit(EXIT_FAILURE);
    }
164
    /* Nulti izvod je bas taj polinom */
166     if (n == 0)
        return *p;
168
    /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove
170     funkcija za racunanje prvog izvoda polinoma */
    r = izvod(p);
172     for (i = 1; i < n; i++)
        r = izvod(&r);
174
    /* Vraca se dobijeni polinom */
176     return r;
}

```

*main.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   #include "polinom.h"
5
   int main(int argc, char **argv)
7   {
   Polinom p, q, z, r;
9   double x;
   int n;
11
   /* Unosi se polinom p */
13   p = ucitaj();
15
   /* Ispisuje se polinom p */
   ispisi(&p);
17
   /* Unosi se polinom q */
19   q = ucitaj();
21
   /* Polinomi se sabiraju i ispisuje se izracunati zbir */
   z = saberi(&p, &q);
23   printf("Zbir polinoma je polinom z:\n");
   ispisi(&z);
25
   /* Polinomi se mnoze i ispisuje se izracunati proizvod */
27   r = pomnozi(&p, &q);
   printf("Prozvod polinoma je polinom r:\n");
29   ispisi(&r);
31
   /* Ispisuje se vrednost polinoma u unetoj tacki */
   printf("Unesite tacku u kojoj racunate vrednost polinoma z:\n");

```

## 1 Uvodni zadaci

---

```
33 scanf("%lf", &x);
34 printf("Vrednost polinoma z u tacki %.2f je %.2f\n", x,
35       izracunaj(&z, x));
36
37 /* Racuna se n-ti izvod polinoma i ispisuje se rezultat */
38 printf("Unesite izvod polinoma koji zelite:\n");
39 scanf("%d", &n);
40 r = n_izvod(&r, n);
41 printf("%d. izvod polinoma r je: ", n);
42 ispisi(&r);
43
44 exit(EXIT_SUCCESS);
45 }
```

### Rešenje 1.5

*stampanje\_bitova.h*

```
1 #ifndef _STAMPANJE_BITOVA_H
2 #define _STAMPANJE_BITOVA_H
3
4 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
5 celog broja u memoriji. Bitove koji predstavljaju binarnu
6 reprezentaciju broja treba ispisati sa leva na desno, tj. od
7 bita najveće težine ka bitu najmanje težine */
8 void stampaj_bitove(unsigned x);
9
10 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
11 celog broja tipa 'short' u memoriji. */
12 void stampaj_bitove_short(short x);
13
14 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
15 karaktera u memoriji. */
16 void stampaj_bitove_char(char x);
17
18 #endif
```

*stampanje\_bitova.c*

```
1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 void stampaj_bitove(unsigned x)
5 {
6     /* Broj bitova celog broja */
7     unsigned velicina = sizeof(unsigned) * 8;
8
9     /* Maska koja se koristi za ocitavanje bitova celog broja */
```

```
10 unsigned maska;
12 /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
14 na mestu bita najvece tezine sadrzi jedinicu, a na svim ostalim
16 mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
18 se jedini bit jedinica pomera udesno, kako bi se odredio
20 naredni bit broja x koji je argument funkcije. Zatim se
22 odgovarajuca cifra, ('0' ili '1'), ispisuje na standardni
24 izlaz. Neophodno je da promenljiva maska bude deklarirana kao
26 neoznacena ceo broj kako bi se pomeranjem u desno vrsilo logicko
28 pomeranje (popunjavanje nulama), a ne aritmeticko pomeranje
30 (popunjavanje znakom broja). */
32 for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
34 putchar(x & maska ? '1' : '0');
36
38 putchar('\n');
40 }
42 void stampaj_bitove_short(short x)
44 {
46 /* Broj bitova celog broja tipa short */
48 unsigned velicina = sizeof(short) * 8;
50
52 /* Maska koja se koristi za ocitavanje bitova broja tipa short */
54 unsigned short maska;
56
58 for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
60 putchar(x & maska ? '1' : '0');
62
64 putchar('\n');
66 }
68 void stampaj_bitove_char(char x)
70 {
72 /* Broj bitova karaktera */
74 unsigned velicina = sizeof(char) * 8;
76
78 /* Maska koja se koristi za ocitavanje bitova jednog karaktera */
80 unsigned char maska;
82
84 for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
86 putchar(x & maska ? '1' : '0');
88
90 putchar('\n');
92 }
```

*main.c*

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

## 1 Uvodni zadaci

---

```
4 int main()
5 {
6     int broj_int;
7     short broj_short;
8     char broj_char;
9
10    /* Ucitava se broj tipa int */
11    printf("Unesite broj tipa int: ");
12    scanf("%x", &broj_int);
13
14    /* Ispisuje se binarna reprezentacija unetog broja */
15    printf("Binarna reprezentacija: ");
16    stampaj_bitove(broj_int);
17
18    /* Ucitava se broj tipa short */
19    printf("Unesite broj tipa short: ");
20    scanf("%hx", &broj_short);
21
22    /* Ispisuje se binarna reprezentacija unetog broja */
23    printf("Binarna reprezentacija: ");
24    stampaj_bitove_short(broj_short);
25
26    /* Ucitava se broj tipa char */
27    printf("Unesite broj tipa char: ");
28    scanf("%hhx", &broj_char);
29
30    /* Ispisuje se binarna reprezentacija unetog broja */
31    printf("Binarna reprezentacija: ");
32    stampaj_bitove_char(broj_char);
33
34    return 0;
35 }
```

### Rešenje 1.6

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
5    kreiranjem odgovarajuće 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 čija binarna reprezentacija izgleda
12       100000...0000000, koja služi za očitavanje bita najveće
13       težine. U svakoj iteraciji maska se pomera u desno za 1 mesto,
14       i očitava se sledeći bit. Petlja se završava kada više 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 odgovarajuće 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 oznacen ceo broj x naredba x>>=1
31        vrši aritmeticko pomeranje u desno, tj. popunjavanje bita
32        najveće težine bitom znaka. U tom slučaju, za negativnu
33        vrednost promenljive x, nikad ne bi bio ispunjen uslov x!=0 i
34        program bi bio zarobljen u beskonacnoj petlji. Zbog toga se
35        koristi pomeranje broja x ulevo i maska koja očitava bit
36        najveće težine. */
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
52     /* Dozvoljava se korisniku da bira na koji nacin ce biti
53        izracunat broj jedinica u zapisu broja */
54     printf("Unesite redni broj funkcije:\n");
55     scanf("%d", &i);
56
57     /* Ispisuje se odgovarajuci rezultat na osnovu unesenog rednog
58        broja funkcije */
59     if (i == 1) {
60         printf("Poziva se funkcija prebroj_bitove_1\n");
61         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_1(x));
62     } else if (i == 2) {
63         printf("Poziva se funkcija prebroj_bitove_2\n");
64         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_2(x));
65     } else {
66         fprintf(stderr,
67             "Greska: Neodgovarajuci redni broj funkcije.\n");
68         exit(EXIT_FAILURE);
69     }
70 }
```

## 1 Uvodni zadaci

---

```
69     }
71     exit(EXIT_SUCCESS);
}
```

### 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 vraća najveći neoznačeni broj sastavljen od istih
5     bitova koji se nalaze u binarnoj reprezentaciji vrednosti
6     promenljive x */
7  unsigned najveći(unsigned x)
8  {
9     unsigned velicina = sizeof(unsigned) * 8;
10
11    /* Formira se maska 100000...0000000 */
12    unsigned maska = 1 << (velicina - 1);
13
14    /* Rezultat se inicijalizuje vrednošću 0 */
15    unsigned rezultat = 0;
16
17    /* Promenljiva x se pomera u levo sve dok postoje jedinice u
18       njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
19       različita od nule). */
20    for (; x != 0; x <<= 1) {
21        /* Za svaku jedinicu koja se korišćenjem maske detektuje na
22           poziciji najveće težine u binarnoj reprezentaciji promenljive
23           x, potiskuje se jedna nova jedinica sa leva u rezultat */
24        if (x & maska) {
25            rezultat >>= 1;
26            rezultat |= maska;
27        }
28    }
29
30    /* Vraća se izračunata vrednost */
31    return rezultat;
32 }
33
34 /* Funkcija vraća najmanji neoznačeni broj sastavljen od istih
35    bitova koji se nalaze u binarnoj reprezentaciji vrednosti
36    promenljive x */
37    unsigned najmanji(unsigned x)
38    {
39        /* Rezultat se inicijalizuje vrednošću 0 */
40        unsigned rezultat = 0;
41
42        /* Promenljiva x se pomera u desno sve dok postoje jedinice u
```

```

43     njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
        razlicita od nule). */
45     for (; x != 0; x >>= 1) {
        /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
47         detektuje na poziciji najmanje tezine u binarnoj
            reprezentaciji promenjive x, potiskuje se jedna nova
49         jedinicu sa desna u rezultat */
        if (x & 1) {
51             rezultat <<= 1;
            rezultat |= 1;
53         }
        }
55
        /* Vraca se izracunata vrednost */
57     return rezultat;
    }
59
    int main()
61 {
        int broj;
63
        /* Ucitava se broj sa ulaza */
65     scanf("%x", &broj);
67
        /* Ispisuju se, redom, najveci i najmanji broj formirani od
            bitova unetog broja */
69     printf("Najveci:\n");
        stampaj_bitove(najveci(broj));
71
        printf("Najmanji:\n");
73     stampaj_bitove(najmanji(broj));
75
        return 0;
    }

```

### Rešenje 1.8

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

```

1     #include <stdio.h>
2     #include "stampanje_bitova.h"
3
4     /* Funkcija postavlja na nulu n bitova pocev od pozicije p */
    unsigned postavi_0(unsigned x, unsigned n, unsigned p)
6     {
7         /******
8         Formira se maska cija binarna reprezentacija ima n bitova
            postavljenih na 0 pocev od pozicije p, dok su svi ostali
10        postavljeni na 1. Na primer, za n=5 i p=10 formira se maska oblika
            1111 1111 1111 1111 1111 1000 0011 1111
12        To se postize na sledeci nacin:

```



## 1 Uvodni zadaci

```

14 ~0                1111 1111 1111 1111 1111 1111 1111 1111
14 (~0 << n)        1111 1111 1111 1111 1111 1111 1110 0000
16 ~(~0 << n)       0000 0000 0000 0000 0000 0000 0001 1111
16 ~(~0 << n) << ( p-n+1) 0000 0000 0000 0000 0000 0111 1100 0000
16 ~(~0 << n) << (p-n+1) 1111 1111 1111 1111 1111 1000 0011 1111
18 *****/
18 unsigned maska = ~(~0 << n) << (p - n + 1));
20
20 return x & maska;
22 }
24 /* Funkcija postavlja na jedinicu n bitova pocev od pozicije p */
24 unsigned postavi_1(unsigned x, unsigned n, unsigned p)
26 {
26 /******
28 Formira se maska kod koje je samo n bitova pocev od pocev od
28 pozicije p jednako 1, a ostali su 0.
30 Na primer, za n=5 i p=10 formira se maska oblika
30 0000 0000 0000 0000 0000 0111 1100 0000
32 *****/
32 unsigned maska = ~(~0 << n) << (p - n + 1);
34
34 return x | maska;
36 }
38 /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
38 predstavlja n bitova pocev od pozicije p u binarnoj
40 reprezentaciji broja x */
40 unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
42 {
42 /******
44 Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
44 Na primer, za n=5
46 0000 0000 0000 0000 0000 0000 0001 1111
48 *****/
48 unsigned maska = ~(~0 << n);
50
50 /* Najpre se vrednost promenljive x pomera u desno tako da
52 trazeno polje bude uz desni kraj. Zatim se maskiraju ostali
52 bitovi, sem zeljenih n i funkcija vraca tako izracunatu
54 vrednost */
54 return maska & (x >> (p - n + 1));
56 }
58 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
58 postavljeni na vrednosti n bitova najmanje tezine binarne
60 reprezentacije broja y */
60 unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p,
62 unsigned y)
64 {
64 /* Kreira se maska kod kod koje su poslednjih n bitova 1, a
ostali su 0. */

```

```
        unsigned poslednjih_n_1 = ~(~0 << n);
66
        /* Kao i kod funkcije postavi_0, i ovde se kreira maska koja ima
68         n bitova postavljenih na 0 pocevsi od pozicije p, dok su
           ostali bitovi 1 */
70        unsigned srednjih_n_0 = ~(~(-0 << n) << (p - n + 1));
72
        /* U promenljivu x_postavi_0 se smesta vrednost dobijena kada se
           u binarnoj reprezentaciji vrednosti promenljive x postavi na 0
74         n bitova na pozicijama pocev od p */
        unsigned x_postavi_0 = x & srednjih_n_0;
76
        /* U promenljivu y_pomeri_srednje se smesta vrednost dobijena od
           binarne reprezentacije vrednosti promenljive y cijih je n
78         bitova najnije tezine pomera tako da stoje pocev od pozicije
           p. Ostali bitovi su nule. Sa (y & poslednjih_n_1) postave na 0
80         svi bitovi osim najnižih n */
82        unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);
84
        return x_postavi_0 ^ y_pomeri_srednje;
    }
86
    /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
88     njih n */
    unsigned invertuj(unsigned x, unsigned n, unsigned p)
    {
        /* Formira se maska sa n jedinica pocev od pozicije p */
92        unsigned maska = ~(~0 << n) << (p - n + 1);
94
        /* Operator ekskluzivno ili invertuje sve bitove gde je
           odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
96        return maska ^ x;
    }
98
    int main()
    {
100        unsigned x, p, n, y;
102
        /* Ucitavaju se vrednosti sa standardnog ulaza */
104        printf("Unesite neoznaceni broj x:\n");
        scanf("%u", &x);
106        printf("Unesite neoznaceni broj n:\n");
        scanf("%u", &n);
108        printf("Unesite neoznaceni broj p:\n");
        scanf("%u", &p);
110        printf("Unesite neoznaceni broj y:\n");
        scanf("%u", &y);
112
        /* Ispisuju se binarne reprezentacije broja x i broja koji se
114         izracunava funkcijom postavi_0 sa argumentima x, n i p */
        printf("x = %10u %36s = ", x, "");
116        stampaj_bitove(x);
```

## 1 Uvodni zadaci

```
118     printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
119     stampaj_bitove(postavi_0(x, n, p));
120     printf("\n");
121
122     /* Ispisuju se binarne reprezentacije broja x i broja koji se
123        izracunava funkcijom postavi_1 sa argumentima x, n i p */
124     printf("x = %10u %36s = ", x, "");
125     stampaj_bitove(x);
126     printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
127     stampaj_bitove(postavi_1(x, n, p));
128     printf("\n");
129
130     /* Ispisuju se binarne reprezentacije broja x i broja koji se
131        izracunava funkcijom vrati_bitove sa argumentima x, n i p */
132     printf("x = %10u %36s = ", x, "");
133     stampaj_bitove(x);
134     printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
135     stampaj_bitove(vrati_bitove(x, n, p));
136     printf("\n");
137
138     /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji
139        se izracunava funkcijom postavi_1_n_bitova sa argumentima x,
140        n, p */
141     printf("x = %10u %36s = ", x, "");
142     stampaj_bitove(x);
143     printf("y = %10u %36s = ", y, "");
144     stampaj_bitove(y);
145     printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
146     stampaj_bitove(postavi_1_n_bitova(x, n, p, y));
147     printf("\n");
148
149     /* Ispisuju se binarne reprezentacije broja x i broja koji se
150        izracunava funkcijom invertuj sa argumentima x, n i p */
151     printf("x = %10u %36s = ", x, "");
152     stampaj_bitove(x);
153     printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
154     stampaj_bitove(invertuj(x, n, p));
155
156     return 0;
157 }
```

### 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 ulevo za n mesta. */
5 unsigned rotiraj_ulevo(int x, unsigned n)
6 {
```

```
8     unsigned bit_najvece_tezine;
10
12     /* Maska koja ima samo bit na poziciji najvece tezine postavljen
14     na 1, neophodna je da bi pre pomeranja x ulevo za 1, bit na
16     poziciji najvece tezine bio sacuvan. */
18     unsigned bit_najvece_tezine_maska =
20     1 << (sizeof(unsigned) * 8 - 1);
22     int i;
24
26     /* n puta se vrši rotacija za jedan bit ulevo. U svakoj iteraciji
28     se odredi bit na poziciji najvece tezine, a potom se pomera
30     binarna reprezentacija trenutne vrednosti promenljive x ulevo
32     za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
34     na vrednost koju je imao bit na poziciji najvece tezine koji je
36     istisnut pomeranjem */
38     for (i = 0; i < n; i++) {
40         bit_najvece_tezine = x & bit_najvece_tezine_maska;
42         x = x << 1 | (bit_najvece_tezine ? 1 : 0);
44     }
46
48     /* Vraca se izracunata vrednost */
50     return x;
52 }
54
56 /* Funkcija neoznaceni broj x rotira udesno za n mesta. */
58 unsigned rotiraj_udesno(unsigned x, unsigned n)
{
    unsigned bit_najmanje_tezine;
    int i;

    /* n puta se ponavlja rotacija udesno za jedan bit. U svakoj
    iteraciji se odredjuje bit na poziciji najmanje tezine broja
    x, zatim tako odredjeni bit se pomera ulevo tako da bit na
    poziciji najmanje tezine dodje do pozicije najvece tezine.
    Zatim, nakon pomeranja binarne reprezentacije trenutne
    vrednosti promenljive x za 1 udesno, bit na poziciji najvece
    tezine se postavlja na vrednost vec zapamcenog bita koji je
    bio na poziciji najmanje tezine. */
    for (i = 0; i < n; i++) {
        bit_najmanje_tezine = x & 1;
        x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
    }

    /* Vraca se izracunata vrednost */
    return x;
}

/* Verzija funkcije koja broj x rotira udesno za n mesta, gde je
argument funkcije x oznaceni ceo broj */
int rotiraj_udesno_oznaceni(int x, unsigned n)
{
    unsigned bit_najmanje_tezine;
```

## 1 Uvodni zadaci

```
int i;
60
/* U svakoj iteraciji se odredjuje bit na poziciji najmanje
62   tezone i smesta u promenljivu bit_najmanje_tezine. Kako je x
   oznacen ceo broj, tada se prilikom pomeranja udesno vrshi
64   aritmeticko pomeranje i cuva se znak broja. Dakle, razlikuju
   se dva slucaja u zavisnosti od znaka broja x. Nije dovoljno da
66   se ova provera izvrsi pre petlje, s obzirom da rotiranjem
   udesno na poziciju najvece tezone moze doci i 0 i 1, nezavisno
68   od pocetnog znaka broja smestenog u promenljivu x. */
for (i = 0; i < n; i++) {
70   bit_najmanje_tezine = x & 1;

72   if (x < 0)
   /******
74   Siftovanjem udesno broja koji je negativan dobija se 1 kao bit
   na poziciji najvece tezone. Na primer ako je x
76   1010 1011 1100 1101 1110 0001 0010 001b
   (sa b je oznacen ili 1 ili 0 na poziciji najmanje tezone)
78   Onda je sadrzaj promenljive bit_najmanje_tezine:
   0000 0000 0000 0000 0000 0000 0000 000b
80   Nakon siftovanja sadrzaja promenljive x za 1 udesno
   1101 0101 1110 0110 1111 0000 1001 0001
82   Kako bi umesto 1 na poziciji najvece tezone u trenutnoj binarnoj
   reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
84   poziciju najvece tezone jer bi se time dobile 0, a u ovom slucaju
   su potrebne jedinice zbog bitovskog & zato se prvo vrshi
86   komplementiranje, a zatim pomeranje
   ~bit_najmanje_tezine << (sizeof(int)*8 -1)
88   B000 0000 0000 0000 0000 0000 0000 0000
   gde B oznacava ~b.
90   Potom se ponovo vrshi komplementiranje kako bi se b nalazilo na
   poziciji najvece tezone i sve jedinice na ostalim pozicijama
92   ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
   b111 1111 1111 1111 1111 1111 1111 1111
94   *****/
   x = (x >> 1) & ~(~bit_najmanje_tezine <<
96     (sizeof(int) * 8 - 1));
   else
98     x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
}

100 /* Vraca se izracunata vrednost */
102 return x;
}

104 int main()
106 {
   unsigned x, n;

108   /* Ucitavaju se vrednosti sa standardnog ulaza */
110   printf("Unesite neoznaceni ceo broj x:");
```

```

scanf("%x", &x);
112 printf("Unesite neoznaceni ceo broj n:");
scanf("%x", &n);
114
/* Ispisuje se binarna reprezentacija broja x */
116 printf("x\t\t\t\t\t= ");
stampaj_bitove(x);
118
/* Testiraju se napisane funkcije */
120 printf("rotiraj_ulevo(%x,%u)\t\t= ", x, n);
stampaj_bitove(rotiraj_ulevo(x, n));
122
printf("rotiraj_udesno(%x,%u)\t\t= ", x, n);
124 stampaj_bitove(rotiraj_udesno(x, n));
126
printf("rotiraj_udesno_oznaceni(%x,%u)\t= ", x, n);
stampaj_bitove(rotiraj_udesno_oznaceni(x, n));
128
printf("rotiraj_udesno_oznaceni(-%x,%u)\t= ", x, n);
130 stampaj_bitove(rotiraj_udesno_oznaceni(-x, n));
132
return 0;
}

```

### Rešenje 1.10

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

```

1 #include <stdio.h>
#include "stampanje_bitova.h"
3
/* Funkcija vraća vrednost čija je binarna reprezentacija slika u
5 ogledalu binarne reprezentacije broja x. */
unsigned ogledalo(unsigned x)
7 {
    unsigned najnizi_bit;
9     unsigned rezultat = 0;
11
    int i;
    /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji
13 tekuce vrednosti broja x se određuje i pamti u promenljivoj
    najnizi_bit, nakon čega se na promenljivu x primeni pomeranje
15 u desno */
    for (i = 0; i < sizeof(x) * 8; i++) {
17         najnizi_bit = x & 1;
        x >>= 1;
19         /* Potiskivanjem trenutnog rezultata ka levom kraju svi
        prethodno postavljeni bitovi dobijaju veću poziciju. Novi
21 bit se postavlja na najnizu poziciju */
        rezultat <<= 1;
23         rezultat |= najnizi_bit;
    }
}

```

## 1 Uvodni zadaci

---

```
25 }
26
27 /* Vraca se izracunata vrednost */
28 return rezultat;
29 }
30
31 int main()
32 {
33     int broj;
34
35     /* Ucitava se broj sa ulaza */
36     scanf("%x", &broj);
37
38     /* Ispisuje se njegova binarna reprezentacija */
39     stampaj_bitove(broj);
40
41     /* Ispisuje se i binarna reprezentacija broja izracunatog pozivom
42     funkcije ogledalo */
43     stampaj_bitove(ogledalo(broj));
44
45     return 0;
46 }
```

### Rešenje 1.11

```
1 #include <stdio.h>
2
3 /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n
4 broj jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
5 int broj_01(unsigned int n)
6 {
7     int broj_nula, broj_jedinica;
8     unsigned int maska;
9
10    broj_nula = 0;
11    broj_jedinica = 0;
12
13    /* Maska je inicijalizovana tako da moze da analizira bit najvece
14    tezine */
15    maska = 1 << (sizeof(unsigned int) * 8 - 1);
16
17    /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
18    iteraciji pomera u desno pa ce jedini bit koji je postavljen
19    na 1 biti na svim pozicijama u binarnoj reprezentaciji maske */
20    while (maska != 0) {
21        /* Proverava se da li se na poziciji koju odredjuje maska
22        nalazi 0 ili 1 i uvecava se odgovarajuci brojac. */
23        if (n & maska) {
24            broj_jedinica++;
25        } else {
26            broj_nula++;
27        }
28        maska = maska >> 1;
29    }
30    return broj_jedinica > broj_nula;
31 }
```

```

    }
28
    /* Pomera se maska u desnu stranu */
30    maska = maska >> 1;
    }
32
    /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
34     suprotnom vraca 0 */
    return (broj_jedinica > broj_nula) ? 1 : 0;
36 }

38 int main()
    {
40     unsigned int n;

42     /* Ucitava se neoznaceni broj */
    scanf("%u", &n);

44     /* Ispisuje se rezultat */
46     printf("%d\n", broj_01(n));

48     return 0;
    }

```

### Rešenje 1.12

```

#include <stdio.h>
2
/* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju
4  u binarnom zapisu celog čneoznaenog broja x */
int broj_parova(unsigned int x)
6 {
    int broj_parova;
8     unsigned int maska;

10     /* Vrednost promenljive koja predstavlja broj parova se
        inicijalizuje na 0 */
12     broj_parova = 0;

14     /* Postavlja se maska tako da moze da procita da li su dva
        najmanja bita u zapisu broja x 11 */
16     /* Binarna reprezentacija broja 3 je 000...00011 */
    maska = 3;
18

19     while (x != 0) {
20         /* Proverava se da li se na najmanjim pozicijama broja x nalazi
                par bitova 11 */
22         if ((x & maska) == maska) {
                broj_parova++;
24         }
    }

```



## 1 Uvodni zadaci

```
26     /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
28         proveravao sledeci par bitova. Pomeranjem u desno bit
           najvece tezine se popunjava nulom jer je x neoznaceni broj */
30     x = x >> 1;
31 }
32 /* Vraca se izracunata vrednost */
33 return broj_parova;
34 }
35
36 int main()
37 {
38     unsigned int x;
39
40     /* Ucitava se neoznaceni broj */
41     scanf("%u", &x);
42
43     /* Ispisuje se rezultat */
44     printf("%d\n", broj_parova(x));
45
46     return 0;
47 }
```

### Rešenje 1.14

```
1 #include <stdio.h>
2
3 /* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1
4     jer su za svaku heksadekadnu cifru potrebne 4 binarne cifre i
5     jedna dodatna pozicija za terminirajucu nulu. Prethodni izraz se
6     moze zapisati kao sizeof(unsigned int)*2+1. */
7 #define MAKS_DUZINA sizeof(unsigned int)*2 +1
8
9 /* Funkcija za neoznaceni broj x formira nisku s koja sadrzi njegov
10    heksadekadni zapis */
11 void prevod(unsigned int x, char s[])
12 {
13     int i;
14     unsigned int maska;
15     int vrednost;
16
17     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
18        maska za citanje 4 uzastopne cifre */
19     maska = 15;
20
21     /******
22        Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
23        tezine. Na primer za broj cija je binarna reprezentacija
24        0000000001101000100001111010101
25        u prvom koraku se citaju bitovi izdvojeni sa <...>:
26        000000000110100010000111101<0101>
27     */
28 }
```

```
u drugom koraku:
28 000000000011010001000011<1101>0101
u trecem koraku:
30 00000000001101000100<0011>11010101 i tako redom...

Indeks i oznacava poziciju na koju se smesta vrednost.
*****/
32 for (i = MAKS_DUZINA - 2; i >= 0; i--) {
34 /* Vrednost izdvojene cifre */
36 vrednost = x & maska;

38 /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
40 dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega
42 od 10 do 15 odgovarajuci karakter se dobija tako sto se prvo
44 izracunatu vrednost dodaje ASCII kod 'A' (time se dobija
46 odgovarajuce slovo 'A', 'B', ... 'F') */
48 if (vrednost < 10) {
49     s[i] = vrednost + '0';
50 } else {
51     s[i] = vrednost - 10 + 'A';
52 }

53 /* Promenljiva x se pomera za 4 bita u desnu stranu i time se u
54 narednoj iteraciji posmatraju sledeca 4 bita */
55 x = x >> 4;
56 }

57 /* Upisuje se terminirajuca nula */
58 s[MAKS_DUZINA - 1] = '\0';
59 }

60 int main()
61 {
62     unsigned int x;
63     char s[MAKS_DUZINA];

64     /* Ucitava se broj sa ulaza */
65     scanf("%u", &x);

66     /* Poziva se funkcija za prevodjenje */
67     prevod(x, s);

68     /* I stampa se formirana niska */
69     printf("%s\n", s);

70     return 0;
71 }
72
73
74 }
```

## Rešenje 1.17

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /*****
5  Resenje linearne slozenosti:
6  x^0 = 1
7  x^k = x * x^(k-1)
8  *****/
9 int stepen(int x, int k)
10 {
11     if (k == 0)
12         return 1;
13
14     return x * stepen(x, k - 1);
15     /* kraci zapis: return k == 0 ? 1 : x * stepen(x,k-1); */
16 }
17
18 /*****
19  Resenje logaritamske slozenosti:
20  x^0 =1;
21  x^k = x * (x^2)^(k/2) - za neparno k
22  x^k = (x^2)^(k/2) - za parno k
23  Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
24  se doslo do rezultata, i stoga je efikasnije.
25  *****/
26 int stepen_2(int x, int k)
27 {
28     if (k == 0)
29         return 1;
30
31     /* Ukoliko je stepen k paran */
32     if ((k % 2) == 0)
33         return stepen_2(x * x, k / 2);
34
35     /* Ukoliko je stepen k neparan */
36     return x * stepen_2(x * x, k / 2);
37 }
38
39 int main()
40 {
41     int x, k, ind;
42
43     /* Ucitavanje rednog broja funkcije koja ce se primeniti */
44     printf("Unesite redni broj funkcije (1/2):\n");
45     scanf("%d", &ind);
46
47     /* Ucitavanje vrednosti za x i k */
48     printf("Unesite broj x:\n");
49     scanf("%d", &x);
50     printf("Unesite broj k:\n");
```

```

scanf("%d", &k);
52
/* Ispisuje se vrednost koju vraća odgovarajuća funkcija */
54 if (x == 1)
    printf("%d\n", stepen(x, k));
56 else if (x == 2)
    printf("%d\n", stepen_2(x, k));
58 else {
    fprintf(stderr,
60     "Greska: Neodgovarajući redni broj funkcije.\n");
    exit(EXIT_FAILURE);
62 }

64 exit(EXIT_SUCCESS);
}

```

### 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
   funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
8 definicije. Funkcija je mogla biti deklarirana i u telu funkcije
   paran. */
10 unsigned neparan(unsigned n);

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

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

32 int main()
   {

```

## 1 Uvodni zadaci

---

```
34  int n;
36  /* Ucitava se ceo broj */
   scanf("%d", &n);
38
   /* Ispisuje se rezultat */
40  printf("Uneti broj ima %s paran broj cifara.\n",
         (paran(n) == 1 ? "" : "ne"));
42
   return 0;
44 }
```

### Rešenje 1.19

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Pomocna funkcija koja izracunava n! * rezultat. Koristi repnu
   5   rekurziju. rezultat je argument u kome se akumulira do tada
       izracunatu vrednost faktorijela. Kada dodje do izlaza iz
   7   rekurzije iz rekurzije potrebno je da vratimo rezultat. */
   int faktorijel_repna(int n, int rezultat)
9   {
   11      if (n == 0)
           return rezultat;
13
   13      return faktorijel_repna(n - 1, n * rezultat);
   }
15
   /* U sledecim funkcijama je prikazan postupak oslobadjanja od repne
   17   rekurzije koja postoji u funkciji faktorijel_repna. */
19
   /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
   21   naredbama kojima se vrednost argumenta funkcije postavlja na
       vrednost koja bi se prosledjivala rekurzivnom pozivu i
   23   navodjenjem goto naredbe za vracanje na pocetak tela funkcije. */
   int faktorijel_repna_v1(int n, int rezultat)
   {
25   pocetak:
       if (n == 0)
27           return rezultat;
29
       rezultat = n * rezultat;
       n = n - 1;
31       goto pocetak;
   }
33
   /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra
   35   programerska praksa i prethodna funkcija se koristi samo kao
       medjukorak. Sledi iterativno resenje bez bezuslovnih skokova */
37   int faktorijel_repna_v2(int n, int rezultat)
```

```
{
39  while (n != 0) {
    rezultat = n * rezultat;
41    n = n - 1;
  }
43
  return rezultat;
45 }

47 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
49 broja n, mora da se salje i 1 za vrednost drugog argumenta u
kome ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde
radi udobnosti korisnika, jer je sasvim prirodno da za
51 faktorijel zahteva samo 1 parametar. Funkcija faktorijel
izracunava n!, tako sto odgovarajucoj gore navedenoj funkciji
53 koja zaista racuna faktorijel, salje ispravne argumente i vraca
rezultat koju joj ta funkcija vrati. Za testiranje, zameniti u
55 telu funkcije faktorijel poziv faktorijel_repna sa pozivom
faktorijel_repna_v1, a zatim sa pozivom funkcije
57 faktorijel_repna_v2. */
int faktorijel(int n)
59 {
  return faktorijel_repna(n, 1);
61 }

63 int main()
{
65   int n;

67   /* Ucitava se ceo broj */
printf("Unesite n (<= 12): ");
69   scanf("%d", &n);
if (n > 12) {
71     fprintf(stderr, "Greska: Nedozvoljena vrednost za n.\n");
exit(EXIT_FAILURE);
73   }

75   /* Ispisuje se rezultat poziva funkcije faktorijel */
printf("%d! = %d\n", n, faktorijel(n));
77
exit(EXIT_SUCCESS);
79 }
```

### 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;
11
12     if (n == 0)
13         return 0;
14
15     for (i = 2; i <= n; i++) {
16         tmp = a * f_1 + b * f_0;
17         f_0 = f_1;
18         f_1 = tmp;
19     }
20
21     return f_1;
22 }
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 slucaju da se rekurzijom problem svodi na vise
36 manjih potproblema koji se mogu preklapati, postoji opasnost da
37 se pojedini potproblemi manjih dimenzija resavaju veci broj
38 puta. Npr.  $F(20) = a \cdot F(19) + b \cdot F(18)$ , a  $F(19) = a \cdot F(18) +$ 
39  $b \cdot F(17)$ , tj. 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 dinamicnim programiranjem.
42 Potproblemi se resavaju samo jednom, a njihova resenja se pamte
43 u memoriji (obicno u nizovima ili matricama), odakle se koriste
44 ako tokom resavanja ponovo budu potrebni.
45
46 U narednoj funkciji vec izracunati clanovi niza se cuvaju u
47 statickom nizu celih brojeva, jer se staticki niz ne smesta na
48 stek, kao sto je to slucaj sa lokalnim promenljivama, vec na
49 segment podataka, odakle je dostupan svim pozivima rekurzivne
50 funkcije. */
```

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

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

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

69 /* Rekurzivni pozivi */
71 return f[n] =
    a * f_napredna(n - 1, a, b) + b * f_napredna(n - 2, a, b);
73 }

75 int main()
{
77 int n, a, b, ind;

79 /* Unosi se redni broj funkcije koja ce se primeniti */
    printf("Unesite redni broj funkcije:\n");
81 printf("1 - iterativna\n");
    printf("2 - rekurzivna\n");
83 printf("3 - rekurzivna napredna\n");
    scanf("%d", &ind);

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

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

93 /* Na osnovu vrednosti promenljive ind ispisuje se rezultat
95 poziva funkcije f_iterativna, f_rekurzivna ili f_napredna */
    if (ind == 0)
97 printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
    else if (ind == 1)
99 printf("F(%d) = %d\n", n, f_rekurzivna(n, a, b));
    else
101 printf("F(%d) = %d\n", n, f_napredna(n, a, b));
```



## 1 Uvodni zadaci

---

```
103 | return 0;
    | }
```

### Rešenje 1.22

```
#include <stdio.h>
2
/* Funkcija odredjuje zbir cifara zadatog broja x */
4 int zbir_cifara(unsigned int x)
{
6   /* Izlazak iz rekurzije: ako je broj jednocifren */
   if (x < 10)
8     return x;

10  /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
    poslednje cifre + poslednja cifra tog broja */
12  return zbir_cifara(x / 10) + x % 10;
}

14
int main()
16 {
   unsigned int x;
18
   /* Ucitava se ceo broj */
20  scanf("%u", &x);

22  /* Ispisuje se zbir cifara ucitanog broja */
   printf("%d\n", zbir_cifara(x));
24
   return 0;
26 }
```

### Rešenje 1.23

```
#include <stdio.h>
2 #include <stdlib.h>
   #define MAKS_DIM 100
4
/* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
6  suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
   element niza. */
8 int suma_niza_1(int *a, int n)
{
10  if (n <= 0)
     return 0;
12
   return suma_niza_1(a, n - 1) + a[n - 1];
14 }
```

```

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

```

### Rešenje 1.24

```

#include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
5     dimenzije n */

```

## 1 Uvodni zadaci

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

12
   /* Resava se problem manje dimenzije */
14  int max = maksimum_niza(niz, n - 1);

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

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

26  /* Sve dok se ne stigne do kraja ulaza, brojevi se ucitavaju u
   niz. Promenljiva i predstavlja indeks tekućeg broja. U niz se
28  ne može ucitati više od MAKS_DIM brojeva, pa se u slučaju da
   promenljiva i dostigne vrednost MAKS_DIM prekida unos novih
30  brojeva. */
   int i = 0;
32  while (scanf("%d", &brojevi[i]) != EOF) {
      i++;
34   if (i == MAKS_DIM)
       break;
36  }
   n = i;

38
   /* Stampa se maksimum unetog niza brojeva */
40  printf("%d\n", maksimum_niza(brojevi, n));

42  return 0;
  }
```

### Rešenje 1.25

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

5 /* Funkcija koja izracunava skalarni proizvod dva data vektora */
int skalarno(int a[], int b[], int n)
7 {
   /* Izlazak iz rekurzije: vektori su duzine 0 */
9   if (n == 0)
       return 0;
```

```

11  /* Na osnovu resenja problema dimenzije n-1, resava se problem
13  dimenzije n primenom definicije skalarnog proizvoda a*b =
15  a[0]*b[0] + a[1]*b[1] +...+ a[n-2]*a[n-2] + a[n-1]*a[n-1]
17  Dakle, skalarni proizvod dva vektora duzine n se dobija kada
19  se na skalarni proizvod dva vektora duzine n-1 koji se dobiju
21  od polazna dva vektora otklanjanjem poslednjih elemenata, doda
23  proizvod poslednja dva elementa polaznih vektora. */
25  else
27      return skalarno(a, b, n - 1) + a[n - 1] * b[n - 1];
29  }

31  int main()
33  {
35      int i, a[MAKS_DIM], b[MAKS_DIM], n;

37      /* Unosi se dimenzija nizova */
39      printf("Unesite dimenziju nizova:");
41      scanf("%d", &n);

43      /* Provera da li je dimenzija niza odgovarajuca */
45      if (n < 0 || n > MAKS_DIM) {
47          fprintf(stderr,
49              "Greska: Dimenzija mora biti prirodan broj <= %d!\n",
51              MAKS_DIM);
53          exit(EXIT_FAILURE);
55      }

57      /* Unose se elementi nizova */
59      printf("Unesite elemente prvog niza:");
61      for (i = 0; i < n; i++)
63          scanf("%d", &a[i]);

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

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

75      exit(EXIT_SUCCESS);
77  }

```

### 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)

```

## 1 Uvodni zadaci

---

```
{
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)
    return a[0] == x ? 1 : 0;
12
14  /* U promenljivu bp se smesta broj pojava broja x u prvih n-1
   elemenata niza a. Ukupan broj pojavljivanja broja x u celom
   nizu a je jednak bp uvecanom za jedan ukoliko je se na
16  poziciji n-1 u nizu a nalazi broj x */
   int bp = br_pojave(x, a, n - 1);
18  return a[n - 1] == x ? 1 + bp : bp;
}

20
22  int main()
23  {
   int x, a[MAKS_DIM];
24  int n, i = 0;

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

30  /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u
   niz. Promenljiva i predstavlja indeks tekuceg broja. U niz se
32  ne moze ucitati vise od MAKS_DIM brojeva, pa se u slucaju da
   promenljiva i dostigne vrednost MAKS_DIM prekida unos novih
34  brojeva. */
   printf("Unesite elemente niza:");
   i = 0;
   while (scanf("%d", &a[i]) != EOF) {
38     i++;
     if (i == MAKS_DIM)
40         break;
   }
42  n = i;

44  /* Ispisuje se broj pojavljivanja */
   printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
46
   return 0;
48 }
```

## Rešenje 1.27

```
1 #include <stdio.h>
2 #define MAKS_DIM 256
3
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
9      *   vraca se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi
10     *   niza */
11     if (n < 3)
12         return 0;
13
14     /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
15     *   dovoljno je da su oni poslednja tri clana niza ili da se oni
16     *   rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
17     return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
18         || tri_uzastopna_clana(x, y, z, a, n - 1);
19 }
20
21 int main()
22 {
23     int x, y, z, a[MAKS_DIM], 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
31     *   niz. Promenljiva i predstavlja indeks tekuceg broja. U niz se
32     *   ne moze ucitati vise od MAKS_DIM brojeva, pa se u slucaju da
33     *   promenljiva i dostigne vrednost MAKS_DIM prekida unos novih
34     *   brojeva. */
35     printf("Unesite elemente niza:");
36     int i = 0;
37     while (scanf("%d", &a[i]) != EOF) {
38         i++;
39         if (i == MAKS_DIM)
40             break;
41     }
42     n = i;
43
44     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana
45     *   ispisuje se odgovarajuca poruka */
46     if (tri_uzastopna_clana(x, y, z, a, n))
47         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
48     else
49         printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
```

## 1 Uvodni zadaci

---

```
51     return 0;
    }
```

### Rešenje 1.28

```
#include <stdio.h>

2
/* Funkcija koja broji bitove postavljene na 1. */
4 int prebroj(int x)
   {
6     /* Izlaz iz rekurzije */
     if (x == 0)
8         return 0;

10    /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x
        je postavljen na 1. Koriscenjem odgovarajuće maske proverava
12    se vrednost bita na poziciji najveće težine i na osnovu toga
        se razlikuju dva slučaja. Ukoliko je na toj poziciji nula,
14    onda je broj jedinica u zapisu x isti kao broj jedinica u
        zapisu broja x<<1, jer se pomeranjem u levo sa desne stane
16    dopisuju 0. Ako je na poziciji najveće težine jedinica,
        rezultat dobijen rekurzivnim pozivom funkcije za x<<1 treba
18    uvećati za jedan. Za rekurzivni poziv se šalje vrednost koja
        se dobija kada se x pomeri u levo. Napomena: argument funkcije
20    x je označen ceo broj, usled čega se ne koristi pomeranje
        udesno, jer funkciji može biti prosledjen i negativan broj. Iz
22    tog razloga, odlučujemo se da proveramo najvisi, umesto
        najnižeg bita */
24    if (x & (1 << (sizeof(x) * 8 - 1)))
        return 1 + prebroj(x << 1);
26    else
        return prebroj(x << 1);
28    /******
        Krace zapisano
30    return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + prebroj(x<<1);
        *****/
32 }

34 int main()
   {
36     int x;

38     /* Ucitava se ceo broj */
     scanf("%x", &x);

40     /* Ispisuje se rezultat */
     printf("%d\n", prebroj(x));

42     return 0;
44 }
```

## Rešenje 1.30

```

2  #include <stdio.h>
4  /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre */
4  int maks_oktalna_cifra(unsigned x)
6  {
6     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
6        vrednost najveće oktalne cifre u broju 0 */
8     if (x == 0)
8         return 0;
10
10    /* Odredjuje se poslednja oktalna cifra u broju */
12    int poslednja_cifra = x & 7;
14
14    /* Odredjuje se maksimalna oktalna cifra u broju kada se iz njega
14       izbrise poslednja oktalna cifra */
16    int maks_bez_poslednje_cifre = maks_oktalna_cifra(x >> 3);
18
18    return poslednja_cifra >
18        maks_bez_poslednje_cifre ? poslednja_cifra :
20        maks_bez_poslednje_cifre;
22 }
24
24 int main()
24 {
26     unsigned x;
28
28     /* Ucitava se neoznaceni ceo broj */
28     scanf("%u", &x);
30
30     /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
32     printf("%d\n", maks_oktalna_cifra(x));
34
34     return 0;
34 }

```

## Rešenje 1.31

```

2  #include <stdio.h>
4  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre */
4  int maks_heksadekadna_cifra(unsigned x)
6  {
6     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
6        vrednost najveće heksadekadne cifre u broju 0 */
8     if (x == 0)
8         return 0;
10
10    /* Odredjuje se poslednja heksadekadna cifra u broju */

```



## 1 Uvodni zadaci

```
12  int poslednja_cifra = x & 15;
14  /* Odredjuje se maksimalna heksadekadna cifra broja kada se iz
16  njega izbrise poslednja heksadekadna cifra */
16  int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);
18  return poslednja_cifra >
20      maks_bez_poslednje_cifre ? poslednja_cifra :
20      maks_bez_poslednje_cifre;
22  }
24  int main()
24  {
26      unsigned x;
28      /* Ucitava se neoznaceni ceo broj */
28      scanf("%u", &x);
30      /* Ispisivanje vrednosti najveće heksadekadne cifre unetog broja */
30      printf("%d\n", maks_heksadekadna_cifra(x));
32      return 0;
34  }
```

### Rešenje 1.32

```
#include <stdio.h>
2 #include <string.h>
4 /* Niska može imati najviše 31 karaktera + 1 za terminalnu nulu */
4 #define MAKS_DIM 32
6 /* Funkcija ispituje da li je zadata niska dužine n palindrom */
8 int palindrom(char s[], int n)
8 {
10     /* Izlaz iz rekurzije - trivijalno, niska dužine 0 ili 1 je
10     palindrom */
12     if ((n == 1) || (n == 0))
12         return 1;
14     /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
16     poslednji karakter i da je palindrom niska koja nastaje kada
16     se polaznoj nisci otklone prvi i poslednji karakter */
18     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
18 }
20
20 int main()
22 {
22     char s[MAKS_DIM];
24     int n;
```

```

26  /* Ucitava se niska sa standardnog ulaza */
    scanf("%s", s);
28
    /* Odredjuje se duzina niske */
30  n = strlen(s);
32
    /* Ispisuje se da li je niska palindrom ili nije */
    if (palindrom(s, n))
34      printf("da\n");
    else
36      printf("ne\n");
38  return 0;
}

```

### Rešenje 1.33

```

1  #include <stdio.h>
    #include <stdlib.h>
3  #define MAKS_DUZINA_PERMUTACIJE 15
5  /* Funkcija koja ispisuje elemente niza a duzine n */
    void ispisi_niz(int a[], int n)
7  {
    int i;
9
    for (i = 1; i <= n; i++)
11     printf("%d ", a[i]);
    printf("\n");
13 }
15 /* Funkcija koja vraca 1, ako se broj x nalazi u nizu a duzine n,
    inace vraca 0 */
17 int koriscen(int a[], int n, int x)
    {
19     int i;
21
    /* Obilaze se svi elementi niza */
    for (i = 1; i <= n; i++)
23     /* Ukoliko se naidje na trazenu vrednost, pretraga se prekida */
        if (a[i] == x)
25         return 1;
27
    /* Zakljucuje se da broj nije pronadjen */
    return 0;
29 }
31 /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n}
    dobija kao argument niz a[] u koji se smesta permutacija, broj m
33     oznacava da se u okviru tog poziva funkcije na m-tu poziciju u
    permutaciji smesta jedan od preostalih celih brojeva, n je

```

## 1 Uvodni zadaci

```
35     velicina skupa koji se permutuje. Funkciju se inicijalno poziva
36     sa argumentom m = 1, jer formiranje permutacije pocinje od
37     pozicije broj 1. Stoga, a[0] se ne koristi. */
void permutacija(int a[], int m, int n)
39 {
40     int i;
41
42     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti
43     broj premasila velicinu skupa, onda se svi brojevi vec nalaze
44     u permutaciji i ispisuje se permutacija. */
45     if (m > n) {
46         ispisi_niz(a, n);
47         return;
48     }
49
50     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
51     mesto u nizu (broj koji se do sada nije pojavio u
52     permutaciji). Zatim, rekurzivno se pronalaze one permutacije
53     koje odgovaraju ovako postavljenom pocetku permutacije. Kada
54     se to zavrshi, vrsi se provera da li postoji jos neki broj koji
55     moze da se stavi na m-to mesto u nizu (to se radi u petlji).
56     Ako ne postoji, funkcija zavrшава sa radom. Ukoliko takav broj
57     postoji, onda se ponovo poziva rekurzivno pronalazenje
58     odgovarajucih permutacija, ali sada sa drugacije postavljenim
59     prefiksom. */
60     for (i = 1; i <= n; i++) {
61         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
62         pozicije, onda se on postavlja na poziciju m i poziva se
63         ponovo funkcija da dopuni ostatak permutacije posle
64         upisivanja i na poziciju m. Inace, nastavlja se dalje, trazi
65         se broj koji se nije pojavio do sada u permutaciji. */
66         if (!koriscen(a, m - 1, i)) {
67             a[m] = i;
68             permutacija(a, m + 1, n);
69         }
70     }
71 }
72
73 int main(void)
74 {
75     int n;
76     int a[MAKS_DUZINA_PERMUTACIJE + 1];
77
78     /* Ucitava se broj n iz odgovarajuceg opsega */
79     scanf("%d", &n);
80     if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
81         fprintf(stderr,
82             "Greska: Duzina permutacije mora biti broj iz intervala
83             [0, %d]!\n",
84             MAKS_DUZINA_PERMUTACIJE);
85         exit(EXIT_FAILURE);
86     }
87 }
```

```

87  /* Ispisuju se permutacije duzine n */
    permutacija(a, 1, n);
89
    exit(EXIT_SUCCESS);
91 }

```

### Rešenje 1.34

```

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

/* Rekurzivna funkcija za racunanje binomnog koeficijenta */
int binomni_koeficijent(int n, int k)
{
    /* Ukoliko je k=0 ili k=n, onda je binomni koeficijent 0. Ukoliko
       je k strogo izmedju 0 i n, onda se koristi formula  $b_k(n,k) =
       b_{k-1}(n,k-1) + b_k(n-1,k)$  koja se moze izvesti iz definicije
       binomnog koeficijenata */
    return (0 < k && k < n) ?
        binomni_koeficijent(n - 1, k - 1) +
        binomni_koeficijent(n - 1, k) : 1;
}

/*****
Iterativno izracunavanje binomnog koeficijenta
*****/

int binomni_koeficijent (int n, int k) {
    int i, j, b;

    for (b=i=1, j=n; i<=k; b =b * j-- / i++);

    return b;
}

Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
*****/

/* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
   zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
   ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
   veca od sume elemenata prethodne hipotenuze. */
int suma_elemenata_hipotenuze(int n)
{
    return n > 0 ? 2 * suma_elemenata_hipotenuze(n - 1) : 1;
}

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

```

## 1 Uvodni zadaci

---

```
44  /* Ucitavaju se brojevi d i r */
    scanf("%d %d", &d, &r);
46
    /* Ispisuje se Paskalov trougao */
    putchar('\n');
48  for (n = 0; n <= d; n++) {
50      for (i = 0; i < d - n; i++)
52          printf(" ");
54      for (k = 0; k <= n; k++)
56          printf("%4d", binomni_koeficijent(n, k));
58      putchar('\n');
60  }
62
    /* Proverava se da li je r nenegativan */
    if (r < 0) {
64        fprintf(stderr,
66            "Greska: Redni broj hipotenuze mora biti veci ili jednak
        od 0!\n");
        exit(EXIT_FAILURE);
68    }
    printf("%d\n", suma_elementa_hipotenuze(r));
    exit(EXIT_SUCCESS);
}
```

## 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
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuca dimenzija niza.
```

[Rešenje 2.1]

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

## 2 Pokazivači

---

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

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

### Primer 1

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

### Primer 2

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

[Rešenje 2.2]

**Zadatak 2.3** Korišćenjem pokazivačke sintakse, napisati funkciju koja vrednosti elemenata u prvoj polovini niza povećava za jedan, a u drugoj polovini smanjuje za jedan. Ukoliko niz ima neparan broj elemenata, onda vrednost srednjeg elementa niza ostaviti nepromenjenim. Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojnog niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom.

### Primer 1

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

### Primer 2

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

### Primer 3

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju niza: 0
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuca dimenzija niza.
```

### Primer 4

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju niza: 101
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuca dimenzija niza.
```

[Rešenje 2.3]

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

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

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

### Primer 1

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

### Primer 2

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

[Rešenje 2.4]

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

### Primer 1

```
POKRETANJE: ./a.out a b 11 212
INTERAKCIJA SA PROGRAMOM:
Broj argumenata
koji su palindromi je 4.
```

### Primer 2

```
POKRETANJE: ./a.out
INTERAKCIJA SA PROGRAMOM:
Broj argumenata
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

```
POKRETANJE: ./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

```
POKRETANJE: ./a.out ulaz.txt 2
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA SA PROGRAMOM:
IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

### Primer 3

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj
ima reci koje imaju 1 karakter
INTERAKCIJA SA PROGRAMOM:
IZLAZ ZA GREŠKE:
Greska: Nedovoljan broj
argumenata komandne linije.
Program se poziva sa
./a.out ime_dat br_karaktera
```

[Rešenje 2.6]

**Zadatak 2.7** Napisati program koji kao prvi argument komandne linije prihvata putanju do datoteke za koju treba proveriti koliko reči ima zadati sufiks (ili prefiks), koji se zadaje kao drugi argument komandne linije. Smatrati da reč ne sadrži više od 100 karaktera. Program je neophodno pozvati sa jednom od opcija `-s` ili `-p`. U zavisnosti od opcije program proverava 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

```
POKRETANJE: ./a.out ulaz.txt ke -s
ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj ima
reci koje se završavaju na ke
INTERAKCIJA SA PROGRAMOM:
Broj reci koje se završavaju na ke je 2.
```

### Primer 2

```
POKRETANJE: ./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*

```

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

```

*Primer 4*

```

POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
Ovo je sadrzaj ulaza.
INTERAKCIJA SA PROGRAMOM:
IZLAZ ZA GREŠKE:
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 učitane matricu, a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

*Primer 1*

```

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

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 0
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuca 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 po vrstama:
1 2 3
1 2 3
1 2 3
Unesite elemente druge matrice po vrstama:
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 zatvorenje relacije (najmanju refleksivnu relaciju koja je nadskup date).
- (e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja je nadskup date).
- (f) Napisati funkciju koja određuje refleksivno-tranzitivno zatvorenje relacije (najmanju refleksivnu i tranzitivnu relaciju koja sadrži datu). NAPOMENA: *Koristiti Varšalov algoritam.*

Napisati program koji učitava matricu iz datoteke čije se ime zadaje kao prvi argument komandne linije. U prvoj liniji datoteke nalazi se 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*

```

POKRETANJE: ./a.out ulaz.txt

ULAZ.TXT
4
1 0 0 0
0 1 1 0
0 0 1 0
0 0 0 0

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

```

[Rešenje 2.10]

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

- (a) Napisati funkciju koja određuje najveći element matrice na sporednoj dijagonali.

## 2 Pokazivači

---

- (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

```
POKRETANJE: ./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 najvećim elementom je 2.
Broj negativnih elemenata matrice je 3.
```

### Primer 2

```
POKRETANJE: ./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 najvećim elementom je 0.
Broj negativnih elemenata matrice je 16.
```

[Rešenje 2.11]

**Zadatak 2.12** Napisati funkciju kojom se proverava da li je zadata kvadratna matrica dimenzije  $n \times n$  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 po vrstama:
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 po vrstama:
1 2 3
5 6 7
1 4 2
Matrica nije ortonormirana.
    
```

[Rešenje 2.12]

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

- (a) Napisati funkciju koja učitava elemente matrice sa standardnog ulaza
- (b) Napisati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice, 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 po vrstama:
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 po vrstama:
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 po vrstama:
1 2 3
4 5 6
7 8 9
Unesite stepen koji se racuna: 8
8. stepen matrice je:
510008400 626654232 743300064
1154967822 1419124617 1683281412
1799927244 2211595002 2623262760
```

## 2.3 Dinamička alokacija memorije

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

### Primer 1

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

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju niza: -1
Greska: Neuspesna 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. Na standardni izlaz ispisati ovaj niz brojeva u obrnutom poretku. Ne praviti nikakve pretpostavke o dimenziji niza. Zadatak uraditi na dva načina:

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

Od korisnika tražiti da izabere način realokacije memorije.

## 2.3 Dinamička alokacija memorije

### Primer 1

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

### Primer 2

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

[Rešenje 2.16]

**Zadatak 2.17** Napisati funkciju koja kao rezultat vraća nisku koja se dobija nadovezivanjem dve niske, bez promene njihovog sadržaja. Napisati program koji testira rad napisane funkcije. Sa standardnog ulaza učitati dve niske karaktera. Pretpostaviti da niske neće biti duže od 50 karaktera i da neće sadržati 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, a zatim i elementi matrice. Na standardni izlaz ispisati trag matrice. Ne praviti nikakve pretpostavke o maksimalnoj dimenziji matrice.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona:
2 3
Unesite elemente matrice po vrstama:
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:
2 2
Unesite elemente matrice po vrstama:
-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.



## 2 Pokazivači

---

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

Napisati programe koji testiraju napisanu biblioteku.

## 2.3 Dinamička alokacija memorije

- (1) Sa standardnog ulaza učitati broj vrsta i broj kolona matrice, a zatim i elemente matrice. Nakon toga sadržaj matrice upisati u datoteku *matrica.txt*.

<p><i>Primer 1</i></p> <pre>INTERAKCIJA SA PROGRAMOM: Unesite broj vrsta matrice: 3 Unesite broj kolona matrice: 4 Unesite 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</pre>	<p><i>Primer 2</i></p> <pre>INTERAKCIJA SA PROGRAMOM: Unesite broj vrsta matrice: 5 Unesite broj kolona matrice: 0 IZLAZ ZA GREŠKE: Greska: Broj vrsta i broj kolona ne mogu biti negativni brojevi.</pre>
--	--

- (2) Program prima kao prvi argument komandne linije putanju do datoteke u kojoj se, redom, nalaze dimenzija i elementi kvadratne matrice. Zatim učitava matricu i ispisuje je na standardni izlaz.

<p><i>Test 1</i></p> <pre>POKRETANJE: ./a.out ulaz.txt  ULAZ.TXT 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  IZLAZ: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</pre>	<p><i>Test 2</i></p> <pre>POKRETANJE: ./a.out ulaz.txt  ULAZ.TXT 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  IZLAZ ZA GREŠKE: Greska: Neispravan pocetak datoteke.</pre>	<p><i>Test 3</i></p> <pre>POKRETANJE: ./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 za rad sa celobrojnim matricama iz zadatka 2.19.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona:
2 3
Unesite elemente matrice po vrstama:
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 za rad sa celobrojnim matricama iz zadatka 2.19.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona:
2 3
Unesite elemente matrice po vrstama:
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:
2 4
Unesite elemente matrice po vrstama:
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

```
POKRETANJE: ./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 dimenzije  $m \times n$  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 datoteke se nalaze broj vrsta  $m$  i broj kolona  $n$  matrica, 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

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

### Primer 1

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

### Primer 2

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

## 2 Pokazivači

**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_slicice ime_reprezentacije_kojoj_slicica_pripada`  
Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronađe ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. NAPOMENA: *Koristiti `realloc()` funkciju za realokaciju memorije.*

### Primer 1

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

### INTERAKCIJA SA PROGRAMOM:

```
Petru ukupno nedostaje 7 slicica.
Reprezentacija za koju je sakupio
najmanji broj slicica je Brazil.
```

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

### Primer 1

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

### INTERAKCIJA SA PROGRAMOM:

```
U datoteci su zadata temena
cetvorougla.
Obim je 8.
Povrsina je 4.
```

### Primer 2

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

### INTERAKCIJA SA PROGRAMOM:

```
U datoteci su zadata temena
petougla.
Obim je 18.80.
Povrsina je 22.59.
```

## 2.4 Pokazivači na funkcije

**Zadatak 2.27** Napisati program koji tabelarno štampa vrednosti proizvoljne realne funkcije sa jednim realnim argumentom, odnosno izračunava i ispisuje vrednosti date funkcije u  $n$  ekvidistantnih tačaka na intervalu  $[a, b]$ . Realni

## 2.4 Pokazivači na funkcije

brojevi  $a$  i  $b$  ( $a < b$ ), kao i ceo broj  $n$  ( $n \geq 2$ ), učitavaju se 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

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

### Primer 2

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

[Rešenje 2.27]

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

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

Sa standardnog ulaza uneti ime funkcije i vrednosti  $n$  i  $a$ .

### 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 indeksne 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 }

```

```
16 }
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 sto
31            za posledicu ima da prvi pokazuje na sledeci element u nizu */
32         *prvi++ = *poslednji;
33
34         /* Vrednost promenljive t se postavlja na adresu na koju
35            pokazuje pokazivac poslednji. Ovaj pokazivac se zatim
36            umanjuje za jedan, cime pokazivac poslednji pokazuje na
37            element koji mu prethodi u nizu */
38         *poslednji-- = t;
39     }
40
41     /******
42     Drugi nacín za obrtanje niza
43
44     for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
45          prvi++, poslednji--) {
46         int t = *prvi;
47         *prvi = *poslednji;
48         *poslednji = t;
49     }
50     *****/
51 }
52
53 int main()
54 {
55     /* Deklarise se niz od najviše MAX elemenata */
56     int a[MAX];
57
58     /* Broj elemenata niza a */
59     int n;
60
61     /* Pokazivac na elemente niza */
62     int *p;
63
64     printf("Unesite dimenziju niza: ");
65     scanf("%d", &n);
66
67     /* Proverava se da li je doslo do prekoracenja ogranicenja
```



## 2 Pokazivači

```
68     dimenzije */
69     if (n <= 0 || n > MAX) {
70         fprintf(stderr, "Greska: Neodgovarajuca dimenzija niza.\n");
71         exit(EXIT_FAILURE);
72     }

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

78     obrni_niz_v1(a, n);

80     printf("Nakon obrtanja elemenata, niz je:\n");
81     for (p = a; p - a < n; p++)
82         printf("%d ", *p);
83     printf("\n");

84     obrni_niz_v2(a, n);

86     printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
87     for (p = a; p - a < n; p++)
88         printf("%d ", *p);
89     printf("\n");

92     exit(EXIT_SUCCESS);
93 }
```

### Rešenje 2.2

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

4 #define MAX 100

6 /* Funkcija izracunava zbir elemenata niza */
7 double zbir(double *a, int n)
8 {
9     double s = 0;
10    int i;

12    for (i = 0; i < n; s += *(a + i++));

14    return s;
15 }

16 /* Funkcija izracunava proizvod elemenata niza */
17 double proizvod(double *a, int n)
18 {
19     double p = 1;

22    for (; n; n--)
```

```
    p *= *(a + n - 1));
24
    return p;
26 }

28 /* Funkcija odredjuje minimalni element niza */
double min(double *a, int n)
30 {
    /* Na pocetku, minimalni element je prvi element */
32     double min = *a;
    int i;

34     /* Ispituje se da li se medju ostalim elementima niza nalazi
36         minimalni */
    for (i = 1; i < n; i++)
38         if (*(a + i) < min)
            min = *(a + i);

40     return min;
42 }

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

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

56     return max;
58 }

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

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

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

74     printf("Unesite elemente niza:\n");
```

## 2 Pokazivači

```
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
4 #define MAX 100
5
6 /* Funkcija povecava za jedan sve elemente u prvoj polovini niza a
7    smanjuje za jedan sve elemente u drugoj polovini niza. Ukoliko
8    niz ima neparan broj elemenata, srednji element ostaje
9    nepromenjen */
10 void povecaj_smanji(int *a, int n)
11 {
12     int *prvi = a;
13     int *poslednji = a + n - 1;
14
15     while (prvi < poslednji) {
16
17         /* Uvecava se element na koji pokazuje pokazivac prvi */
18         (*prvi)++;
19
20         /* Pokazivac prvi se pomera na sledeci element */
21         prvi++;
22
23         /* Smanjuje se vrednost elementa na koji pokazuje pokazivac
24            poslednji */
25         (*poslednji)--;
26
27         /* Pokazivac poslednji se pomera na prethodni element */
28         poslednji--;
29     }
30
31     /******
32     Drugi nacin:
33     while (prvi < poslednji) {
34         (*prvi++)++;
35         (*poslednji--)--;
36     }
37     *****/
```

```

38 }
40 int main()
41 {
42     int a[MAX];
43     int n;
44     int *p;
45
46     printf("Unesite dimenziju niza: ");
47     scanf("%d", &n);
48
49     /* Proverava se da li je doslo do prekoracenja ogranicenja
50        dimenzije */
51     if (n <= 0 || n > MAX) {
52         fprintf(stderr, "Greska: Neodgovarajuca dimenzija niza.\n");
53         exit(EXIT_FAILURE);
54     }
55
56     printf("Unesite elemente niza:\n");
57     for (p = a; p - a < n; p++)
58         scanf("%d", p);
59
60     povecaj_smanji(a, n);
61
62     printf("Transformisan niz je:\n");
63     for (p = a; p - a < n; p++)
64         printf("%d ", *p);
65     printf("\n");
66
67     exit(EXIT_SUCCESS);
68 }

```

## 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     printf("Kako zelite da ispisete argumente? Koriscenjem"
10           " indeksne ili pokazivacke sintakse (I ili P)? ");
11     scanf("%c", &tip_ispisa);
12
13     printf("Argumenti komandne linije su:\n");
14     if (tip_ispisa == 'I') {
15         /* Ispisuju se argumenti komandne linije koriscenjem indeksne
16            sintakse */
17         for (i = 0; i < argc; i++)

```

## 2 Pokazivači

```
18     printf("%d %s\n", i, argv[i]);
19 } else if (tip_ispisa == 'P') {
20     /* Ispisuju se argumenti komandne linije koriscenjem
21        pokazivacke sintakse */
22     i = argc;
23     for (; argc > 0; argc--)
24         printf("%d %s\n", i - argc, *argv++);
25
26     /* Nakon ove petlje argc je jednako nuli, a argv pokazuje na
27        polje u memoriji koje se nalazi iza poslednjeg argumenta
28        komandne linije. Kako je u promenljivoj i sacuvana vrednost
29        broja argumenta komandne linije to sada moze ponovo da se
30        postavi argv da pokazuje na nulti argument komandne linije */
31     argv = argv - i;
32     argc = i;
33 }
34
35 printf("Pocetna slova argumenata komandne linije:\n");
36 if (tip_ispisa == 'I') {
37     /* koristeći indeksnu sintaksu */
38     for (i = 0; i < argc; i++)
39         printf("%c ", argv[i][0]);
40     printf("\n");
41 } else if (tip_ispisa == 'P') {
42     /* koristeći pokazivacku sintaksu */
43     for (i = 0; i < argc; i++)
44         printf("%c ", **argv++);
45     printf("\n");
46 }
47
48 return 0;
49 }
```

### Rešenje 2.5

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX 100
5
6 /* Funkcija ispituje da li je niska palindrom, odnosno da li se
7    isto cita spreda i odpozadi */
8 int palindrom(char *niska)
9 {
10     int i, j;
11     for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
12         if (*(niska + i) != *(niska + j))
13             return 0;
14     return 1;
15 }
```

```

17 int main(int argc, char **argv)
18 {
19     int i, n = 0;
20
21     /* Multi argument komandne linije je ime izvrsnog programa */
22     for (i = 1; i < argc; i++)
23         if (palindrom(*(argv + i)))
24             n++;
25
26     printf("Broj argumenata koji su palindromi je %d.\n", n);
27
28     return 0;
29 }

```

## Rešenje 2.6

```

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

#define MAX_KARAKTERA 100

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

int main(int argc, char **argv)
{
    char rec[MAX_KARAKTERA + 1];
    int br = 0, n;
    FILE *in;

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

    /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
    komandne linije. */
    in = fopen(*(argv + 1), "r");
    if (in == NULL) {

```

## 2 Pokazivači

```
36     fprintf(stderr, "Greska: ");
37     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
38     exit(EXIT_FAILURE);
39 }
40
41 n = atoi(*(argv + 2));
42
43 /* Broje se reci cija je duzina jednaka broju zatom drugim
44    argumentom komandne linije */
45 while (fscanf(in, "%s", rec) != EOF)
46     if (duzina(rec) == n)
47         br++;
48
49 printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);
50
51 /* Zatvara se datoteka */
52 fclose(in);
53
54 exit(EXIT_SUCCESS);
55 }
```

### 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 }
```

```
29     return i;
30 }
31
32 /* Funkcija ispituje da li je niska zadata drugim argumentom
33    funkcije sufiks niske zadate prvi argumentom funkcije */
34 int sufiks_niske(char *niska, char *sufiks)
35 {
36     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
46    funkcije 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 argumente, prijavljuje se
64        greska */
65     if (argc < 4) {
66         fprintf(stderr, "Greska: ");
67         fprintf(stderr,
68                 "Nedovoljan broj argumenata komandne linije.\n");
69         fprintf(stderr, "Program se poziva sa\n");
70         fprintf(stderr, "%s ime_dat suf/pref -s/-p\n", argv[0]);
71         exit(EXIT_FAILURE);
72     }
73
74     FILE *in;
75     int br = 0;
76     char rec[MAX_KARAKTERA + 1];
77
78     in = fopen(*(argv + 1), "r");
79     if (in == NULL) {
80         fprintf(stderr, "Greska: ");
81     }
82 }
```



## 2 Pokazivači

```
81     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
      exit(EXIT_FAILURE);
83 }

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

99     fclose(in);
101     exit(EXIT_SUCCESS);
103 }
```

### Rešenje 2.8

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

5  #define MAX 100

7  /* Funkcija izracunava trag matrice */
      int trag(int M[][MAX], int n)
9  {
      int trag = 0, i;
11     for (i = 0; i < n; i++)
          trag += M[i][i];
13     return trag;
      }

15

17 /* Funkcija izracunava euklidsku normu matrice */
      double euklidska_norma(int M[][MAX], int n)
19 {
      double norma = 0.0;
      int i, j;

21     for (i = 0; i < n; i++)
23         for (j = 0; j < n; j++)
            norma += M[i][j] * M[i][j];
25 }
```

```
    return sqrt(norma);
27 }

29 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
int gornja_vandijagonalna_norma(int M[][MAX], int n)
31 {
    int norma = 0;
33     int i, j;

35     for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++)
37             norma += abs(M[i][j]);
    }

39     return norma;
41 }

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

47     printf("Unesite broj vrsta matrice: ");
49     scanf("%d", &n);

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

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

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

69

71     /******
    Ispisuju se elementi matrice koriscenjem pokazivacke sintakse.
    Kod ovako definisane matrice, elementi su uzastopno smesteni u
73     memoriju, kao na traci. To znaci da su svi elementi prve vrste
    redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
75     prve vrste smesten je prvi element druge vrste za kojim slede
    svi elementi te vrste i tako dalje redom.
77
```

## 2 Pokazivači

```
79     for( i = 0; i < n ; i++) {
        for ( j=0 ; j<n ; j++)
            printf("%d ", *(A+i+j));
81     printf("\n");
    }
83     *****/

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

89     printf("Euklidska norma matrice je %.2f.\n",
            euklidska_norma(A, n));
91     printf("Vandijagonalna norma matrice je = %d.\n",
            gornja_vandijagonalna_norma(A, n));
93
95     exit(EXIT_SUCCESS);
}
```

### Rešenje 2.9

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

17     /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
       standardni izlaz */
19     void ispsi_matricu(int m[][MAX], int n)
       {
21         int i, j;
23
24         for (i = 0; i < n; i++) {
25             for (j = 0; j < n; j++)
26                 printf("%d ", m[i][j]);
27             printf("\n");
           }
29     }

   /* 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     /* Prošla je provera jednakosti za sve parove elemenata koji su
42        na istim pozicijama. To znaci da su matrice jednake */
43     return 1;
44 }
45
46 /* Funkcija izracunava zbir dve kvadratne matrice */
47 void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
48 {
49     int i, j;
50
51     for (i = 0; i < n; i++)
52         for (j = 0; j < n; j++)
53             c[i][j] = a[i][j] + b[i][j];
54 }
55
56 /* Funkcija izracunava proizvod dve kvadratne matrice */
57 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
58 {
59     int i, j, k;
60
61     for (i = 0; i < n; i++)
62         for (j = 0; j < n; j++) {
63             /* Mnozi se i-ta vrsta prve sa j-tom kolonom druge matrice */
64             c[i][j] = 0;
65             for (k = 0; k < n; k++)
66                 c[i][j] += a[i][k] * b[k][j];
67         }
68 }
69
70 int main()
71 {
72     /* Matrice ciji se elementi zadaju sa ulaza */
73     int a[MAX][MAX], b[MAX][MAX];
74
75     /* Matrice zbira i proizvoda */
76     int zbir[MAX][MAX], proizvod[MAX][MAX];
77
78     /* Dimenzija kvadratnih matrica */
79     int n;
80
81     printf("Unesite broj vrsta matrica:\n");
82     scanf("%d", &n);
```

## 2 Pokazivači

```
83
84 /* Proverava se da li je doslo do prekoracenja */
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 po vrstama:\n");
92 ucitaj_matricu(a, n);
93 printf("Unesite elemente druge matrice po vrstama:\n");
94 ucitaj_matricu(b, n);
95
96 /* Izracunava se zbir i proizvod matrica */
97 saberi(a, b, zbir, n);
98 pomnozi(a, b, proizvod, n);
99
100 /* Ispisuje se rezultat */
101 if (jednake_matrice(a, b, n) == 1)
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>
#include <stdlib.h>

#define MAX 64

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

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

```
18     return 1;
19 }
20
21 /* Funkcija prepisuje sadrzaj matrice original u matricu kopija */
22 void kopiraj_matricu(int original[][MAX], int n, int kopija[][MAX])
23 {
24     int i, j;
25
26     for (i = 0; i < n; i++)
27         for (j = 0; j < n; j++)
28             kopija[i][j] = original[i][j];
29 }
30
31 /* Funkcija odredjuje reflektivno zatvorenje zadate relacije. Ono
32 je odredjeno matricom koja sadrzi sve elemente polazne matrice
33 dopunjene jedinicama na glavnoj dijagonali */
34 void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
35 {
36     int i;
37
38     /* Kopiraju se vrednosti elemenata pocetne matrice */
39     kopiraj_matricu(m, n, zatvorenje);
40
41     /* Na glavnoj dijagonali se postavljaju jedinice */
42     for (i = 0; i < n; i++)
43         zatvorenje[i][i] = 1;
44 }
45
46 /* Funkcija proverava da li je relacija simetricna. Relacija je
47 simetricna ako za svaki par elemenata vazi: ako je element i u
48 relaciji sa elementom j, onda je i element j u relaciji sa
49 elementom i. Ovakve matrice su simetricne u odnosu na glavnu
50 dijagonalu */
51 int simetricnost(int m[][MAX], int n)
52 {
53     int i, j;
54
55     /* Obilaze se elementi ispod glavne dijagonale matrice i
56 upoređuju se sa njima simetricnim elementima */
57     for (i = 0; i < n; i++)
58         for (j = 0; j < i; j++)
59             if (m[i][j] != m[j][i])
60                 return 0;
61
62     return 1;
63 }
64
65 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono je
66 odredjeno matricom koja sadrzi sve elemente polazne matrice
67 dopunjene tako da matrica postane simetricna u odnosu na glavnu
68 dijagonalu */
69 void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
```

```
70 {
71     int i, j;
72
73     /* Kopiraju se vrednosti elemenata pocetne matrice */
74     kopiraj_matricu(m, n, zatvorenje);
75
76     for (i = 0; i < n; i++)
77         for (j = 0; j < n; j++)
78             if (zatvorenje[i][j] == 1)
79                 zatvorenje[j][i] = 1;
80 }
81
82 /* Funkcija proverava da li je relacija tranzitivna. Relacija je
83    tranzitivna ako ispunjava sledece svojstvo: ako je element i u
84    relaciji sa elementom j i element j u relaciji sa elementom k,
85    onda je i element i u relaciji sa elementom k */
86 int tranzitivnost(int m[][MAX], int n)
87 {
88     int i, j, k;
89
90     for (i = 0; i < n; i++)
91         for (j = 0; j < n; j++)
92             /* Ispituje se da li postoji element koji narusava *
93                tranzitivnost */
94             for (k = 0; k < n; k++)
95                 if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
96                     return 0;
97
98     return 1;
99 }
100
101 /* Funkcija odredjuje reflektivno-tranzitivno zatvorenje zadate
102    relacije koriscenjem Varsalovog algoritma */
103 void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
104 {
105     int i, j, k;
106
107     /* Odredjuje se reflektivno zatvorenje matrice */
108     ref_zatvorenje(m, n, zatvorenje);
109
110     /* Primenom Varsalovog algoritma odredjuje se tranzitivno
111        zatvorenje matrice */
112     for (k = 0; k < n; k++)
113         for (i = 0; i < n; i++)
114             for (j = 0; j < n; j++)
115                 if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
116                     && (zatvorenje[i][j] == 0))
117                     zatvorenje[i][j] = 1;
118 }
119
120 /* Funkcija ispisuje elemente matrice */
```

```
122 void pisi_matricu(int m[][MAX], int n)
123 {
124     int i, j;
125
126     for (i = 0; i < n; i++) {
127         for (j = 0; j < n; j++)
128             printf("%d ", m[i][j]);
129         printf("\n");
130     }
131 }
132
133 int main(int argc, char *argv[])
134 {
135     FILE *ulaz;
136     int m[MAX][MAX];
137     int pomocna[MAX][MAX];
138     int n, i, j;
139
140     /* Proverava se da li korisnik nije uneo trazene argumente */
141     if (argc < 2) {
142         printf("Greska: ");
143         printf("Nedovoljan broj argumenata komandne linije.\n");
144         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
145         exit(EXIT_FAILURE);
146     }
147
148     /* Otvara se datoteka za citanje */
149     ulaz = fopen(argv[1], "r");
150     if (ulaz == NULL) {
151         fprintf(stderr, "Greska: ");
152         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
153         exit(EXIT_FAILURE);
154     }
155
156     /* Ucitava se dimenzija matrice */
157     fscanf(ulaz, "%d", &n);
158
159     /* Proverava se da li je doslo do prekoracenja dimenzije */
160     if (n > MAX || n <= 0) {
161         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
162         fprintf(stderr, "matrice.\n");
163         exit(EXIT_FAILURE);
164     }
165
166     /* Ucitava se element po element matrice */
167     for (i = 0; i < n; i++)
168         for (j = 0; j < n; j++)
169             fscanf(ulaz, "%d", &m[i][j]);
170
171     /* Ispisuje se rezultat */
172     printf("Relacija %s refleksivna.\n",
173           reflektivnost(m, n) == 1 ? "jeste" : "nije");
```



## 2 Pokazivači

```
174 printf("Relacija %s simetricna.\n",
176         simetricnost(m, n) == 1 ? "jeste" : "nije");
178 printf("Relacija %s tranzitivna.\n",
180         tranzitivnost(m, n) == 1 ? "jeste" : "nije");
182 printf("Refleksivno zatvorenje relacije:\n");
184 ref_zatvorenje(m, n, pomocna);
186 pisi_matricu(pomocna, n);
188 printf("Simetricno zatvorenje relacije:\n");
190 sim_zatvorenje(m, n, pomocna);
192 pisi_matricu(pomocna, n);
194 printf("Refleksivno-tranzitivno zatvorenje relacije:\n");
196 ref_tran_zatvorenje(m, n, pomocna);
198 pisi_matricu(pomocna, n);
200 /* Zatvara se datoteka */
202 fclose(ulaz);
204 exit(EXIT_SUCCESS);
206 }
```

### Rešenje 2.11

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

#define MAX 32

/* Funkcija izracunava najveći element na sporednoj dijagonali. Za
   elemente sporedne dijagonale vazi da je zbir indeksa vrste i
   indeksa kolone jednak n-1 */
int max_sporedna_dijagonala(int m[][MAX], int n)
{
    int i;
    int max_na_sporednoj_dijagonali = m[0][n - 1];

    for (i = 1; i < n; i++)
        if (m[i][n - 1 - i] > max_na_sporednoj_dijagonali)
            max_na_sporednoj_dijagonali = m[i][n - 1 - i];

    return max_na_sporednoj_dijagonali;
}

/* Funkcija izracunava indeks kolone najmanjeg elementa */
int indeks_min(int m[][MAX], int n)
{
    int i, j;
```

```
26     int min = m[0][0], indeks_kolone = 0;
27
28     for (i = 0; i < n; i++)
29         for (j = 0; j < n; j++)
30             if (m[i][j] < min) {
31                 min = m[i][j];
32                 indeks_kolone = j;
33             }
34
35     return indeks_kolone;
36 }
37
38 /* Funkcija izracunava indeks vrste najveceg elementa */
39 int indeks_max(int m[][MAX], int n)
40 {
41     int i, j;
42     int max = m[0][0], indeks_vrste = 0;
43
44     for (i = 0; i < n; i++)
45         for (j = 0; j < n; j++)
46             if (m[i][j] > max) {
47                 max = m[i][j];
48                 indeks_vrste = i;
49             }
50     return indeks_vrste;
51 }
52
53 /* Funkcija izracunava broj negativnih elemenata matrice */
54 int broj_negativnih(int m[][MAX], int n)
55 {
56     int i, j;
57     int broj_negativnih = 0;
58
59     for (i = 0; i < n; i++)
60         for (j = 0; j < n; j++)
61             if (m[i][j] < 0)
62                 broj_negativnih++;
63
64     return broj_negativnih;
65 }
66
67 int main(int argc, char *argv[])
68 {
69     int m[MAX][MAX];
70     int n;
71     int i, j;
72
73     /* Proverava se broj argumenata komandne linije */
74     if (argc < 2) {
75         printf("Greska: ");
76         printf("Nedovoljan broj argumenata komandne linije.\n");
77         printf("Program se poziva sa %s br_vrsta_mat.\n", argv[0]);
78     }
```

## 2 Pokazivači

```
    exit(EXIT_FAILURE);
78 }

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

82
84 if (n > MAX || n <= 0) {
    fprintf(stderr, "Greska: Neodgovarajuci broj ");
    fprintf(stderr, "vrsta matrice.\n");
86    exit(EXIT_FAILURE);
    }

88
89 /* Ucitava se matrica */
90 printf("Unesite elemente matrice dimenzije %dx%d:\n", n, n);
91 for (i = 0; i < n; i++)
92     for (j = 0; j < n; j++)
93         scanf("%d", &m[i][j]);
94
95 /* Ispisuju se rezultati izracunavanja */
96 printf("Najveci element sporedne dijagonale je %d.\n",
97        max_sporodna_dijagonala(m, n));
98
99 printf("Indeks kolone sa najmanjim elementom je %d.\n",
100        indeks_min(m, n));
101
102 printf("Indeks vrste sa najvećim elementom je %d.\n",
103        indeks_max(m, n));
104
105 printf("Broj negativnih elemenata matrice je %d.\n",
106        broj_negativnih(m, n));
107
108 exit(EXIT_SUCCESS);
    }
```

### Rešenje 2.12

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 32
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 }
```

```
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 je zadata matrica ortonormirana,
31     odnosno, da li je normirana i ortogonalna. Matrica je normirana
32     ako je proizvod svake vrste matrice sa samom sobom jednak
33     jedinici. Matrica je ortogonalna, ako je proizvod dve bilo koje
34     razlicite vrste matrice jednak nuli */
35 int ortonormirana(int m[][MAX], int n)
36 {
37     int i, j, k;
38     int proizvod;
39
40     /* Ispituje se uslov normiranosti */
41     for (i = 0; i < n; i++) {
42         proizvod = 0;
43         for (j = 0; j < n; j++)
44             proizvod += m[i][j] * m[i][j];
45         if (proizvod != 1)
46             return 0;
47     }
48
49     /* Ispituje se uslov ortogonalnosti */
50     for (i = 0; i < n - 1; i++) {
51         for (j = i + 1; j < n; j++) {
52             proizvod = 0;
53             for (k = 0; k < n; k++)
54                 proizvod += m[i][k] * m[j][k];
55             if (proizvod != 0)
56                 return 0;
57         }
58     }
59
60     /* Ako su oba uslova ispunjena, matrica je ortonormirana */
61     return 1;
62 }
63
64 int main()
65 {
66     int A[MAX][MAX];
67     int n;
```

## 2 Pokazivači

```
69 printf("Unesite broj vrsta matrice: ");
   scanf("%d", &n);

71
   if (n > MAX || n <= 0) {
73     fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
       fprintf(stderr, "matrice.\n");
75     exit(EXIT_FAILURE);
   }

77
   printf("Unesite elemente matrice po vrstama:\n");
79   ucitaj_matricu(A, n);
   printf("Matrica %s ortonormirana.\n",
81         ortonormirana(A, n) ? "je" : "nije");

83
   exit(EXIT_SUCCESS);
}
```

### Rešenje 2.13

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

4 #define MAX_V 10
   #define MAX_K 10

6
   /* Funkcija proverava da li su ispisani svi elementi iz matrice,
8     odnosno da li se narušio prirodan poredak medju granicama */
   int kraj_ispisa(int vrh, int dno, int levo, int desno)
10  {
       return !(vrh <= dno && levo <= desno);
12  }

14  /* Funkcija spiralno ispisuje elemente matrice */
   void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16  {
       int i, j, vrh, dno, levo, desno;

18
       vrh = levo = 0;
20       dno = n - 1;
       desno = m - 1;

22
       while (!kraj_ispisa(vrh, dno, levo, desno)) {
24         for (j = levo; j <= desno; j++)
           printf("%d ", a[vrh][j]);

26
           /* Spusta se prvi red za naredni krug ispisa */
28         vrh++;

30
           if (kraj_ispisa(vrh, dno, levo, desno))
             break;
}
```

```
32     for (i = vrh; i <= dno; i++)
33         printf("%d ", a[i][desno]);
34
35     /* Pomera se desna kolona za naredni krug ispisa blize levom
36        kraju */
37     desno--;
38
39     if (kraj_ispisa(vrh, dno, levo, desno))
40         break;
41
42     /* Ispisuje se donja vrsta */
43     for (j = desno; j >= levo; j--)
44         printf("%d ", a[dno][j]);
45
46     /* Podize se donja vrsta za naredni krug ispisa */
47     dno--;
48
49     if (kraj_ispisa(vrh, dno, levo, desno))
50         break;
51
52     /* Ispisuje se prva kolona */
53     for (i = dno; i >= vrh; i--)
54         printf("%d ", a[i][levo]);
55
56     /* Priprema se leva kolona za naredni krug ispisa */
57     levo++;
58 }
59 putchar('\n');
60 }
61
62 /* Funkcija ucitava matricu */
63 void ucitaj_matricu(int a[][MAX_K], int n, int m)
64 {
65     int i, j;
66
67     for (i = 0; i < n; i++)
68         for (j = 0; j < m; j++)
69             scanf("%d", &a[i][j]);
70 }
71
72 int main()
73 {
74     int a[MAX_V][MAX_K];
75     int m, n;
76
77     printf("Unesite broj vrsta i broj kolona:\n");
78     scanf("%d %d", &n, &m);
79
80     if (n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
81         fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
82         fprintf(stderr, "matrice.\n");
83     }
```

## 2 Pokazivači

---

```
84     exit(EXIT_FAILURE);
85 }
86
87 printf("Unesite elemente matrice po vrstama:\n");
88 ucitaj_matricu(a, n, m);
89
90 printf("Spiralno ispisana matrica: ");
91 ispisi_matricu_spiralno(a, n, m);
92
93 exit(EXIT_SUCCESS);
94 }
```

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

## Rešenje 2.16

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define KORAK 10
5
6 int main()
7 {
8     /* Adresa prvog alociranog bajta */
9     int *a = NULL;
10
11     /* Velicina alocirane memorije */
12     int alocirano;
13
14     /* Broj elemenata niza */
15     int n;
16
17     /* Broj koji se ucitava sa ulaza */
18     int x;
19     int i;
20     int *b = NULL;
21     char realokacija;
22
23     /* Inicijalizacija */
24     alocirano = n = 0;
25
26     printf("Unesite zeljeni nacin realokacije (M ili R):\n");
27     scanf("%c", &realokacija);
28
29     printf("Unesite brojeve, nulu za kraj:\n");
30     scanf("%d", &x);
31
32     while (x != 0) {
33         if (n == alocirano) {
34             alocirano = alocirano + KORAK;
35
36             if (realokacija == 'M') {
37                 /* Vrsi se realokacija memorije sa novom velicinom */
38                 b = (int *) malloc(alocirano * sizeof(int));
39
40                 if (b == NULL) {
41                     fprintf(stderr,
42                         "Greska: Neuspesna alokacija memorije.\n");
43                     free(a);
44                     exit(EXIT_FAILURE);
45                 }
46
47                 /* Svih n elemenata koji pocinju na adresi a prepisuju se
48                 na novu adresu b */
49                 for (i = 0; i < n; i++)
50                     b[i] = a[i];
```



## 2 Pokazivači

```
52     free(a);
54     /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
56        bloka cija je adresa prilikom alokacije zapamcena u
58        promenljivoj b */
59     a = b;
60     } else if (realokacija == 'R') {
61
62         /* Zbog funkcije realloc je neophodno da i u prvoj
64            iteraciji "a" bude inicijalizovano na NULL */
65         a = (int *) realloc(a, alocirano * sizeof(int));
66         if (a == NULL) {
67             fprintf(stderr,
68                    "Greska: Neuspesna realokacija memorije.\n");
69             exit(EXIT_FAILURE);
70         }
71     }
72     a[n++] = x;
73     scanf("%d", &x);
74 }
75 printf("Niz u obrnutom poretku je: ");
76 for (n--; n >= 0; n--)
77     printf("%d ", a[n]);
78 printf("\n");
79
80 /* Oslobadja se dinamicki alocirana memorija */
81 free(a);
82
83 exit(EXIT_SUCCESS);
84 }
```

### Rešenje 2.17

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 1000
6
7 /* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
8    nadovezivanja niski. Adresa kreiranog niza se vraca kao povratna
9    vrednost. */
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) {
```

```

18     fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
19     exit(EXIT_FAILURE);
20 }
21
22 /* Kopiraju se i nadovezuju niske karaktera */
23 strcpy(p, s);
24 strcat(p, t);
25
26 return p;
27 }
28
29 int main()
30 {
31     char *s = NULL;
32     char s1[MAX], s2[MAX];
33
34     printf("Unesite dve niske karaktera:\n");
35     scanf("%s", s1);
36     scanf("%s", s2);
37
38     /* Poziva se funkcija koja nadovezuje niske */
39     s = nadovezi(s1, s2);
40
41     /* Prikazuje se rezultat */
42     printf("Nadovezane niske: %s\n", s);
43
44     /* Oslobadja se memorija alocirana u funkciji nadovezi() */
45     free(s);
46
47     exit(EXIT_SUCCESS);
48 }

```

### 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

```

## 2 Pokazivači

```
19  printf("Unesite broj vrsta i broj kolona:\n ");
    scanf("%d%d", &n, &m);

21  /* Dinamicki se rezervise prostor za niz vrsta matrice */
    A = (double **) malloc(sizeof(double *) * n);

23

25  /* Proverava se da li je uspelo rezervisanje memorije */
    if (A == NULL) {
        fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
        exit(EXIT_FAILURE);
    }

29

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

33

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

43

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

49

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

53

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

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

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

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

65

67  exit(EXIT_SUCCESS);
}
```

## Rešenje 2.19

*matrica.h*

```
1  #ifndef _MATRICA_H_
2  #define _MATRICA_H_ 1
3
4  /* Funkcija dinamički alokira memoriju za matricu dimenzije n x m */
5  int **alociraj_matricu(int n, int m);
6
7  /* Funkcija dinamički alokira memoriju za kvadratnu matricu
8     dimenzije n x n */
9  int **alociraj_kvadratnu_matricu(int n);
10
11 /* Funkcija oslobadja memoriju za matricu sa n vrsta */
12 int **dealociraj_matricu(int **matrica, int n);
13
14 /* Funkcija ucitava vec alociranu matricu dimenzije n x m sa
15     standardnog ulaza */
16 void ucitaj_matricu(int **matrica, int n, int m);
17
18 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n sa
19     standardnog ulaza */
20 void ucitaj_kvadratnu_matricu(int **matrica, int n);
21
22 /* Funkcija ispisuje matricu dimenzije n x m na standardni izlaz */
23 void ispisi_matricu(int **matrica, int n, int m);
24
25 /* Funkcija ispisuje kvadratnu matricu dimenzije n na standardni
26     izlaz */
27 void ispisi_kvadratnu_matricu(int **matrica, int n);
28
29 /* Funkcija ucitava vec alociranu matricu dimenzije n x m iz
30     datoteke f */
31 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m,
32                                 FILE * f);
33
34 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n x n
35     iz datoteke f */
36 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
37                                           FILE * f);
38
39 /* Funkcija upisuje matricu dimenzije n x m u datoteku f */
40 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);
41
42 /* Funkcija upisuje kvadratnu matricu dimenzije n x n u datoteku f */
43 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
44                                         FILE * f);
45
46 #endif
```

## 2 Pokazivači

---

*matrica.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 int **alociraj_matricu(int n, int m)
6 {
7     int **matrica = NULL;
8     int i, j;
9
10    /* Alocira se prostor za niz vrsta matrice */
11    matrica = (int **) malloc(n * sizeof(int *));
12    /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije
13       ce biti NULL, sto mora biti provereno u main funkciji */
14    if (matrica == NULL)
15        return NULL;
16
17    /* Alocira se prostor za svaku vrstu matrice */
18    for (i = 0; i < n; i++) {
19        matrica[i] = (int *) malloc(m * sizeof(int));
20        /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
21           prethodno alocirani resursi, i povratna vrednost je NULL */
22        if (matrica[i] == NULL) {
23            for (j = 0; j < i; j++)
24                free(matrica[j]);
25            free(matrica);
26            return NULL;
27        }
28    }
29    return matrica;
30 }
31
32 int **alociraj_kvadratnu_matricu(int n)
33 {
34     return alociraj_matricu(n, n);
35 }
36
37 int **dealociraj_matricu(int **matrica, int n)
38 {
39     int i;
40     /* Oslobadja se prostor rezervisan za svaku vrstu */
41     for (i = 0; i < n; i++)
42         free(matrica[i]);
43     /* Oslobadja se memorija za niz pokazivaca na vrste */
44     free(matrica);
45
46     /* Matrica postaje prazna, tj. nealocirana */
47     return NULL;
48 }
```

```
50 void ucitaj_matricu(int **matrica, int n, int m)
51 {
52     int i, j;
53     /* Elementi matrice se ucitavaju po vrstama */
54     for (i = 0; i < n; i++)
55         for (j = 0; j < m; j++)
56             scanf("%d", &matrica[i][j]);
57 }
58
59 void ucitaj_kvadratnu_matricu(int **matrica, int n)
60 {
61     ucitaj_matricu(matrica, n, n);
62 }
63
64 void ispisi_matricu(int **matrica, int n, int m)
65 {
66     int i, j;
67     /* Ispis po vrstama */
68     for (i = 0; i < n; i++) {
69         for (j = 0; j < m; j++)
70             printf("%d ", matrica[i][j]);
71         printf("\n");
72     }
73 }
74
75 void ispisi_kvadratnu_matricu(int **matrica, int n)
76 {
77     ispisi_matricu(matrica, n, n);
78 }
79
80 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m,
81                               FILE * f)
82 {
83     int i, j;
84     /* Elementi matrice se ucitavaju po vrstama */
85     for (i = 0; i < n; i++)
86         for (j = 0; j < m; j++)
87             /* Ako je nemoguće učitati sledeći element, povratna vrednost
88              funkcije je 1, kao indikator neuspešnog učitavanja */
89             if (fscanf(f, "%d", &matrica[i][j]) != 1)
90                 return 1;
91
92     /* Uspešno učitana matrica */
93     return 0;
94 }
95
96 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
97                                           FILE * f)
98 {
99     return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
100 }
```

## 2 Pokazivači

```
102 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
103 {
104     int i, j;
105     /* Ispis po vrstama */
106     for (i = 0; i < n; i++) {
107         for (j = 0; j < m; j++)
108             /* Ako je nemoguće ispisati sledeći element, povratna
109              vrednost funkcije je 1, kao indikator neuspešnog ispisa */
110             if (fprintf(f, "%d ", matrica[i][j]) <= 0)
111                 return 1;
112         fprintf(f, "\n");
113     }
114
115     /* Uspešno upisana matrica */
116     return 0;
117 }
118
119 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
120                                       FILE * f)
121 {
122     return upisi_matricu_u_datoteku(matrica, n, n, f);
123 }
```

*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     /* Učitava se broj vrsta i broj kolona matrice */
12     printf("Unesite broj vrsta matrice: ");
13     scanf("%d", &n);
14     printf("Unesite broj kolona matrice: ");
15     scanf("%d", &m);
16
17     /* Provera dimenzija matrice */
18     if (n <= 0 || m <= 0) {
19         fprintf(stderr,
20                "Greska: Broj vrsta i broj kolona ne mogu biti negativni
21                 brojevi.\n");
22         exit(EXIT_FAILURE);
23     }
24
25     /* Rezervise se memorijski prostor za matricu i proverava se da
26        li je memorijski prostor uspešno rezervisan */
```

```

matrica = alociraj_matricu(n, m);
27 if (matrica == NULL) {
    fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
29     exit(EXIT_FAILURE);
}

31 /* Ucitava se matrica sa standardnog ulaza */
33 printf("Unesite elemente matrice po vrstama:\n");
ucitaj_matricu(matrica, n, m);

35 /* Otvara se datoteka za upis matrice */
37 if ((f = fopen("matrica.txt", "w")) == NULL) {
    fprintf(stderr, "Greska: Neuspesno otvaranje datoteke.\n");
39     matrica = dealociraj_matricu(matrica, n);
    exit(EXIT_FAILURE);
41 }

43 /* Upis matrice u datoteku */
45 if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
    fprintf(stderr,
47         "Greska: Neuspesno upisivanje matrice u datoteku.\n");
    matrica = dealociraj_matricu(matrica, n);
    exit(EXIT_FAILURE);
49 }

51 /* Zatvara se datoteka */
fclose(f);

53 /* Oslobadja se memorija koju je zauzimala matrica */
55 matrica = dealociraj_matricu(matrica, n);

57 exit(EXIT_SUCCESS);
}

```

*main\_b.c*

```

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

4
int main(int argc, char **argv)
6 {
    int **matrica = NULL;
8     int n;
    FILE *f;

10
    /* Provera argumenata komandne linije */
12     if (argc != 2) {
        fprintf(stderr, "Greska: Koriscenje programa: %s datoteka\n",
14             argv[0]);
        exit(EXIT_FAILURE);
    }
}

```



## 2 Pokazivači

```
16  }
18  /* Otvara se datoteka za citanje */
19  if ((f = fopen(argv[1], "r")) == NULL) {
20      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke.\n");
21      exit(EXIT_FAILURE);
22  }
24  /* Ucitava se dimenzija matrice */
25  if (fscanf(f, "%d", &n) != 1) {
26      fprintf(stderr, "Greska: Neispravan pocetak datoteke.\n");
27      exit(EXIT_FAILURE);
28  }
30  /* Provera dimenzija matrice */
31  if (n <= 0) {
32      fprintf(stderr, "Greska: Neodgovarajca dimenzija matrice.\n");
33      exit(EXIT_FAILURE);
34  }
36  /* Rezervise se memorijski prostor za matricu i vrsi se provera */
37  matrica = alociraj_kvadratnu_matricu(n);
38  if (matrica == NULL) {
39      fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
40      exit(EXIT_FAILURE);
41  }
42
43  /* Ucitava se matrica iz datoteke */
44  if (ucitaj_kvadratnu_matricu_iz_datoteke(matrica, n, f) != 0) {
45      fprintf(stderr,
46          "Greska: Neuspesno ucitavanje matrice iz datoteke.\n");
47      matrica = dealociraj_matricu(matrica, n);
48      exit(EXIT_FAILURE);
49  }
50
51  /* Zatvara se datoteka */
52  fclose(f);
54  /* Ispis matrice na standardni izlaz */
55  ispisi_kvadratnu_matricu(matrica, n);
56
57  /* Oslobadja se memorija koju je zauzimala matrica */
58  matrica = dealociraj_matricu(matrica, n);
60
61  exit(EXIT_SUCCESS);
62  }
```

### Rešenje 2.20

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

```

3  #include <math.h>
   #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++) {
        for (j = 0; j <= i; j++)
13        printf("%d ", M[i][j]);
        printf("\n");
15    }
   }

17  int main()
19  {
        int m, n;
21    int **matrica = NULL;

23    printf("Unesite broj vrsta i broj kolona:\n ");
        scanf("%d %d", &n, &m);
25

        /* Rezervise se memorija za matricu */
27    matrica = alociraj_matricu(n, m);
        /* Provera alokacije */
29    if (matrica == NULL) {
        fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
31        exit(EXIT_FAILURE);
    }

33    printf("Unesite elemente matrice po vrstama:\n");
35    ucitaj_matricu(matrica, n, m);

37    printf("Elementi ispod glavne dijagonale matrice:\n");
        ispisi_elemente_ispod_dijagonale(matrica, n, m);
39

        /* Oslobadja se memorija */
41    matrica = dealociraj_matricu(matrica, n);

43    exit(EXIT_SUCCESS);
   }

```

### Rešenje 2.22

```

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

```

```
{
8   int i, j;

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

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

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

32     return zbir;
    }

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

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

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

52     for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
54             printf("%.2f ", m[i][j]);
            printf("\n");
56     }
    }
58 }
```

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

64     m = (float **) malloc(n * sizeof(float *));
    if (m == NULL) {
66         fprintf(stderr, "Greska: Neupesna alokacija memorije.\n");
        exit(EXIT_FAILURE);
68     }

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

74         if (m[i] == NULL) {
            fprintf(stderr, "Greska: Neupesna alokacija memorije.\n");
76             for (j = 0; j < i; j++)
                free(m[j]);
            free(m);
78             exit(EXIT_FAILURE);
80         }
    }
82     return m;
}

84 /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom
    dimenzije n x n */
void oslobodi_memoriju(float **m, int n)
88 {
    int i;

90     for (i = 0; i < n; i++)
92         free(m[i]);
    free(m);
94 }

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

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

110
```

## 2 Pokazivači

```
112  /* Otvara se datoteka za citanje */
113  ulaz = fopen(argv[1], "r");
114  if (ulaz == NULL) {
115      fprintf(stderr, "Greska: ");
116      fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
117      exit(EXIT_FAILURE);
118  }
119
120  /* Cita se dimenzija matrice */
121  fscanf(ulaz, "%d", &n);
122
123  /* Rezervise se memorija */
124  a = alociraj_memoriju(n);
125
126  /* Ucitavaju se elementi matrice */
127  ucitaj_matricu(ulaz, a, n);
128
129  float zbir = zbir_ispod_sporodne_dijagonale(a, n);
130
131  /* Poziva se funkcija za transformaciju matrice */
132  izmeni(a, n);
133
134  /* Ispisuju se rezultati */
135  printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
136  printf("je %.2f.\n", zbir);
137
138  printf("Transformisana matrica je:\n");
139  ispisi_matricu(a, n);
140
141  /* Oslobadja se memorija */
142  oslobodi_memoriju(a, n);
143
144  /* Zatvara se datoteka */
145  fclose(ulaz);
146
147  exit(EXIT_SUCCESS);
148 }
```

### Rešenje 2.27

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <string.h>
5
6  /* Funkcija tabela() prihvata granice intervala a i b, broj
7   ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
8   funkciju koja prihvata double argument i vraca double vrednost.
9   Za tako datu funkciju ispisuju se njene vrednosti u intervalu
10  [a,b] u n ekvidistantnih tacaka intervala */
11 void tabela(double a, double b, int n, double (*fp) (double))
```

```
{
13  int i;
    double x;
15
    printf("-----\n");
17  for (i = 0; i < n; i++) {
        x = a + i * (b - a) / (n - 1);
19    printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
    }
21  printf("-----\n");
}

double sqr(double a)
23 {
25     return a * a;
27 }

int main(int argc, char *argv[])
29 {
31     double a, b;
    int n;
33
    char ime_funkcije[6];
35
    /* Pokazivac na funkciju koja ima jedan argument tipa double i
37     povratnu vrednost istog tipa */
    double (*fp) (double);
39
    /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
41     greska */
    if (argc < 2) {
43         fprintf(stderr, "Greska: ");
        fprintf(stderr,
45             "Nedovoljan broj argumenata komandne linije.\n");
        fprintf(stderr,
47             "Program se poziva sa %s ime_funkcije iz math.h.\n",
                argv[0]);
49         exit(EXIT_FAILURE);
    }
51
    /* Niska ime_funkcije sadrzi ime trazene funkcije koja je
53     navedena u komandnoj liniji */
    strcpy(ime_funkcije, argv[1]);
55
    /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
57     if (strcmp(ime_funkcije, "sin") == 0)
        fp = &sin;
59     else if (strcmp(ime_funkcije, "cos") == 0)
        fp = &cos;
61     else if (strcmp(ime_funkcije, "tan") == 0)
        fp = &tan;
63     else if (strcmp(ime_funkcije, "atan") == 0)
```

```
    fp = &atan;
65 else if (strcmp(ime_funkcije, "acos") == 0)
    fp = &acos;
67 else if (strcmp(ime_funkcije, "asin") == 0)
    fp = &asin;
69 else if (strcmp(ime_funkcije, "exp") == 0)
    fp = &exp;
71 else if (strcmp(ime_funkcije, "log") == 0)
    fp = &log;
73 else if (strcmp(ime_funkcije, "log10") == 0)
    fp = &log10;
75 else if (strcmp(ime_funkcije, "sqrt") == 0)
    fp = &sqrt;
77 else if (strcmp(ime_funkcije, "floor") == 0)
    fp = &floor;
79 else if (strcmp(ime_funkcije, "ceil") == 0)
    fp = &ceil;
81 else if (strcmp(ime_funkcije, "sqr") == 0)
    fp = &sqr;
83 else {
    fprintf(stderr, "Greska");
85     fprintf(stderr,
        "Program jos uvek ne podrzava trazenu funkciju!\n");
87     exit(EXIT_FAILURE);
    }
89
    printf("Unesite krajeve intervala:\n");
91     scanf("%lf %lf", &a, &b);

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

97     /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
        uneti broj veci od 2 */
99     if (n < 2) {
        fprintf(stderr, "Greska: Broj tacaka mreze mora biti bar 2!\n");
101         exit(EXIT_FAILURE);
    }
103

    /* Ispisuje se ime funkcije */
105     printf("    x %10s(x)\n", ime_funkcije);

107     /* Funkciji tabela() se prosledjuje funkcija koja je zadata kao
        argument komandne linije */
109     tabela(a, b, n, fp);

111     exit(EXIT_SUCCESS);
}
```

## 3

# Algoritmi pretrage i sortiranja

## 3.1 Algoritmi pretrage

**Zadatak 3.1** Napisati iterativne funkcije za pretragu 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 prosleđuje prirodan broj  $n$  koji nije veći od 1000000 i broj  $x$  kao drugi argument komandne linije. Potrebna vremena za izvršavanje ovih funkcija dopisati u datoteku `vremena.txt`.



### 3 Algoritmi pretrage i sortiranja

---

#### Test 1

```
POKRETANJE: ./a.out 1000000 23542
```

```
VREMENA.TXT
```

```
IZLAZ:
Linearna pretraga:
Element nije u nizu
Binarna pretraga:
Element nije u nizu
Interpolaciona pretraga:
Element nije u nizu
```

```
Dimenzija niza: 1000000
Linearna: 3615091 ns
Binarna: 1536 ns
Interpolaciona: 558 ns
```

#### Test 2

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

```
VREMENA.TXT
```

```
IZLAZ:
Linearna pretraga:
Element nije u nizu
Binarna pretraga:
Element nije u nizu
Interpolaciona pretraga:
Element nije u nizu
```

```
Dimenzija niza: 1000000
Linearna: 3615091 ns
Binarna: 1536 ns
Interpolaciona: 558 ns
```

```
Dimenzija niza: 100000
Linearna: 360803 ns
Binarna: 1187 ns
Interpolaciona: 628 ns
```

[Rešenje 3.1]

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

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite trazeni broj: 11
Unesite sortiran niz elemenata:
2 5 6 8 10 11 23
Linearna pretraga
Pozicija elementa je 5.
Binarna pretraga
Pozicija elementa je 5.
Interpolaciona pretraga
Pozicija elementa je 5.
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite trazeni broj: 14
Unesite sortiran niz elemenata:
10 32 35 43 66 89 100
Linearna pretraga
Element se ne nalazi u nizu.
Binarna pretraga
Element se ne nalazi u nizu.
Interpolaciona pretraga
Element se ne nalazi u nizu.
```

[Rešenje 3.2]

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za

### 3.1 Algoritmi pretrage

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 standardni izlaz. U slučaju više studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti `-indeks` ili `-prezime`. U slučaju neuspešnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složeno. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

#### Primer 1

```
POKRETANJE: ./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

```
POKRETANJE: ./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 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 tačkaka. 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 tačkaka u datoteci veći od 0 i ne veći od 1024.

#### Test 1

```
POKRETANJE: ./a.out dat.txt -x
DAT.TXT
12 53
2.342 34.1
-0.3 23
-1 23.1
123.5 756.12
IZLAZ:
-0.3 23
```

#### Test 2

```
POKRETANJE: ./a.out dat.txt
DAT.TXT
12 53
2.342 34.1
-0.3 23
-1 2.1
123.5 756.12
IZLAZ:
-1 2.1
```

[Rešenje 3.5]

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

#### Test 1

```
IZLAZ:
1.57031
```

[Rešenje 3.6]

**Zadatak 3.7** Napisati funkciju koja metodom polovljenja intervala određuje nulu izabrane funkcije na proizvoljnom 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).*

#### Primer 1

```
POKRETANJE: ./a.out cos
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 0 2
Unesite preciznost: 0.001
1.57031
```

#### Primer 2

```
POKRETANJE: ./a.out sin
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 1 5
Unesite preciznost: 0.00001
3.1416
```

#### Primer 3

```

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

#### Primer 4

```

POKRETANJE: ./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]

### 3 Algoritmi pretrage i sortiranja

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati pozitivan ceo broj, a na standardni izlaz ispisati njegov logaritam.

```
Test 1          Test 2          Test 3
|| ULAZ:        || ULAZ:        || ULAZ:
|| 4            || 17         || 1031
||             ||           ||
|| IZLAZ:       || IZLAZ:       || IZLAZ:
|| 2 2         || 4 4         || 10 10
```

[Rešenje 3.10]

\* **Zadatak 3.11** U prvom kvadrantu dato je  $1 \leq N \leq 10000$  duži svojim koordinatama. Duži mogu da se seku, preklapaju, itd. Napisati program koji pronalazi najmanji ugao  $0 \leq \alpha \leq 90^\circ$ , na dve decimale, takav da je suma dužina duži sa obe strane polupoluprave iz koordinatnog početka pod uglom  $\alpha$  jednak. Neke duži bivaju presečene, a neke ne. Program prvo učitava broj  $N$ , a zatim i same koordinate temena duži. UPUTSTVO: *Vršiti binarnu pretragu intervala  $[0, 90^\circ]$ .*

```
Primer 1          Primer 2          Primer 3
|| INTERAKCIJA SA PROGRAMOM: || INTERAKCIJA SA PROGRAMOM: || INTERAKCIJA SA PROGRAMOM:
|| Unesite broj tacaka: 2     || Unesite broj tacaka: 2     || Unesite broj tacaka: 3
|| Unesite koordinate tacaka: || Unesite koordinate tacaka: || Unesite koordinate tacaka:
|| 2 0 2 1                   || 1 0 1 1                   || 1 0 1 1
|| 1 2 2 2                   || 0 1 1 1                   || 2 0 2 1
|| 26.57                     || 45                         || 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  $n$ .

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

[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 Algoritmi pretrage i sortiranja

---

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite prvu nisku anagram
Unesite drugu nisku rangana
jesu
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite prvu nisku anagram
Unesite drugu nisku anagrmm
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. Pretpostaviti da u niz neće biti uneto više od

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

*Primer 1*

```

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

*Primer 2*

```

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

*Primer 3*

```

INTERAKCIJA SA PROGRAMOM:
Unesite trazeni 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 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. 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 dve datoteke 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`. Pretpostaviti da ime studenta nije duže od 10, a prezime od 15 karaktera.



### 3 Algoritmi pretrage i sortiranja

---

#### Test 1

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

PRVI-DEO.TXT                                GEO-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
DRUGI-DEO.TXT                                Nenad Brankovic
Aleksandra Cvetic                            Ognjen Peric
Bojan Golubovic                             Sofija Filipovic
Dragan Markovic                             Vladimir Savic
Filip Dukic                                  Uros Milic
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

```
POKRETANJE: ./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

```
POKRETANJE: ./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]

**Zadatak 3.20** Napisati program koji učitava imena i prezimena građana 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 u će u datoteci biti najviše 1000 građana, da je za ime, odnosno prezime građana dovoljno 15 karaktera, 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 GREŠKE:
Neuspesno otvaranje
datoteke za citanje
```

[Rešenje 3.20]

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

*Test 1*

```
POKRETANJE: ./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

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

**Zadatak 3.22** Napisati funkciju koja sortira niz niski po broju suglasnika u niski. Ukoliko reči imaju isti broj suglasnika, sortirati ih po dužini niske rastuće, a ukoliko su i dužine jednake onda leksikografski rastuće. Napisati program koji testira ovu funkciju za niske koje se zadaju u datoteci `niske.txt`. Pretpostaviti da u nizu nema više od 128 elemenata, kao i da svaka niska sadrži najviše 31 karakter.

#### Test 1

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

[Rešenje 3.22]

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

## Primer 1

```

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

INTERAKCIJA SA PROGRAMOM:
Asortiman:
KOD Naziv artikla Ime proizvođjaca Cena
23 MedenoSrce Pionir 150.00
1001 Keks Jaffa 120.00
2145 Pardon Marbo 70.00
2530 Napolitanke Bambi 230.00
-----
- Za kraj za kraj rada kase, pritisnite CTRL+D!
- Za nov racun unesite kod artikla:

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

UKUPNO: 270.00 dinara.

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

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

UKUPNO: 230.00 dinara.

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

Kraj rada kase!

```

[Rešenje 3.23]

**Zadatak 3.24** Napisati program koji iz datoteke `aktivnost.txt` čita podatke o aktivnostima studenata na praktikumima i upisuje tri spiska redom u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt`. Na prvom su studenti sortirani leksi-kografski 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 sor-

### 3 Algoritmi pretrage i sortiranja

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

#### Test 1

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

DAT1.TXT
Studenti sortirani po imenu
leksikografski rastuce:
Aleksandra Cvetic 4 6
Andrija Petrovic 2 5
Anja Ilic 3 1
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Lazar Micic 1 3
Marija Stankovic 1 3
Nenad Brankovic 2 4
Ognjen Peric 1 2
Uros Milic 2 5

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

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

[Rešenje 3.24]

**Zadatak 3.25** U datoteci `pesme.txt` nalaze se informacije o gledanosti pesama na Youtube-u. Svaki 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.

## 3.2 Algoritmi sortiranja

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

```
Test 1
POKRETANJE: ./a.out
PESME.TXT
5
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321
Mika - Kisa, 5341
IZLAZ:
Jelena - Sunce, 92321
Mika - Kisa, 5341
Ana - Nebo, 2342
Pera - Ptice, 327
Laza - Oblaci, 29

Test 2
POKRETANJE: ./a.out -i
PESME.TXT
5
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321
Mika - Kisa, 5341
IZLAZ:
Ana - Nebo, 2342
Jelena - Sunce, 92321
Laza - Oblaci, 29
Mika - Kisa, 5341
Pera - Ptice, 327

Test 3
POKRETANJE: ./a.out -n
PESME.TXT
5
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321
Mika - Kisa, 5341
IZLAZ:
Mika - Kisa, 5341
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321
```

[Rešenje 3.25]

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

### Primer 1

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

\* **Zadatak 3.27** 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, slika 3.1 po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze najmanja, najveća, itd... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene. Napisati program koji u najviše  $2n - 3$

### 3 Algoritmi pretrage i sortiranja

---

3	5	2	1
4	4	1_	2
5_	3	3	3
1	1	4	4
2	2_	5	5

Slika 3.1: Zadatak 3.27

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 vrsti. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad 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]

**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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad 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 standardni izlaz za greške. Niz struktura ima manje od 100 elemenata i uređen je u rastućem leksikografskom poretku po prezimenima. Pretraživanje niza vršiti bibliotečkom funkcijom `bsearch`. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

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

#### Test 1

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

#### Test 2

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

#### Test 3

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

**Zadatak 3.31** Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova i mogućnost zadavanja različitih kriterijuma



### 3 Algoritmi pretrage i sortiranja

---

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 standardni izlaz 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 standardni izlaz.

#### 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:

- leksikografski,
- po dužini.

Niske se učitavaju iz datoteke `niske.txt`. Pretpostaviti da datoteka ne sadrži više od 1000 niski kao i da je svaka niska dužine najviše 30 karaktera. Program

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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 standardni izlaz.

#### Primer 1

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

INTERAKCIJA SA PROGRAMOM:
Leksikografski sortirane niske:
aleksandar ana andjela ljubica matej milica milos nikola petar
Uneti trazenu nisku: matej
Niska "matej" je pronadjena u nizu na poziciji 4
Niske sortirane po duzini:
ana matej milos petar milica nikola andjela ljubica aleksandar
Niska "matej" je pronadjena u nizu na poziciji 1
```

[Rešenje 3.33]

**Zadatak 3.34** Uraditi 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. Pretpostaviti da neće biti više od 500 studenata i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

### 3 Algoritmi pretrage i sortiranja

---

#### Primer 1

```
POKRETANJE: ./a.out kolokvijum.txt
```

```
ULAZNA DATOTEKA (KOLOKVIJUM.TXT):
```

```
Aleksandra Cvetic 15
Bojan Golubovic 30
Dragan Markovic 25
Filip Dukic 20
Ivana Stankovic 25
Marija Stankovic 15
Ognjen Peric 20
Uros Milic 10
Andrija Petrovic 0
Anja Ilic 5
Ivana Markovic 5
Lazar Micic 20
Nenad Brankovic 15
```

```
INTERAKCIJA SA PROGRAMOM:
```

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

[Rešenje 3.35]

**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  $S$  bibliotekom 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
POKRETANJE: ./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
POKRETANJE: ./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
POKRETANJE: ./a.out datoteka.txt
DATOTEKA.TXT
1.1.2013.
13.12.2016.
11.11.2011.
3.5.2015.
5.2.2009.
INTERAKCIJA SA PROGRAMOM:
Unesite sledeci datum: 13.12.2016.
postoji
Unesite sledeci datum: 10.5.2015.
ne postoji
Unesite sledeci 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 vrse pretragu. Ukoliko se trazeni element
10    pronadje u nizu, one vracaju indeks pozicije na kojoj je element
11    pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
12    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.
17    Pretraga se vrsi 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     }
48     return levi;
49 }
```

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

```

### 3 Algoritmi pretrage i sortiranja

```
99     else if (x > a[srednji])
100         levi = srednji + 1;
101     else
102         /* Ako je element sa indeksom srednji jednak trazenom, onda
103            se pretraga zavrшава na poziciji srednji */
104         return srednji;
105     }
106     /* U slucaju neuspesne pretrage vraća se -1 */
107     return -1;
108 }
109
110 int main(int argc, char **argv)
111 {
112     int a[MAX];
113     int n, i, x;
114     struct timespec vreme1, vreme2, vreme3, vreme4, vreme5, vreme6;
115     FILE *f;
116     /* Provera argumenata komandne linije */
117     if (argc != 3) {
118         fprintf(stderr,
119                "Greska: Program se poziva sa %s dim_niza broj\n",
120                argv[0]);
121         exit(EXIT_FAILURE);
122     }
123
124     /* Dimenzija niza */
125     n = atoi(argv[1]);
126     if (n > MAX || n <= 0) {
127         fprintf(stderr, "Greska: Dimenzija niza neodgovarajuca\n");
128         exit(EXIT_FAILURE);
129     }
130
131     /* Broj koji se trazi */
132     x = atoi(argv[2]);
133     /* Elementi niza se generisu slucajno, tako da je svaki sledeci
134        veci od prethodnog. Funkcija srandom() inicijalizuje pocetnu
135        vrednost sa kojom se kreće u izracunavanje sekvence
136        pseudo-slucajnih brojeva. Kako generisani niz ne bi uvek bio
137        isti, ova vrednost se postavlja na tekuće vreme u sekundama od
138        Nove godine 1970, tako da je za svako sledeće pokretanje
139        programa (u vremenskim intervalima vecim od jedne sekunde) ove
140        vrednost drugacija. random()%100 vraća brojeve izmedju 0 i 99 */
141     srandom(time(NULL));
142     for (i = 0; i < n; i++)
143         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
144     /* Linearna pretraga */
145     printf("Linearna pretraga:\n");
146     /* Vreme proteklo od Nove godine 1970 */
147     clock_gettime(CLOCK_REALTIME, &vreme1);
148     i = linearna_pretraga(a, n, x);
149     /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
150        linearnu pretragu */
```

```

151 clock_gettime(CLOCK_REALTIME, &vreme2);
    if (i == -1)
153     printf("Element nije u nizu\n");
    else
155     printf("Element je u nizu na poziciji %d\n", i);
    /* Binarna pretraga */
157     printf("Binarna pretraga:\n");
    clock_gettime(CLOCK_REALTIME, &vreme3);
159     i = binarna_pretraga(a, n, x);
    clock_gettime(CLOCK_REALTIME, &vreme4);
161     if (i == -1)
        printf("Element nije u nizu\n");
163     else
        printf("Element je u nizu na poziciji %d\n", i);
    /* Interpolaciona pretraga */
165     printf("Interpolaciona pretraga:\n");
    clock_gettime(CLOCK_REALTIME, &vreme5);
167     i = interpolaciona_pretraga(a, n, x);
    clock_gettime(CLOCK_REALTIME, &vreme6);
169     if (i == -1)
        printf("Element nije u nizu\n");
171     else
        printf("Element je u nizu na poziciji %d\n", i);
    /* Podaci o izvršavanju programa bivaju upisani u log */
175     if ((f = fopen("vremena.txt", "a")) == NULL) {
        fprintf(stderr, "Greska: Neuspesno otvaranje log datoteke.\n");
177         exit(EXIT_FAILURE);
    }
179
    fprintf(f, "Dimenzija niza: %d\n", n);
181     fprintf(f, "\tLinearna: %10ld ns\n",
        (vreme2.tv_sec - vreme1.tv_sec) * 1000000000 +
183         vreme2.tv_nsec - vreme1.tv_nsec);
    fprintf(f, "\tBinarna: %19ld ns\n",
185         (vreme4.tv_sec - vreme3.tv_sec) * 1000000000 +
        vreme4.tv_nsec - vreme3.tv_nsec);
187     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
        (vreme6.tv_sec - vreme5.tv_sec) * 1000000000 +
189         vreme6.tv_nsec - vreme5.tv_nsec);
    /* Zatvara se datoteka */
191     fclose(f);
193     exit(EXIT_SUCCESS);
}

```

### Rešenje 3.2

```

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

4 #define MAX 1024

```



### 3 Algoritmi pretrage i sortiranja

---

```
6  /* Rekurzivna linearna pretraga od pocetka niza */
7  int linearna_pretraga_r1(int a[], int n, int x)
8  {
9      int tmp;
10     /* Izlaz iz rekurzije */
11     if (n <= 0)
12         return -1;
13     /* Ako je prvi element trazeni */
14     if (a[0] == x)
15         return 0;
16     /* Pretraga ostatka niza */
17     tmp = linearna_pretraga_r1(a + 1, n - 1, x);
18     return tmp < 0 ? tmp : tmp + 1;
19 }
20
21 /* Rekurzivna linearna pretraga od kraja niza */
22 int linearna_pretraga_r2(int a[], int n, int x)
23 {
24     /* Izlaz iz rekurzije */
25     if (n <= 0)
26         return -1;
27     /* Ako je poslednji element trazeni */
28     if (a[n - 1] == x)
29         return n - 1;
30     /* Pretraga ostatka niza */
31     return linearna_pretraga_r2(a, n - 1, x);
32 }
33
34 /* Rekurzivna binarna pretraga */
35 int binarna_pretraga_r(int a[], int l, int d, int x)
36 {
37     int srednji;
38     if (l > d)
39         return -1;
40     /* Sredisnja pozicija na kojoj se trazi vrednost x */
41     srednji = (l + d) / 2;
42     /* Ako je element na sredisnjoj poziciji trazeni */
43     if (a[srednji] == x)
44         return srednji;
45     /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
46        pretrazuje se desna polovina niza */
47     if (a[srednji] < x)
48         return binarna_pretraga_r(a, srednji + 1, d, x);
49     /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
50        pretrazuje se leva polovina niza */
51     else
52         return binarna_pretraga_r(a, l, srednji - 1, x);
53 }
54
55 /* Rekurzivna interpolaciona pretraga */
56 int interpolaciona_pretraga_r(int a[], int l, int d, int x)
```

```
{
58  int p;
   /* Ako je trazeni element manji od prvog ili veci od poslednjeg */
60  if (x < a[l] || x > a[d])
       return -1;
62  /* Ako je ostao jedan element u delu niza koji se pretrazuje */
   if (a[d] == a[l])
64     return 1;
   /* Pozicija na kojoj se trazi vrednost x */
66  p = 1 + (d - l) * (x - a[l]) / (a[d] - a[l]);
   if (a[p] == x)
68     return p;
   /* Pretraga sufiksa niza */
70  if (a[p] < x)
       return interpolaciona_pretraga_r(a, p + 1, d, x);
72  /* Pretraga prefiksa niza */
   else
74     return interpolaciona_pretraga_r(a, l, p - 1, x);
}

76  int main()
78  {
   int a[MAX], x, i, indeks;

80
   /* Ucitava se trazeni broj */
82  printf("Unesite trazeni broj: ");
   scanf("%d", &x);

84
   /* Ucitavaju se elementi niza sve do kraja ulaza - ocekuje se da
86  korisnik pritisne CTRL+D za naznaku kraja */
   i = 0;
88  printf("Unesite sortiran niz elemenata: ");
   while (i < MAX && scanf("%d", &a[i]) == 1) {
90     if (i > 0 && a[i] < a[i - 1]) {
           fprintf(stderr, "Greska: Elementi moraju biti uneseni ");
92         fprintf(stderr, "u neopadajucem poretku\n");
           exit(EXIT_FAILURE);
94     }
       i++;
96 }

98  /* Rezultati linearnih pretraga */
   printf("Linearna pretraga\n");
100  indeks = linearna_pretraga_r1(a, i, x);
   if (indeks == -1)
102     printf("Element se ne nalazi u nizu.\n");
   else
104     printf("Pozicija elementa je %d.\n", indeks);
   indeks = linearna_pretraga_r2(a, i, x);
106  if (indeks == -1)
       printf("Element se ne nalazi u nizu.\n");
108  else
```

### 3 Algoritmi pretrage i sortiranja

```
    printf("Pozicija elementa je %d.\n", indeks);
110
    /* Rezultati binarna pretrage */
112    printf("Binarna pretraga\n");
    indeks = binarna_pretraga_r(a, 0, i - 1, x);
114    if (indeks == -1)
        printf("Element se ne nalazi u nizu.\n");
116    else
        printf("Pozicija elementa je %d.\n", indeks);
118
    /* Rezultati interpolacione pretrage */
120    printf("Interpolaciona pretraga\n");
    indeks = interpolaciona_pretraga_r(a, 0, i - 1, x);
122    if (indeks == -1)
        printf("Element se ne nalazi u nizu.\n");
124    else
        printf("Pozicija elementa je %d.\n", indeks);
126
    exit(EXIT_SUCCESS);
128 }
```

### Rešenje 3.3

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

8 /* 0 svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
10 typedef struct {
    /* Indeks mora biti tipa long jer su podaci u datoteci preveliki
12     za int, npr. 20140123 */
    long indeks;
14     char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
16 } Student;

18 /* Ucitani niz studenata ce biti sortiran rastuce prema indeksu, jer
   su studenti u datoteci vec sortirani. Iz tog razloga pretraga po
20     indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
   jer nema garancije da postoji uredjenje po prezimenu. */
22
24 /* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
   sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
   ako element nije pronadjen. */
26 int binarna_pretraga(Student a[], int n, long x)
{
28     int levi = 0;
```

```
30 int desni = n - 1;
31 int srednji;
32 /* Dokle god je indeks levi levo od indeksa desni */
33 while (levi <= desni) {
34     /* Racuna se srednja pozicija */
35     srednji = (levi + desni) / 2;
36     /* Ako je indeks studenta na toj poziciji veci od trazenog,
37     tada se trazeni indeks mora nalaziti u levoj polovini niza */
38     if (x < a[srednji].indeks)
39         desni = srednji - 1;
40     /* Ako je pak manji od trazenog, 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 trazenom indeksu x, tada je pronadjen
46         student sa trazenom indeksom na poziciji srednji */
47         return srednji;
48 }
49 /* Ako nije pronadjen, vraca se -1 */
50 return -1;
51 }
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         /* Poredi se prezime i-tog studenta i poslatog x */
58         if (strcmp(a[i].prezime, x) == 0)
59             return i;
60     return -1;
61 }
62
63 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
64 prosledjuju kao argumenti komandne linije */
65 int main(int argc, char *argv[])
66 {
67     Student dosije[MAX_STUDENATA];
68     FILE *fin = NULL;
69     int i;
70     int br_studenata = 0;
71     long trazeni_indeks = 0;
72     char trazeno_prezime[MAX_DUZINA];
73     int bin_pretraga;
74
75     /* Provera da li je korisnik prilikom poziva programa prosledio
76     ime datoteke sa informacijama o studentima i opciju pretrage */
77     if (argc != 3) {
78         fprintf(stderr,
79             "Greska: Program se poziva sa %s ime_datoteke opcija\n",
80             argv[0]);
```

### 3 Algoritmi pretrage i sortiranja

```
    exit(EXIT_FAILURE);
82 }

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

94 /* Otvara se datoteka */
96 fin = fopen(argv[1], "r");
if (fin == NULL) {
98     fprintf(stderr,
                "Greska: Neuspesno otvaranje datoteke %s za citanje\n",
100     argv[1]);
    exit(EXIT_FAILURE);
102 }

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

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

120 /* Pretraga po indeksu */
if (bin_pretraga) {
122     /* Unos indeksa koji se binarno trazi u nizu */
    printf("Unesite indeks studenta cije informacije zelite: ");
124     scanf("%ld", &trazen_indeks);
    i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
126     /* Rezultat binarne pretrage */
    if (i == -1)
128         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
130         printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132 }
}
```

```

134  /* Pretraga po prezimenu */
135  else {
136      /* Unos prezimena koje se linearno trazi u nizu */
137      printf("Unesite prezime studenta cije informacije zelite: ");
138      scanf("%s", trazeno_prezime);
139      i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
140      /* Rezultat linearne pretrage */
141      if (i == -1)
142          printf("Ne postoji student sa prezimenom %s\n",
143                trazeno_prezime);
144      else
145          printf("Indeks: %ld, Ime i prezime: %s %s\n",
146                dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
147  }
148  exit(EXIT_SUCCESS);
149  }

```

### 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     /* Racuna se pozicija 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,
31        jer je poznato da je niz sortiran po indeksu u rastucem

```

### 3 Algoritmi pretrage i sortiranja

```
        poretku. */
33     if (x < a[srednji].indeks)
        return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
35     /* Inace ga treba traziti u desnoj polovini */
        else
37         return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
    }
39
40 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
41 {
42     /* Ako je niz prazan, vraca se -1 */
43     if (n == 0)
44         return -1;
45     /* Kako se trazi prvi student sa traženim prezimenom, pocinje se
        sa prvim studentom u nizu. */
47     if (strcmp(a[0].prezime, x) == 0)
48         return 0;
49     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
50     return i >= 0 ? 1 + i : -1;
51 }
52
53 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
54 {
55     /* Ako je niz prazan, vraca se -1 */
56     if (n == 0)
57         return -1;
58     /* Ako se trazi poslednji student sa traženim prezimenom, pocinje
        se sa poslednjim studentom u nizu. */
59     if (strcmp(a[n - 1].prezime, x) == 0)
60         return n - 1;
61     return linearna_pretraga_rekurzivna(a, n - 1, x);
62 }
63
64 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
        prosledjuju kao argumenti komandne linije */
65 int main(int argc, char *argv[])
66 {
67     Student dosije[MAX_STUDENATA];
68     FILE *fin = NULL;
69     int i;
70     int br_studenata = 0;
71     long trazen_indeks = 0;
72     char trazeno_prezime[MAX_DUZINA];
73     int bin_pretraga;
74
75     /* Provera da li je korisnik prilikom poziva programa prosledio
        ime datoteke sa informacijama o studentima i opciju pretrage */
76     if (argc != 3) {
77         fprintf(stderr,
78             "Greska: Program se poziva sa %s ime_datoteke opcija\n",
79             argv[0]);
80         exit(EXIT_FAILURE);
81     }
82 }
```

```

}
85
/* Provera prosledjene opcije */
87 if (strcmp(argv[2], "-indeks") == 0)
    bin_pretraga = 1;
89 else if (strcmp(argv[2], "-prezime") == 0)
    bin_pretraga = 0;
91 else {
    fprintf(stderr,
93         "Greska: Opcija mora biti -indeks ili -prezime\n");
    exit(EXIT_FAILURE);
95 }

/* Otvara se datoteka */
97 fin = fopen(argv[1], "r");
99 if (fin == NULL) {
    fprintf(stderr,
101         "Greska: Neuspesno otvaranje datoteke %s za citanje\n",
            argv[1]);
103    exit(EXIT_FAILURE);
}

/* Cita se sve dok postoji red sa informacijama o studentu */
105
107 i = 0;
while (1) {
109     if (i == MAX_STUDENATA)
        break;
111     if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
113         dosije[i].prezime) != 3)
        break;
115     i++;
}
117 br_studenata = i;

119 /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
fclose(fin);
121

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

129    /* Rezultat binarne pretrage */
    if (i == -1)
131        printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
133        printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135 }

```



### 3 Algoritmi pretrage i sortiranja

```
/* Pretraga po prezimenu */
137 else {
    /* Unos prezimena koje se linearno trazi u nizu */
139 printf("Unesite prezime studenta cijje informacije zelite: ");
    scanf("%s", trazeno_prezime);
141 i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
                                     trazeno_prezime);
143 /* Rezultat linearne pretrage */
    if (i == -1)
145 printf("Ne postoji student sa prezimenom %s\n",
          trazeno_prezime);
    else
147 printf("Indeks: %ld, Ime i prezime: %s %s\n",
          dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
149 }
151 exit(EXIT_SUCCESS);
153 }
```

#### Rešenje 3.5

```
1 #include <stdio.h>
  #include <string.h>
3 #include <math.h>
  #include <stdlib.h>
5
  #define MAX 1024
7
  /* Struktura koja opisuje tacku u ravni */
9 typedef struct Tacka {
    float x;
11 float y;
} Tacka;
13
  /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
15 pocetka (0,0) */
float rastojanje(Tacka A)
17 {
    return sqrt(A.x * A.x + A.y * A.y);
19 }
21
  /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u
   nizu zadatih tacaka t dimenzije n */
23 Tacka najbliza_koordinatnom(Tacka t[], int n)
  {
25 Tacka najbliza;
    int i;
27 najbliza = t[0];
    for (i = 1; i < n; i++) {
29 if (rastojanje(t[i]) < rastojanje(najbliza)) {
        najbliza = t[i];
    }
  }
```

```
31     }
32   }
33   return najbliza;
34 }
35
36 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
37    tacaka t dimenzije n */
38 Tacka najbliza_x_osi(Tacka t[], int n)
39 {
40     Tacka najbliza;
41     int i;
42     najbliza = t[0];
43     for (i = 1; i < n; i++) {
44         if (fabs(t[i].x) < fabs(najbliza.x)) {
45             najbliza = t[i];
46         }
47     }
48     return najbliza;
49 }
50
51 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih
52    tacaka t dimenzije n */
53 Tacka najbliza_y_osi(Tacka t[], int n)
54 {
55     Tacka najbliza;
56     int i;
57     najbliza = t[0];
58     for (i = 1; i < n; i++) {
59         if (fabs(t[i].y) < fabs(najbliza.y)) {
60             najbliza = t[i];
61         }
62     }
63     return najbliza;
64 }
65
66 int main(int argc, char *argv[])
67 {
68     FILE *ulaz;
69     Tacka tacke[MAX];
70     Tacka najbliza;
71     int i, n;
72
73     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
74        ime datoteke sa tackama */
75     if (argc < 2) {
76         fprintf(stderr,
77             "Greska: Programa se poziva sa %s ime_datoteke\n",
78             argv[0]);
79         exit(EXIT_FAILURE);
80     }
81
82     /* Otvara se datoteka za citanje */
```

### 3 Algoritmi pretrage i sortiranja

```
83  ulaz = fopen(argv[1], "r");
84  if (ulaz == NULL) {
85      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s!\n",
86                  argv[1]);
87      exit(EXIT_FAILURE);
88  }
89
90  /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
91     tackama; i predstavlja indeks tekuce tacke */
92  i = 0;
93  while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94      i++;
95  }
96  n = i;
97
98  /* Proverava se koji su dodatni argumenti komandne linije. */
99  /* Ako nema dodatnih argumenata */
100 if (argc == 2)
101     /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
102     najbliza = najbliza_koordinatnom(tacke, n);
103 /* Inace proverava se koji je dodatni argument prosledjen. Ako je
104    u pitanju opcija -x */
105 else if (strcmp(argv[2], "-x") == 0)
106     /* Racuna se rastojanje u odnosu na x osu */
107     najbliza = najbliza_x_osi(tacke, n);
108 /* Ako je u pitanju opcija -y */
109 else if (strcmp(argv[2], "-y") == 0)
110     /* Racuna se rastojanje u odnosu na y osu */
111     najbliza = najbliza_y_osi(tacke, n);
112 else {
113     /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
114        korisnika i prekida se izvršavanje programa */
115     fprintf(stderr, "Greska: Pogresna opcija\n");
116     exit(EXIT_FAILURE);
117 }
118
119 /* Stampaju se koordinate trazene tacke */
120 printf("%g %g\n", najbliza.x, najbliza.y);
121
122 /* Zatvara se datoteka */
123 fclose(ulaz);
124
125 exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.6

```
#include <stdio.h>
2 #include <math.h>
3
4 /* Tacnost */
```

```

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

```

### 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_funkcije[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,
19            "Greska: Program se poziva sa %s ime_funkcije\n",

```

### 3 Algoritmi pretrage i sortiranja

```
20     argv[0]);
21     exit(EXIT_FAILURE);
22 }
23
24 /* Niska ime_funkcije sadrzi ime trazene funkcije koja je
25    navedena u komandnoj liniji */
26 strcpy(ime_funkcije, argv[1]);
27
28 /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
29 if (strcmp(ime_funkcije, "sin") == 0)
30     fp = &sin;
31 else if (strcmp(ime_funkcije, "cos") == 0)
32     fp = &cos;
33 else if (strcmp(ime_funkcije, "tan") == 0)
34     fp = &tan;
35 else if (strcmp(ime_funkcije, "atan") == 0)
36     fp = &atan;
37 else if (strcmp(ime_funkcije, "asin") == 0)
38     fp = &asin;
39 else if (strcmp(ime_funkcije, "log") == 0)
40     fp = &log;
41 else if (strcmp(ime_funkcije, "log10") == 0)
42     fp = &log10;
43 else {
44     fprintf(stderr,
45             "Greska: Program ne podrzava trazenu funkciju!\n");
46     exit(EXIT_SUCCESS);
47 }
48
49 printf("Unesite krajeve intervala: ");
50 scanf("%lf %lf", &l, &d);
51
52 if ((*fp)(l) * (*fp)(d) >= 0) {
53     fprintf(stderr, "Greska: %s na intervalu ", ime_funkcije);
54     fprintf(stderr, "[%g, %g] ne zadovoljava uslove\n", l, d);
55     exit(EXIT_FAILURE);
56 }
57
58 printf("Unesite preciznost: ");
59 scanf("%lf", &epsilon);
60
61 /* Sve dok se ne pronadje trazena vrednost argumenta */
62 while (1) {
63     /* Polovi se interval */
64     s = (l + d) / 2;
65     /* Ako je apsolutna vrednost trazene funkcije u ovoj tacki
66        manja od zadate tacnosti, prekida se pretraga */
67     if (fabs((*fp)(s)) < epsilon) {
68         break;
69     }
70     /* Ako je nula u levom delu intervala, nastavlja se pretraga na
71        [l, s] */
```

```

72     if ((*fp) (l) * (*fp) (s) < 0)
73         d = s;
74     else
75         /* Inace, na intervalu [s, d] */
76         l = s;
77     }
78
79     /* Stampa se vrednost trazene tacke */
80     printf("%g\n", s);
81
82     exit(EXIT_SUCCESS);
83 }

```

### Rešenje 3.8

```

#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
16         njegov prethodnik manji ili jednak nuli, pretraga je
17         završena */
18        if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
19            return s;
20        /* U slucaju broja manjeg ili jednakog nuli, pretrazuje se
21         desna polovina niza */
22        if (niz[s] <= 0)
23            l = s + 1;
24        /* A inace, leva polovina */
25        else
26            d = s - 1;
27    }
28    return -1;
29 }
30
31 int main()
32 {
33     int niz[MAX];
34     int n = 0;
35
36     /* Unos niza */

```

### 3 Algoritmi pretrage i sortiranja

```
38     while (scanf("%d", &niz[n]) == 1)
        n++;
40     /* Stampa se rezultat */
    printf("%d\n", prvi_veci_od_nule(niz, n));
42
    return 0;
44 }
```

#### Rešenje 3.9

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

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

39     /* Stampa se rezultat */
    printf("%d\n", prvi_manji_od_nule(niz, n));
```

```
41     return 0;
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      /* Rekurzivni korak */
10     return 1 + logaritam_a(x >> 1);
11 }
12
13 unsigned int logaritam_b(unsigned int x)
14 {
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
16     najvece vaznosti, tj. najlevlja. Pretragu se vrši od pozicije 0
17     do 31 */
18     int d = 0, l = sizeof(unsigned int) * 8 - 1;
19     int s;
20     /* Sve dok je desna granica pretrage desnije od leve */
21     while (d <= l) {
22         /* Racuna se sredisnja pozicija */
23         s = (l + d) / 2;
24         /* Proverava se da li je na toj poziciji trazena jedinica */
25         if ((1 << s) <= x && (1 << (s + 1)) > x)
26             return s;
27         /* Pretraga desne polovine binarnog zapisa */
28         if ((1 << s) > x)
29             l = s - 1;
30         /* Pretraga leve polovine binarnog zapisa */
31         else
32             d = s + 1;
33     }
34     return s;
35 }
36
37 int main()
38 {
39     unsigned int x;
40
41     /* Unos podatka */
42     scanf("%u", &x);
43
44     /* Provera da li je uneti broj pozitivan */
45     if (x == 0) {
```



### 3 Algoritmi pretrage i sortiranja

```
    fprintf(stderr, "Greska: Logaritam od nule nije definisan\n");
47    exit(EXIT_FAILURE);
    }
49
    /* Ispis povratnih vrednosti funkcija */
51    printf("%u %u\n", logaritam_a(x), logaritam_b(x));
53    exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.12

*sort.h*

```
#ifndef _SORT_H_
2 #define _SORT_H_ 1

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

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

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

```

38     koji je postavljen na pravu poziciju. Nakon toga se isti
        postupak ponavlja nad sve kracim i kracim prefiksima niza, cime
        se jedan po jedan istiskuju elementi na svoje prave pozicije. */
40 void bubble_sort(int a[], int n);

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

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

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

```

sort.c

```

1 #include "sort.h"

3 #define MAX 1000000

```

### 3 Algoritmi pretrage i sortiranja

```
5 void selection_sort(int a[], int n)
6 {
7     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 navedenih elemenata. */
15     for (i = 0; i < n - 1; i++) {
16         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
17         najmanji od elemenata a[i],...,a[n-1]. */
18         min = i;
19         for (j = i + 1; j < n; j++)
20             if (a[j] < a[min])
21                 min = j;
22
23         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo
24         ako su (i) i min razliciti, inace je nepotrebno. */
25         if (min != i) {
26             pom = a[i];
27             a[i] = a[min];
28             a[min] = pom;
29         }
30     }
31 }
32
33 void insertion_sort(int a[], int n)
34 {
35     int i, j;
36
37     /* Na pocetku iteracije pretpostavlja se da je niz
38     a[0],...,a[i-1] sortiran */
39     for (i = 1; i < n; i++) {
40         /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko
41         je potrebno, dok ne zauzme pravo mesto, tako da niz
42         a[0],...,a[i] bude sortiran. Indeks j je trenutna pozicija na
43         kojoj se element koji se umece nalazi. Petlja se zavrшава
44         ili kada element dodje do levog kraja (j==0) ili kada se
45         naidje na element a[j-1] koji je manji od a[j]. */
46         int temp = a[i];
47         for (j = i; j > 0 && temp < a[j - 1]; j--)
48             a[j] = a[j - 1];
49         a[j] = temp;
50     }
51 }
52
53 void bubble_sort(int a[], int n)
54 {
55     int i, j;
```

```
int ind;
57
for (i = n, ind = 1; i > 1 && ind; i--)
59     /* Poput "mehurica" potiskuje se najveći element među
        elementima od a[0] do a[i-1] na poziciju i-1 upoređujući
61     susedne elemente niza i potiskujući veći u desno */
    for (j = 0, ind = 0; j < i - 1; j++)
63         if (a[j] > a[j + 1]) {
            int temp = a[j];
65             a[j] = a[j + 1];
            a[j + 1] = temp;
67             /* Promenljiva ind registruje da je bilo premestanja. Samo
                u tom slučaju ima smisla ici na sledeću iteraciju, jer
69             ako nije bilo premestanja, znači da su svi elementi već
                u dobrom poretku, pa nema potrebe prelaziti na kraći
71             prefiks niza. Algoritam bi bio korektan i bez ovoga.
                Sortiranje bi bilo ispravno, ali manje efikasno, jer bi
73             se često nepotrebno vrsila mnoga upoređivanja, kada je
                već jasno da je sortiranje završeno. */
            ind = 1;
75         }
77 }

79 void shell_sort(int a[], int n)
    {
81     int h = n / 2, i, j;
        while (h > 0) {
83         /* Insertion sort sa korakom h */
            for (i = h; i < n; i++) {
85                 int temp = a[i];
                    j = i;
87                 while (j >= h && a[j - h] > temp) {
                        a[j] = a[j - h];
89                         j -= h;
                    }
91                 a[j] = temp;
            }
93         h = h / 2;
        }
95 }

97 void merge_sort(int a[], int l, int d)
    {
99     int s;
        static int b[MAX];           /* Pomocni niz */
101     int i, j, k;

103     /* Izlaz iz rekurzije */
        if (l >= d)
105         return;

107     /* Odredjivanje sredisnjeg indeksa */
```

### 3 Algoritmi pretrage i sortiranja

---

```
    s = (l + d) / 2;
109
    /* Rekurzivni pozivi */
111    merge_sort(a, l, s);
    merge_sort(a, s + 1, d);
113
    /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
115     niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
        prolazi kroz pomocni niz b[] */
117    i = l;
        j = s + 1;
119    k = 0;

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

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

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

141    /* Prepisuje se "ucesljani" niz u originalni niz */
    for (k = 0, i = l; i <= d; i++, k++)
143        a[i] = b[k];
}
145
/* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
147 void swap(int a[], int i, int j)
{
149     int tmp = a[i];
        a[i] = a[j];
151     a[j] = tmp;
}
153
void quick_sort(int a[], int l, int d)
155 {
    int i, pivot_pozicija;
157
    /* Izlaz iz rekurzije -- prazan niz */
159    if (l >= d)
```

```

    return;
161
162 /* Particionisanje niza. Svi elementi na pozicijama levo od
163 pivot_pozicija (izuzev same pozicije l) su strogo manji od
164 pivota. Kada se pronadje neki element manji od pivota, uvecava
165 se promenljiva pivot_pozicija i na tu poziciju se premesta
166 nadjeni element. Na kraju ce pivot_pozicija zaista biti
167 pozicija na koju treba smestiti pivot, jer ce svi elementi levo
168 od te pozicije biti manji a desno biti veci ili jednaki od
169 pivota. */
    pivot_pozicija = l;
171 for (i = l + 1; i <= d; i++)
        if (a[i] < a[l])
173             swap(a, ++pivot_pozicija, i);

175 /* Postavljanje pivota na svoje mesto */
    swap(a, l, pivot_pozicija);
177

178 /* Rekurzivno sortiranje elementa manjih od pivota */
179 quick_sort(a, l, pivot_pozicija - 1);
180 /* Rekurzivno sortiranje elementa vecih od pivota */
181 quick_sort(a, pivot_pozicija + 1, d);
}

```

*main.c*

```

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

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

8
int main(int argc, char *argv[])
10 {
11     /******
12     tip_sortiranja == 0 => selectionsort, (podrazumevano)
13     tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
14     tip_sortiranja == 2 => bubblesort, -b opcija komandne linije
15     tip_sortiranja == 3 => shellsort, -s opcija komandne linije
16     tip_sortiranja == 4 => mergesort, -m opcija komandne linije
17     tip_sortiranja == 5 => quicksort, -q opcija komandne linije
18     *****/
19     int tip_sortiranja = 0;
20     /******
21     tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
22     tip_niza == 1 => rastuce sortirani nizovi, -r opcija
23     tip_niza == 2 => opadajuce sortirani nizovi, -o opcija
24     *****/
    int tip_niza = 0;

```

### 3 Algoritmi pretrage i sortiranja

---

```
26
27 /* Dimenzija niza koji se sortira */
28 int dimenzija;
29 int i;
30 int niz[MAX];
31
32 /* Provera argumenata komandne linije */
33 if (argc < 2) {
34     fprintf(stderr, "Greska: Program zahteva bar 2 ");
35     fprintf(stderr, "argumenta komandne linije!\n");
36     exit(EXIT_FAILURE);
37 }
38
39 /* Ocitavaju se opcije i argumenati komandne linije */
40 for (i = 1; i < argc; i++) {
41     /* Ako je u pitanju opcija... */
42     if (argv[i][0] == '-') {
43         switch (argv[i][1]) {
44             case 'i':
45                 tip_sortiranja = 1;
46                 break;
47             case 'b':
48                 tip_sortiranja = 2;
49                 break;
50             case 's':
51                 tip_sortiranja = 3;
52                 break;
53             case 'm':
54                 tip_sortiranja = 4;
55                 break;
56             case 'q':
57                 tip_sortiranja = 5;
58                 break;
59             case 'r':
60                 tip_niza = 1;
61                 break;
62             case 'o':
63                 tip_niza = 2;
64                 break;
65             default:
66                 fprintf(stderr, "Greska: Pogresna opcija -%c\n",
67                     argv[i][1]);
68                 exit(EXIT_SUCCESS);
69                 break;
70         }
71     }
72     /* Ako je u pitanju argument, onda je to duzina niza koji treba
73     da se sortira */
74     else {
75         dimenzija = atoi(argv[i]);
76         if (dimenzija <= 0 || dimenzija > MAX) {
77             fprintf(stderr, "Greska: Dimenzija niza neodgovarajuca!\n");
78         }
79     }
80 }
```

```

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

```



### 3 Algoritmi pretrage i sortiranja

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

#### Rešenje 3.13

```
#include <stdio.h>
2 #include <string.h>
4 #define MAX_DIM 128
6 /* Funkcija za sortiranje niza karaktera */
void selectionSort(char s[])
8 {
    int i, j, min;
10    char pom;
    for (i = 0; s[i] != '\0'; i++) {
12        min = i;
        for (j = i + 1; s[j] != '\0'; j++)
14            if (s[j] < s[min])
                min = j;
16        if (min != i) {
            pom = s[i];
18            s[i] = s[min];
            s[min] = pom;
20        }
    }
22 }
24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
26 {
    int i;
28
    /* Ako dve niske imaju razlicit broj karaktera onda one nisu
30    anagrami */
    if (strlen(s) != strlen(t))
32        return 0;
34
    /* Sortiraju se karakteri */
    selectionSort(s);
36    selectionSort(t);
}
```

```

38  /* Dve sortirane niske su anagrami ako i samo ako su jednake */
    for (i = 0; s[i] != '\0'; i++)
40      if (s[i] != t[i])
          return 0;
42  return 1;
}
44
int main()
46 {
    char s[MAX_DIM], t[MAX_DIM];
48
    /* Ucitavaju se niske sa ulaza */
50  printf("Unesite prvu nisku: ");
    scanf("%s", s);
52  printf("Unesite drugu nisku: ");
    scanf("%s", t);
54
    /* Poziv funkcije */
56  if (anagrami(s, t))
        printf("jesu\n");
58  else
        printf("nisu\n");
60
    return 0;
62 }

```

### Rešenje 3.14

NAPOMENA: Rešenje koristi biblioteku za sortiranje celih brojeva iz zadatka 3.12.

```

1  #include <stdio.h>
    #include "sort.h"
3
    #define MAX 256
5
    /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
7     sortiranom nizu celih brojeva */
    int najmanje_rastojanje(int a[], int n)
9  {
    int i, min;
11  /* Postavlja se najmanje rastojanje na razliku prvog i drugog
        elementa niza */
13  min = a[1] - a[0];
    /* Za sve ostale susedne elemente proverava se njigova razlika */
15  for (i = 2; i < n; i++)
        if (a[i] - a[i - 1] < min)
17      min = a[i] - a[i - 1];
    return min;
19 }

```

### 3 Algoritmi pretrage i sortiranja

```
21 int main()
22 {
23     int i, a[MAX];
24
25     /* Ucitavaju se elementi niza sve do kraja ulaza */
26     i = 0;
27     while (scanf("%d", &a[i]) != EOF)
28         i++;
29
30     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
31     sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
32     se selection sort. */
33     selection_sort(a, i);
34
35     /* Ispisuje se rezultat */
36     printf("%d\n", najmanje_rastojanje(a, i));
37
38     return 0;
39 }
```

#### Rešenje 3.15

NAPOMENA: Rešenje koristi biblioteku za sortiranje celih brojeva iz zadatka 3.12.

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

```

29 {
30     int a[MAX_DIM], i;
31
32     /* Ucitavaju se elementi niza sve do kraja ulaza */
33     i = 0;
34     while (scanf("%d", &a[i]) != EOF)
35         i++;
36
37     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
38        sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
39        se merge sort. */
40     merge_sort(a, 0, i - 1);
41
42     /* Odredjuje se broj koji se najvise puta pojavio u nizu */
43     printf("%d\n", najvise_puta(a, i));
44
45     return 0;
46 }

```

### Rešenje 3.16

NAPOMENA: Rešenje koristi biblioteku za sortiranje celih brojeva iz zadatka 3.12.

```

1  #include <stdio.h>
2  #include "sort.h"
3
4  #define MAX_DIM 256
5
6  /* Funkcija za binarnu pretragu niza vraća 1 ako se element x
7     nalazi u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u
8     rastucem poretku */
9  int binarna_pretraga(int a[], int n, int x)
10 {
11     int levi = 0, desni = n - 1, srednji;
12
13     while (levi <= desni) {
14         srednji = (levi + desni) / 2;
15         if (a[srednji] == x)
16             return 1;
17         else if (a[srednji] > x)
18             desni = srednji - 1;
19         else if (a[srednji] < x)
20             levi = srednji + 1;
21     }
22     return 0;
23 }
24
25 int main()
26 {
27     int a[MAX_DIM], n = 0, zbir, i;

```

### 3 Algoritmi pretrage i sortiranja

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

33  /* Ucitavaju se elementi niza sve do kraja ulaza */
    i = 0;
35  printf("Unesite elemente niza: ");
    while (scanf("%d", &a[i]) != EOF)
37      i++;
    n = i;
39

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

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

55  return 0;
}
```

#### Rešenje 3.17

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

4 /* Funkcija objedinjuje nizove niz1 i niz2 dimenzija dim1 i dim2, a
   rezultat cuva u nizu niz3 za koji je rezervisano dim3 elemenata */
6 int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
           int dim3)
8 {
    int i = 0, j = 0, k = 0;
10    /* U slucaju da je dimenzija treceg niza manja od neophodne,
       funkcija vraca -1 */
12    if (dim3 < dim1 + dim2)
        return -1;
14

    /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja
       jednog od njih */
16    while (i < dim1 && j < dim2) {
18        if (niz1[i] < niz2[j])
            niz3[k++] = niz1[i++];
20        else
```

```

    niz3[k++] = niz2[j++];
22 }
    /* Ostatak prvog niza prepisuje se u treci */
24 while (i < dim1)
    niz3[k++] = niz1[i++];

26
    /* Ostatak drugog niza prepisuje se u treci */
28 while (j < dim2)
    niz3[k++] = niz2[j++];
30 return dim1 + dim2;
}

32
int main()
34 {
    int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
36 int i = 0, j = 0, k, dim3;

38 /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
    Pretpostavka je da na ulazu nema vise od MAX_DIM elemenata */
40 printf("Unesite elemente prvog niza: ");
    while (1) {
42 scanf("%d", &niz1[i]);
    if (niz1[i] == 0)
44 break;
    i++;
46 }
    printf("Unesite elemente drugog niza: ");
48 while (1) {
    scanf("%d", &niz2[j]);
50 if (niz2[j] == 0)
    break;
52 j++;
}

54
    /* Poziv trazene funkcije */
56 dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

58
    /* Ispis niza */
    for (k = 0; k < dim3; k++)
60 printf("%d ", niz3[k]);
    printf("\n");

62
    return 0;
64 }

```

### Rešenje 3.18

```

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

```

### 3 Algoritmi pretrage i sortiranja

```
int main(int argc, char *argv[])
6 {
    FILE *fin1 = NULL, *fin2 = NULL;
    FILE *fout = NULL;
    char ime1[11], ime2[11];
    char prezime1[16], prezime2[16];
    int kraj1 = 0, kraj2 = 0;

    /* Ako nema dovoljno argumenata komandne linije */
    14 if (argc < 3) {
        fprintf(stderr,
    16         "Greska: Program se poziva sa %s datoteka1 datoteka2\n",
            argv[0]);
    18     exit(EXIT_FAILURE);
    }

    /* Otvara se datoteka zadata prvim argumentom komandne linije */
    22 fin1 = fopen(argv[1], "r");
    if (fin1 == NULL) {
    24     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s\n",
            argv[1]);
    26     exit(EXIT_FAILURE);
    }

    /* Otvara se datoteka zadata drugim argumentom komandne linije */
    30 fin2 = fopen(argv[2], "r");
    if (fin2 == NULL) {
    32     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s\n",
            argv[2]);
    34     exit(EXIT_FAILURE);
    }

    /* Otvara se datoteka za upis rezultata */
    38 fout = fopen("ceo-tok.txt", "w");
    if (fout == NULL) {
    40     fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke ceo-tok.txt\n");
    42     exit(EXIT_FAILURE);
    }

    /* Cita se prvi student iz prve datoteke */
    46 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
        kraj1 = 1;

    /* Cita se prvi student iz druge datoteke */
    50 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
        kraj2 = 1;

    /* Sve dok nije dostignut kraj neke datoteke */
    54 while (!kraj1 && !kraj2) {
        int tmp = strcmp(ime1, ime2);
    56     if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 0)) {
```

```

58     /* Ime i prezime iz prve datoteke je leksikografski ranije, i
        biva upisano u izlaznu datoteku */
        fprintf(fout, "%s %s\n", ime1, prezime1);
60     /* Cita se naredni student iz prve datoteke */
        if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
62         kraj1 = 1;
    } else {
64     /* Ime i prezime iz druge datoteke je leksikografski ranije,
        i biva upisano u izlaznu datoteku */
        fprintf(fout, "%s %s\n", ime2, prezime2);
66     /* Cita se naredni student iz druge datoteke */
        if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
68         kraj2 = 1;
    }
70 }
72 }

74 /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
    druge datoteke, onda ima jos studenata u prvoj datoteci, koje
    treba prepisati u izlaznu, redom, jer su vec sortirani po
76 imenu. */
    while (!kraj1) {
78         fprintf(fout, "%s %s\n", ime1, prezime1);
        if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
80             kraj1 = 1;
    }

82

84 /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
    datoteke, onda ima jos studenata u drugoj datoteci, koje treba
    prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
86 while (!kraj2) {
        fprintf(fout, "%s %s\n", ime2, prezime2);
88         if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
            kraj2 = 1;
90     }

92     /* Zatvaraju se datoteke */
    fclose(fin1);
94     fclose(fin2);
    fclose(fout);

96     exit(EXIT_SUCCESS);
98 }

```

### Rešenje 3.19

```

1 #include <stdio.h>
  #include <string.h>
3 #include <math.h>
  #include <stdlib.h>
5
  #define MAX_BR_TACAKA 128

```



### 3 Algoritmi pretrage i sortiranja

---

```
7
/* Struktura koja reprezentuje koordinate tacke */
9 typedef struct Tacka {
    int x;
11    int y;
} Tacka;
13
/* Funkcija racuna rastojanje zadate tacke od koordinatnog pocetka */
15 float rastojanje(Tacka A)
{
17     return sqrt(A.x * A.x + A.y * A.y);
}
19
/* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
21 pocetka */
void sortiraj_po_rastojanju(Tacka t[], int n)
23 {
    int min, i, j;
25     Tacka tmp;

27     for (i = 0; i < n - 1; i++) {
        min = i;
29         for (j = i + 1; j < n; j++) {
            if (rastojanje(t[j]) < rastojanje(t[min])) {
31                 min = j;
            }
33         }
        if (min != i) {
35             tmp = t[i];
            t[i] = t[min];
37             t[min] = tmp;
        }
39     }
}
41
/* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
43 void sortiraj_po_x(Tacka t[], int n)
{
45     int min, i, j;
    Tacka tmp;
47

49     for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++) {
51             if (abs(t[j].x) < abs(t[min].x)) {
                min = j;
53             }
        }
55     if (min != i) {
        tmp = t[i];
57         t[i] = t[min];
        t[min] = tmp;
    }
}
```

```
59     }
60   }
61 }
62
63 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
64 void sortiraj_po_y(Tacka t[], int n)
65 {
66     int min, i, j;
67     Tacka tmp;
68
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;
74             }
75         }
76         if (min != i) {
77             tmp = t[i];
78             t[i] = t[min];
79             t[min] = tmp;
80         }
81     }
82 }
83
84 int main(int argc, char *argv[])
85 {
86     FILE *ulaz;
87     FILE *izlaz;
88     Tacka tacke[MAX_BR_TACAKA];
89     int i, n;
90
91     /* Proverava se broj argumenata komandne linije. Ocekuje se ime
92        izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
93        datoteke, tj. 4 argumenta */
94     if (argc != 4) {
95         fprintf(stderr,
96                 "Greska: Program se poziva sa %s opcija ulaz izlaz\n",
97                 argv[0]);
98         exit(EXIT_FAILURE);
99     }
100
101     /* Otvara se datoteka u kojoj su zadate tacke */
102     ulaz = fopen(argv[2], "r");
103     if (ulaz == NULL) {
104         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s!\n",
105                 argv[2]);
106         exit(EXIT_FAILURE);
107     }
108
109     /* Otvara se datoteka u koju treba upisati rezultat */
110     izlaz = fopen(argv[3], "w");
```

### 3 Algoritmi pretrage i sortiranja

```
111  if (izlaz == NULL) {
113      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s!\n",
          argv[3]);
          exit(EXIT_FAILURE);
115  }

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

125  /* Ukupan broj procitanih tacaka */
          n = i;

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

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

151  /* Zatvaraju se otvorene datoteke */
153  fclose(ulaz);
          fclose(izlaz);
155  exit(EXIT_SUCCESS);
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, min;
18     Gradjanin pom;
19
20     for (i = 0; i < n - 1; i++) {
21         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
22            najmanji od elemenata a[i].ime,...,a[n-1].ime. */
23         min = i;
24         for (j = i + 1; j < n; j++)
25             if (strcmp(a[j].ime, a[min].ime) < 0)
26                 min = j;
27         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo
28            ako su (i) i min razliciti, inace je nepotrebno. */
29         if (min != i) {
30             pom = a[i];
31             a[i] = a[min];
32             a[min] = pom;
33         }
34     }
35 }
36
37 /* Funkcija sortira niz gradjana rastuce po prezimenima */
38 void sort_prezime(Gradjanin a[], int n)
39 {
40     int i, j, min;
41     Gradjanin pom;
42
43     for (i = 0; i < n - 1; i++) {
44         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
45            najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
46         min = i;
47         for (j = i + 1; j < n; j++)
48             if (strcmp(a[j].prezime, a[min].prezime) < 0)
49                 min = j;
50         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo
```

### 3 Algoritmi pretrage i sortiranja

```
        ako su (i) i min razliciti, inace je nepotrebno. */
52     if (min != i) {
54         pom = a[i];
54         a[i] = a[min];
54         a[min] = pom;
56     }
58 }

60 /* Pretraga niza gradjana */
60 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
62 {
64     int i;
64     for (i = 0; i < n; i++)
66         if (strcmp(a[i].ime, x->ime) == 0
66             && strcmp(a[i].prezime, x->prezime) == 0)
68             return i;
68     return -1;
68 }

70

72 int main()
72 {
74     Gradjanin spisak1[MAX], spisak2[MAX];
74     int isti_rbr = 0;
76     int i, n;
76     FILE *fp = NULL;
78
78     /* Otvara se datoteka */
80     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
80         fprintf(stderr,
82             "Greska: Neupesno otvaranje datoteke za citanje.\n");
82         exit(EXIT_FAILURE);
84     }

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

92
94     /* Zatvara se datoteka */
94     fclose(fp);

96     sort_ime(spisak1, n);

98     /******
98     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
100     sortiranih nizova. Koristi se samo u fazi testiranja programa.

102     printf("Biracki spisak [uredjen prema imenima]:\n");
```

```

104     for(i=0; i<n; i++)
        printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
        *****/
106
108     sort_prezime(spisak2, n);
110
112     /******
        Ovaj deo je iskomentarisano jer se u zadatku ne trazi ispis
        sortiranih nizova. Koristi se samo u fazi testiranja programa.
        *****/
114     printf("Biracki spisak [uredjen prema prezimenima]:\n");
        for(i=0; i<n; i++)
            printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
116     *****/
118     /* Linearno se pretrazuju nizovi */
120     for (i = 0; i < n; i++)
        if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
            isti_rbr++;
122
124     /* Alternativno (efikasnije) resenje */
        /******
        for(i=0; i<n ;i++)
            if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
                strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
128             isti_rbr++;
        *****/
130
132     /* Ispisuje se rezultat */
        printf("%d\n", isti_rbr);
134
        exit(EXIT_SUCCESS);
    }

```

### Rešenje 3.22

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX_BR_RECII 128
7  #define MAX_DUZINA_RECII 32
8
9  /* Funkcija koja izracunava broj suglasnika u reci */
10 int broj_suglasnika(char s[])
11 {
12     char c;
13     int i;
14     int suglasnici = 0;
15     /* Prolazi karakter po karakter kroz zadatu nisku */

```

### 3 Algoritmi pretrage i sortiranja

```
17     for (i = 0; s[i]; i++) {
18         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
19            pokriven slucaj i malih i velikih suglasnika. */
20         if (isalpha(s[i])) {
21             c = toupper(s[i]);
22             /* Ukoliko slovo nije samoglasnik uvecava se brojac. */
23             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
24                 suglasnici++;
25         }
26     }
27     /* Vraca se izracunata vrednost */
28     return suglasnici;
29 }
30
31 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
32    duzini reci se mora proslediti zbog pravilnog upravljanja
33    memorijom */
34 void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)
35 {
36     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
37         duzina_j, duzina_min;
38     char tmp[MAX_DUZINA_RECII];
39     for (i = 0; i < n - 1; i++) {
40         min = i;
41         for (j = i; j < n; j++) {
42             /* Prvo se uporedjuje broj suglasnika */
43             broj_suglasnika_j = broj_suglasnika(reci[j]);
44             broj_suglasnika_min = broj_suglasnika(reci[min]);
45             if (broj_suglasnika_j < broj_suglasnika_min)
46                 min = j;
47             else if (broj_suglasnika_j == broj_suglasnika_min) {
48                 /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
49                    se duzine */
50                 duzina_j = strlen(reci[j]);
51                 duzina_min = strlen(reci[min]);
52
53                 if (duzina_j < duzina_min)
54                     min = j;
55                 else {
56                     /* Ako reci imaju i isti broj suglasnika i iste duzine,
57                        uporedjuju se leksikografski */
58                     if (duzina_j == duzina_min
59                         && strcmp(reci[j], reci[min]) < 0)
60                         min = j;
61                 }
62             }
63         }
64     }
65     if (min != i) {
66         strcpy(tmp, reci[min]);
67         strcpy(reci[min], reci[i]);
68         strcpy(reci[i], tmp);
69     }
70 }
```

```
69 }
71 int main()
72 {
73     FILE *ulaz;
74     int i = 0, n;
75
76     /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj
77        reci, a drugi maksimalnu duzinu pojedinačne reci */
78     char reci[MAX_BR_RECII][MAX_DUZINA_RECII];
79
80     /* Otvara se datoteka niske.txt za citanje */
81     ulaz = fopen("niske.txt", "r");
82     if (ulaz == NULL) {
83         fprintf(stderr,
84                "Greska: Neuspesno otvaranje datoteke niske.txt!\n");
85         exit(EXIT_FAILURE);
86     }
87
88     /* Sve dok se moze procitati sledeca rec */
89     while (fscanf(ulaz, "%s", reci[i]) != EOF) {
90         /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
91            prekida se ucitavanje */
92         if (i == MAX_BR_RECII)
93             break;
94         /* Priprema brojac za narednu iteraciju */
95         i++;
96     }
97
98     /* n je duzina niza reci i predstavlja poslednju vrednost
99        koriscenog brojac */
100    n = i;
101    /* Poziva se funkcija za sortiranje reci */
102    sortiraj_reci(reci, n);
103
104    /* Ispis sortiranog niza reci */
105    for (i = 0; i < n; i++) {
106        printf("%s ", reci[i]);
107    }
108    printf("\n");
109
110    /* Zatvara se datoteka */
111    fclose(ulaz);
112
113    exit(EXIT_SUCCESS);
114 }
```



#### Rešenje 3.23

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_ARTIKALA 100000
6
7 /* Struktura koja predstavlja jedan artikal */
8 typedef struct art {
9     long kod;
10    char naziv[20];
11    char proizvojac[20];
12    float cena;
13 } Artikal;
14
15 /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
16    traženim bar kodom */
17 int binarna_pretraga(Artikal a[], int n, long x)
18 {
19     int levi = 0;
20     int desni = n - 1;
21
22     /* Dokle god je indeks levi levo od indeksa desni */
23     while (levi <= desni) {
24         /* Racuna se sredisnji indeks */
25         int srednji = (levi + desni) / 2;
26         /* Ako je sredisnji element veci od traženog, tada se traženi
27            mora nalaziti u levoj polovini niza */
28         if (x < a[srednji].kod)
29             desni = srednji - 1;
30         /* Ako je sredisnji element manji od traženog, tada se traženi
31            mora nalaziti u desnoj polovini niza */
32         else if (x > a[srednji].kod)
33             levi = srednji + 1;
34         else
35             /* Ako je sredisnji element jednak traženom, tada je artikal
36                sa bar kodom x pronadjen na poziciji srednji */
37             return srednji;
38     }
39     /* Ako nije pronadjen artikal za traženim bar kodom, vraca se -1 */
40     return -1;
41 }
42
43 /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44 void selection_sort(Artikal a[], int n)
45 {
46     int i, j;
47     int min;
48     Artikal pom;
49
50     for (i = 0; i < n - 1; i++) {
```

```
52     min = i;
    for (j = i + 1; j < n; j++)
        if (a[j].kod < a[min].kod)
54             min = j;
    if (min != i) {
56         pom = a[i];
        a[i] = a[min];
58         a[min] = pom;
    }
60 }
}
62
int main()
64 {
    Artikal asortiman[MAX_ARTIKALA];
    long kod;
    int i, n;
    float racun;
    FILE *fp = NULL;
70
    /* Otvara se datoteka */
72     if ((fp = fopen("artikli.txt", "r")) == NULL) {
        fprintf(stderr,
74             "Greska: Neuspesno otvaranje datoteke artikli.txt.\n");
        exit(EXIT_FAILURE);
76     }
78
    /* Ucitavaju se artikali */
    i = 0;
80     while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
                    asortiman[i].naziv, asortiman[i].proizvodjac,
82                    &asortiman[i].cena) == 4)
        i++;
84
    /* Zatvara se datoteka */
86     fclose(fp);
88
    n = i;
90
    /* Sortira se celokupan asortiman prodavnice prema kodovima jer
    ce pri kucanju racuna prodavac unositi kod artikla. Prilikom
92     kucanja svakog racuna pretrazuje se asortiman, da bi se
    utvrdila cena artikla. Kucanje racuna obuhvata vise pretraga
94     asortimana i cilj je da ta operacija bude sto efikasnija. Zato
    se koristi algoritam binarne pretrage prilikom pretrazivanja
96     po kodu artikla. Iz tog razloga, potrebno je da asortiman bude
    sortiran po kodovima i to ce biti uradjeno primenom selection
98     sort algoritma. Sortiranje se vrši samo jednom na pocetku, ali
    se zato posle artikli mogu brzo pretrazivati prilikom kucanja
100     proizvoljno puno racuna. Vreme koje se utrosi na sortiranje na
    pocetku izvrsavanja programa, kasnije se isplati jer se za
102     brojna trazanja artikla umesto linearne moze koristiti
```

### 3 Algoritmi pretrage i sortiranja

```
104     efikasnija binarna pretraga. */
105     selection_sort(asortiman, n);
106
107     /* Ispis stanja u prodavnici */
108     printf
109         ("Asortiman:\nKOD           Naziv artikla       Ime
110          proizvodjaca       Cena\n");
111     for (i = 0; i < n; i++)
112         printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113             asortiman[i].naziv, asortiman[i].proizvodjac,
114             asortiman[i].cena);
115
116     kod = 0;
117     while (1) {
118         printf("-----\n");
119         printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
120         printf("- Za nov racun unesite kod artikla:\n");
121         /* Unos bar koda provog artikla sledeceg kupca */
122         if (scanf("%ld", &kod) == EOF)
123             break;
124         /* Trenutni racun novog kupca */
125         racun = 0;
126         /* Za sve artikle trenutnog kupca */
127         while (1) {
128             /* Vrsi se njihov pronalazak u nizu */
129             if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
130                 printf
131                     ("\tGreska: Ne postoji proizvod sa trazanim kodom!\n");
132             } else {
133                 printf("\tTrazili ste:\t%s %s %12.2f\n",
134                     asortiman[i].naziv, asortiman[i].proizvodjac,
135                     asortiman[i].cena);
136                 /* I dodaje se cena na ukupan racun */
137                 racun += asortiman[i].cena;
138             }
139             /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako
140              on nema vise artikla */
141             printf("Unesite kod artikla [ili 0 za prekid]: \t");
142             scanf("%ld", &kod);
143             if (kod == 0)
144                 break;
145         }
146         /* Stampa se ukupan racun trenutnog kupca */
147         printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
148     }
149
150     printf("Kraj rada kase!\n");
151     exit(EXIT_SUCCESS);
152 }
```

## 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
16    leksikografski rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
18 {
19     int i, j;
20     int min;
21     Student pom;
22
23     for (i = 0; i < n - 1; i++) {
24         min = i;
25         for (j = i + 1; j < n; j++)
26             if (strcmp(niz[j].ime, niz[min].ime) < 0)
27                 min = j;
28
29         if (min != i) {
30             pom = niz[min];
31             niz[min] = niz[i];
32             niz[i] = pom;
33         }
34     }
35 }
36
37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
38    zadataka opadajuće, a ukoliko neki studenti imaju isti broj
39    uradjenih zadataka sortiraju se po 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
```

### 3 Algoritmi pretrage i sortiranja

```
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 opadajuće po broju casova
62 na kojima su bili. Ukoliko neki studenti imaju isti broj casova,
63 sortiraju se opadajuće po broju uradjenih zadataka, a ukoliko se
64 i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65 prezimenu opadajuće. */
66 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
67 {
68     int i, j;
69     int max;
70     Student pom;
71     for (i = 0; i < n - 1; i++) {
72         max = i;
73         for (j = i + 1; j < n; j++)
74             if (niz[j].prisustvo > niz[max].prisustvo)
75                 max = j;
76             else if (niz[j].prisustvo == niz[max].prisustvo
77                     && niz[j].zadaci > niz[max].zadaci)
78                 max = j;
79             else if (niz[j].prisustvo == niz[max].prisustvo
80                     && niz[j].zadaci == niz[max].zadaci
81                     && strcmp(niz[j].prezime, niz[max].prezime) > 0)
82                 max = j;
83     if (max != i) {
84         pom = niz[max];
85         niz[max] = niz[i];
86         niz[i] = pom;
87     }
88 }
89 }
90
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     /* Otvara se datoteka za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
100         fprintf(stderr,
101             "Greska: Neuspesno otvaranje datoteke aktivnost.txt.\n");
102         exit(EXIT_FAILURE);
103     }
```

```
103 }
104
105 /* Ucitava se sadrzaj */
106 for (i = 0;
107      fscanf(fp, "%s%s%d", praktikum[i].ime,
108            praktikum[i].prezime, &praktikum[i].prisustvo,
109            &praktikum[i].zadaci) != EOF; i++);
110 /* Zatvara se datoteka */
111 fclose(fp);
112 br_studenata = i;
113
114 /* Kreira se prvi spisak studenata po prvom kriterijumu */
115 sort_ime_leksikografski(praktikum, br_studenata);
116 /* Otvara se datoteka za pisanje */
117 if ((fp = fopen("dat1.txt", "w")) == NULL) {
118     fprintf(stderr,
119            "Greska: Neupesno otvaranje datoteke dat1.txt.\n");
120     exit(EXIT_FAILURE);
121 }
122 /* Upisuje se niz u datoteku */
123 fprintf
124     (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
125 for (i = 0; i < br_studenata; i++)
126     fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
127           praktikum[i].prezime, praktikum[i].prisustvo,
128           praktikum[i].zadaci);
129 /* Zatvara se datoteka */
130 fclose(fp);
131
132 /* Kreira se drugi spisak studenata po drugom kriterijumu */
133 sort_zadatke_pa_imena(praktikum, br_studenata);
134 /* Otvara se datoteka za pisanje */
135 if ((fp = fopen("dat2.txt", "w")) == NULL) {
136     fprintf(stderr,
137            "Greska: Neupesno otvaranje datoteke dat2.txt.\n");
138     exit(EXIT_FAILURE);
139 }
140 /* Upisuje se niz u datoteku */
141 fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
142 fprintf(fp, "pa po duzini imena rastuce:\n");
143 for (i = 0; i < br_studenata; i++)
144     fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
145           praktikum[i].prezime, praktikum[i].prisustvo,
146           praktikum[i].zadaci);
147 /* Zatvara se datoteka */
148 fclose(fp);
149
150 /* Kreira se treci spisak studenata po trecem kriterijumu */
151 sort_prisustvo_pa_zadatke_pa_prezimana(praktikum, br_studenata);
152 /* Otvara se datoteka za pisanje */
153 if ((fp = fopen("dat3.txt", "w")) == NULL) {
154     fprintf(stderr,
```

### 3 Algoritmi pretrage i sortiranja

```
155         "Greska: Neuspesno otvaranje datoteke dat3.txt.\n");
156     exit(EXIT_FAILURE);
157 }
158 /* Upisuje se niz u datoteku */
159 fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
160 fprintf(fp, "pa po broju zadataka,\n");
161 fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
162 for (i = 0; i < br_studenata; i++)
163     fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
164             praktikum[i].prezime, praktikum[i].prisustvo,
165             praktikum[i].zadaci);
166 /* Zatvara se datoteka */
167 fclose(fp);
168
169 exit(EXIT_SUCCESS);
170 }
```

#### Rešenje 3.25

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define KORAK 10
6
7 /* Struktura koja opisuje jednu pesmu */
8 typedef struct {
9     char *izvodjac;
10    char *naslov;
11    int broj_gledanja;
12 } Pesma;
13
14 /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna
15    za rad qsort funkcije) */
16 int uporedi_gledanost(const void *pp1, const void *pp2)
17 {
18     Pesma *p1 = (Pesma *) pp1;
19     Pesma *p2 = (Pesma *) pp2;
20
21     return p2->broj_gledanja - p1->broj_gledanja;
22 }
23
24 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad
25    qsort funkcije) */
26 int uporedi_naslove(const void *pp1, const void *pp2)
27 {
28     Pesma *p1 = (Pesma *) pp1;
29     Pesma *p2 = (Pesma *) pp2;
30
31     return strcmp(p1->naslov, p2->naslov);
32 }
```

```
34 /* Funkcija za uporedjivanje pesama po izvodjaju (potrebna za rad
    qsort funkcije) */
36 int uporedi_izvodjace(const void *pp1, const void *pp2)
    {
38     Pesma *p1 = (Pesma *) pp1;
        Pesma *p2 = (Pesma *) pp2;
40
        return strcmp(p1->izvodjac, p2->izvodjac);
42     }

44 /* Funkcija koja oslobadja memoriju zauzetu dinamicnim nizom pesme
    dimenzije n */
46 void oslobodi(Pesma * pesme, int n)
    {
48     int i;
        for (i = 0; i < n; i++) {
50         free(pesme[i].izvodjac);
            free(pesme[i].naslov);
52     }
        free(pesme);
54     }

56 int main(int argc, char *argv[])
    {
58     FILE *ulaz;
        /* Pokazivac na deo memorije za cuvanje pesama */
60     Pesma *pesme;
        /* Broj mesta alociranih za pesme */
62     int alocirano_za_pesme;
        /* Redni broj pesme cije se informacije citaju */
64     int i;
        /* Ukupan broj pesama */
66     int n;
        int j;
68     char c;
        /* Broj mesta alociranih za propratne informacije o pesmama */
70     int alocirano;
        int broj_gledanja;
72
        /* Priprema se datoteka za citanje */
74     ulaz = fopen("pesme.txt", "r");
        if (ulaz == NULL) {
76         fprintf(stderr,
            "Greska: Neuspesno otvaranje ulazne datoteke!\n");
78         exit(EXIT_FAILURE);
        }
80
        /* Citaju se informacije o pesmama */
82     pesme = NULL;
        alocirano_za_pesme = 0;
84     i = 0;
```



### 3 Algoritmi pretrage i sortiranja

```
86 while (1) {
87     /* Proverava se da li je dostignut kraj datoteke */
88     c = fgetc(ulaz);
89     if (c == EOF) {
90         /* Nema vise sadrzaja za citanje */
91         break;
92     } else {
93         /* Inace, vraca se procitani karakter nazad */
94         ungetc(c, ulaz);
95     }
96
97     /* Proverava se da li postoji dovoljno vec alocirane memorije
98     za citanje nove pesme */
99     if (alocirano_za_pesme == i) {
100        /* Ako ne, ako je potrosena sva alocirana memorija, alocira
101        se novih KORAK mesta */
102        alocirano_za_pesme += KORAK;
103        pesme =
104            (Pesma *) realloc(pesme,
105                             alocirano_za_pesme * sizeof(Pesma));
106
107        /* Proverava se da li je nova memorija uspesno realocirana */
108        if (pesme == NULL) {
109            /* Ako nije ispisuje se obavestenje */
110            fprintf(stderr,
111                    "Greska: Problem sa alokacijom memorije!\n");
112            /* I oslobadja sva memorija zauzeta do ovog koraka */
113            oslobodi(pesme, i);
114            exit(EXIT_FAILURE);
115        }
116    }
117
118    /* Ako jeste, nastavlja se sa citanjem pesama ... */
119    /* Cita se ime izvadjaca */
120    /* Pozicija na koju treba smestiti procitani karakter */
121    j = 0;
122    /* Broj alociranih mesta */
123    alocirano = 0;
124    /* Memorija za smestanje procitanih karaktera */
125    pesme[i].izvodjac = NULL;
126
127    /* Sve do prve beline u liniji (beline koja se nalazi nakon
128    imena izvadjaca) citaju se karakteri iz datoteke */
129    while ((c = fgetc(ulaz)) != ' ') {
130        /* Provera da li postoji dovoljno memorije za smestanje
131        procitanog karaktera */
132        if (j == alocirano) {
133
134            /* Ako ne, ako je potrosena sva alocirana memorija, alocira
135            se novih KORAK mesta */
136            alocirano += KORAK;
```

```
138     pesme[i].izvodjac =
        (char *) realloc(pesme[i].izvodjac,
                        alocirano * sizeof(char));
140
142     /* Provera da li je nova alokacija uspesna */
144     if (pesme[i].izvodjac == NULL) {
        /* Ako nije oslobadja se sva memorija zauzeta do ovog
        koraka */
        oslobodi(pesme, i);
146     /* I prekida sa izvršavanjem programa */
        exit(EXIT_FAILURE);
148     }
    }
150     /* Ako postoji dovoljno alocirane memorije, smesta se vec
    procitani karakter */
152     pesme[i].izvodjac[j] = c;
    j++;
154     /* I nastavlja se sa citanjem */
}

156
158     /* Upis terminirajuće nule na kraj reci */
    pesme[i].izvodjac[j] = '\0';

160     /* Preskace se karakter - */
    fgetc(ulaz);

162
164     /* Preskace se razmak */
    fgetc(ulaz);

166     /* Cita se naslov pesme */
    /* Pozicija na koju treba smestiti procitani karakter */
168     j = 0;
    /* Broj alociranih mesta */
170     alocirano = 0;
    /* Memorija za smestanje procitanih karaktera */
172     pesme[i].naslov = NULL;

174     /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
    karakteri iz datoteke */
176     while ((c = fgetc(ulaz)) != ',') {
        /* Provera da li postoji dovoljno memorije za smestanje
        procitanog karaktera */
178         if (j == alocirano) {
            /* Ako ne, ako je potrosena sva alocirana memorija, alocira
            se novih KORAK mesta */
182             alocirano += KORAK;
            pesme[i].naslov =
184                 (char *) realloc(pesme[i].naslov,
                                    alocirano * sizeof(char));
186
            /* Provera da li je nova alokacija uspesna */
188             if (pesme[i].naslov == NULL) {
```

### 3 Algoritmi pretrage i sortiranja

```
190         /* Ako nije, oslobadja se sva memorija zauzeta do ovog
191            koraka */
192         free(pesme[i].izvodjac);
193         oslobodi(pesme, i);
194         /* I prekida izvršavanje programa */
195         exit(EXIT_FAILURE);
196     }
197     /* Smesta se procitani karakter */
198     pesme[i].naslov[j] = c;
199     j++;
200     /* I nastavlja dalje sa citanjem */
201 }
202 /* Upisuje se terminirajuca nula na kraj reci */
203 pesme[i].naslov[j] = '\0';
204
205 /* Preskace se razmak */
206 fgetc(ulaz);
207
208 /* Cita se broj gledanja */
209 broj_gledanja = 0;
210
211 /* Sve do znaka za novi red (kraja linije) citaju se karakteri
212    iz datoteke */
213 while ((c = fgetc(ulaz)) != '\n') {
214     broj_gledanja = broj_gledanja * 10 + (c - '0');
215 }
216 pesme[i].broj_gledanja = broj_gledanja;
217
218 /* Prelazi se na citanje sledece pesme */
219 i++;
220 }
221
222 /* Informacija o broju procitanih pesama */
223 n = i;
224 /* Zatvara se datoteka */
225 fclose(ulaz);
226
227 /* Analiza argumenta komandne linije */
228 if (argc == 1) {
229     /* Nema dodatnih opcija => sortiranje po broju gledanja */
230     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
231 } else {
232     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
233         /* Sortira se po naslovu */
234         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
235     } else {
236         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
237             /* Sortira se po izvodjacu */
238             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
239         } else {
240             fprintf(stderr, "Greska: Nedoovoljeni argumenti!\n");
```

```

        free(pesme);
242     exit(EXIT_FAILURE);
    }
244 }
}

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

252 /* Oslobadja se memorija */
254 oslobodi(pesme, n);

256 exit(EXIT_SUCCESS);
}

```

### Rešenje 3.28

NAPOMENA: Rešenje koristi biblioteku za rad sa celobrojnim matricama iz zadatka 2.19.

```

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

4

/* Funkcija odredjuje zbir v-te vrste matrice a sa m kolona */
6 int zbir_vrste(int **a, int v, int m)
{
8     int i, zbir = 0;

10    for (i = 0; i < m; i++) {
        zbir += a[v][i];
12    }
    return zbir;
14 }

16 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
    osnovu zbira elemenata koriscenjem selection sort algoritma */
18 void sortiraj_vrste(int **a, int n, int m)
{
20     int i, j, min;

22     for (i = 0; i < n - 1; i++) {
        min = i;
24         for (j = i + 1; j < n; j++) {
            if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
26                 min = j;
            }
        }
28     }
}

```

### 3 Algoritmi pretrage i sortiranja

```
        if (min != i) {
30         int *tmp;
           tmp = a[i];
32         a[i] = a[min];
           a[min] = tmp;
34     }
    }
36 }

38 int main(int argc, char *argv[])
    {
40     int **a, n, m;

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

46     /* Alokacija memorije */
       a = alociraj_matricu(n, m);
48     if (a == NULL) {
           fprintf(stderr, "Greska: Neuspesna alokacija matrice\n");
50         exit(EXIT_FAILURE);
       }
52
       /* Ucitavaju se elementi matrice */
54     printf("Unesite elemente matrice po vrstama:\n");
       ucitaj_matricu(a, n, m);
56
       /* Poziva se funkcija koja sortira vrste matrice prema zbiru */
58     sortiraj_vrste(a, n, m);

60     /* Ispisuje se rezultujuca matrica */
       printf("Sortirana matrica je:\n");
62     ispisi_matricu(a, n, m);

64     /* Oslobadja se memorija */
       a = dealociraj_matricu(a, n);
66
       exit(EXIT_SUCCESS);
68 }
```

#### Rešenje 3.31

```
1 #include <stdio.h>
   #include <stdlib.h>
3 #include <math.h>
   #include <search.h>
5
   #define MAX 100
7
   /* Funkcija poredjenja dva cela broja (neopadajuci poredak) */
```

```
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. */
13     int b1 = *((int *) a);
14     int b2 = *((int *) b);
15
16     /* Vrsi se poredjenje ovih vrednosti. */
17     if (b1 > b2)
18         return 1;
19     else if (b1 < b2)
20         return -1;
21     else
22         return 0;
23
24     /******
25        Umesto poredjenja, moze se koristiti naredba
26        return b1 - b2;
27        Ipak, zbog moguceg prekoračenja prilikom oduzimanja, ovakvo
28        resenje se koristi samo onda kada imamo ogranicene vrednosti
29        promenljivih koje poredimo tj. kada ocekujemo da do
30        prekoračenja ne moze da dodje.
31        *****/
32 }
33
34 /* Funkcija poredjenja dva cela broja (nerastuci poredak) */
35 int poredi_int_nerastuce(const void *a, const void *b)
36 {
37     /* Za obrnuti poredak treba samo promeniti znak vrednosti koju
38        koju vraca prethodna funkcija */
39     return -poredi_int(a, b);
40 }
41
42 int main()
43 {
44     size_t n;
45     int i, x;
46     int a[MAX], *p = NULL;
47
48     /* Unos dimenzije */
49     printf("Uneti dimenziju niza: ");
50     scanf("%ld", &n);
51     if (n > MAX)
52         n = MAX;
53
54     /* Unos elementa niza */
55     printf("Uneti elemente niza:\n");
56     for (i = 0; i < n; i++)
57         scanf("%d", &a[i]);
58
59     /* Sortira se niz celih brojeva */
60     qsort(a, n, sizeof(int), &poredi_int);
```

### 3 Algoritmi pretrage i sortiranja

```
61  /* Prikazuje se sortirani niz */
63  printf("Sortirani niz u rastucem poretku:\n");
65  for (i = 0; i < n; i++)
66      printf("%d ", a[i]);
67      putchar('\n');
69  /* Pretrazuje se niz */
70  /* Vrednost koja ce biti trazena u nizu */
71  printf("Uneti element koji se trazi u nizu: ");
72  scanf("%d", &x);
74  /* Binarna pretraga */
75  printf("Binarna pretraga: \n");
76  p = bsearch(&x, a, n, sizeof(int), &poredi_int);
77  if (p == NULL)
78      printf("Elementa nema u nizu!\n");
79  else
80      printf("Element je nadjen na poziciji %ld\n", p - a);
81  /* Linearna pretraga */
82  printf("Linearna pretraga (lfind): \n");
83  p = lfind(&x, a, &n, sizeof(int), &poredi_int);
84  if (p == NULL)
85      printf("Elementa nema u nizu!\n");
86  else
87      printf("Element je nadjen na poziciji %ld\n", p - a);
89  return 0;
}
```

#### 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_delilaca(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_delilaca(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = broj_delilaca(ak);
42     int n_d_b = broj_delilaca(bk);
43
44     return n_d_a - n_d_b;
45 }
46
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 elementa niza */
60     printf("Uneti elemente niza:\n");
61     for (i = 0; i < n; i++)
62         scanf("%d", &a[i]);
63
64     /* Sortira se niz celih brojeva prema broju delilaca */
65     qsort(a, n, sizeof(int), &poredi_po_broju_delilaca);
66
67     /* Prikazuje se sortirani niz */
68     printf
69         ("Sortirani niz u rastucem poretku prema broju delilaca:\n");
70     for (i = 0; i < n; i++)
```



### 3 Algoritmi pretrage i sortiranja

```
71     printf("%d ", a[i]);
    putchar('\n');
73
    return 0;
75 }
```

#### Rešenje 3.33

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <search.h>
5
   #define MAX_NISKI 1000
7  #define MAX_DUZINA 31
9
   /******
   Niz nizova karaktera ovog potpisa
11  char niske[3][4];
   se moze graficki predstaviti ovako:
13  -----
   | a | b | c | \0 || d | e | \0|  || f | g | h | \0||
15  -----
   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
17  svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
   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
   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23  koji trebaju biti uporedjeni, (npr. pri porecjenju prve i poslednje
   reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
   nad njima.
27  *****/
   int poredi_leksikografski(const void *a, const void *b)
29  {
   return strcmp((char *) a, (char *) b);
31  }
33
   /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
35  int poredi_duzine(const void *a, const void *b)
   {
37  return strlen((char *) a) - strlen((char *) b);
   }
39
   int main()
41  {
   int i;
43  size_t n;
```

```
45 FILE *fp = NULL;
46 char niske[MAX_NISKI][MAX_DUZINA];
47 char *p = NULL;
48 char x[MAX_DUZINA];
49
50 /* Otvara se datoteka */
51 if ((fp = fopen("niske.txt", "r")) == NULL) {
52     fprintf(stderr,
53         "Greska: Neupesno otvaranje datoteke niske.txt.\n");
54     exit(EXIT_FAILURE);
55 }
56
57 /* Cita se sadrzaj datoteke */
58 for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
59
60 /* Zatvara se datoteka */
61 fclose(fp);
62 n = i;
63
64 /* Sortiraju se niske leksikografski. Biblioteckoj funkciji qsort
65    prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
66    niske po duzini */
67 qsort(niske, n, MAX_DUZINA * sizeof(char),
68     &poredi_leksikografski);
69
70 printf("Leksikografski sortirane niske:\n");
71 for (i = 0; i < n; i++)
72     printf("%s ", niske[i]);
73 printf("\n");
74
75 /* Unosi se trazena niska */
76 printf("Uneti trazenu nisku: ");
77 scanf("%s", x);
78
79 /* Binarna pretraga */
80 /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
81    je niz vec sortiran leksikografski. */
82 p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
83     &poredi_leksikografski);
84
85 if (p != NULL)
86     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
87         p, (p - (char *) niske) / MAX_DUZINA);
88 else
89     printf("Niska nije pronadjena u nizu\n");
90
91 /* Sortira se po duzini */
92 qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
93
94 printf("Niske sortirane po duzini:\n");
95 for (i = 0; i < n; i++)
96     printf("%s ", niske[i]);
```

### 3 Algoritmi pretrage i sortiranja

```
printf("\n");
97
/* Linearna pretraga */
99 p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
          &poredi_leksikografski);
101 if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
103         p, (p - (char *) niske) / MAX_DUZINA);
    else
105     printf("Niska nije pronadjena u nizu\n");
107 exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.34

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #include <search.h>

6 #define MAX_NISKI 1000
#define MAX_DUZINA 31
8

/*****
10 Niz pokazivaca na karaktere ovog potpisa
char *niske[3];
12 posle alokacije u main-u se moze graficki predstaviti ovako:
-----
14 | X | -----> | a | b | c | \0|
-----
16 | Y | -----> | d | e | \0|
-----
18 | Z | -----> | f | g | h | \0|
-----

20 Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
njegov element pokazivac koji pokazuje na alocirane karaktere i-te
22 reci. Njegov tip je char*.

24 Kako pokazivaci a i b u sledecoj funkciji sadrže adrese elemenata
koji trebaju biti upoređeni (recimo adresu od X i adresu od Z), i
26 kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
moraju i kastovati. Da bi se leksikografski uporedili elementi X i
28 Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
30 kastovani na odgovarajuci tip.
*****/
32 int poredi_leksikografski(const void *a, const void *b)
{
34     return strcmp(*(char **) a, *(char **) b);
}
```

```
36 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
37     leksikografski, vec po duzini */
38 int poredi_duzine(const void *a, const void *b)
39 {
40     return strlen(*(char **) a) - strlen(*(char **) b);
41 }
42
43
44 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
45     element u nizu sa kojim se poredi, pa njega treba kastovati na
46     char** i dereferencirati, (videti obrazlozenje za prvu funkciju
47     u ovom zadatku, a pokazivac a pokazuje na element koji se trazi.
48     U main funkciji je to x, koji je tipa char*, tako da pokazivac a
49     ovde samo treba kastovati i ne dereferencirati. */
50 int poredi_leksikografski_b(const void *a, const void *b)
51 {
52     return strcmp((char *) a, *(char **) b);
53 }
54
55 int main()
56 {
57     int i;
58     size_t n;
59     FILE *fp = NULL;
60     char *niske[MAX_NISKI];
61     char **p = NULL;
62     char x[MAX_DUZINA];
63
64     /* Otvara se datoteka */
65     if ((fp = fopen("niske.txt", "r")) == NULL) {
66         fprintf(stderr,
67             "Greska: Neupesno otvaranje datoteke niske.txt.\n");
68         exit(EXIT_FAILURE);
69     }
70
71     /* Cita se sadrzaj datoteke */
72     i = 0;
73     while (fscanf(fp, "%s", x) != EOF) {
74         /* Alocira se dovoljno memorije za i-tu nisku */
75         if ((niske[i] = malloc((strlen(x) + 1) * sizeof(char))) == NULL)
76             {
77             fprintf(stderr, "Greska: Neuspesna alokacija niske\n");
78             exit(EXIT_FAILURE);
79         }
80         /* Kopira se procitana niska na svoje mesto */
81         strcpy(niske[i], x);
82         i++;
83     }
84
85     /* Zatvara se datoteka */
86     fclose(fp);
87     n = i;
```

### 3 Algoritmi pretrage i sortiranja

```
88  /* Sortiraju se niske leksikografski. Biblioteckoj funkciji qsort
90     se prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
92     niske po duzini */
    qsort(niske, n, sizeof(char *), &poredi_leksikografski);

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

98     /* Unosi se trazena niska */
    printf("Uneti trazenu nisku: ");
100    scanf("%s", x);

102    /* Binarna pretraga */
    p = bsearch(x, niske, n, sizeof(char *),
104                &poredi_leksikografski_b);
    if (p != NULL)
106        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
                *p, p - niske);
108    else
        printf("Niska nije pronadjena u nizu\n");
110

112    /* Linearna pretraga */
    p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
114    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
                *p, p - niske);
116    else
        printf("Niska nije pronadjena u nizu\n");
118

120    /* Sortira se po duzini */
    qsort(niske, n, sizeof(char *), &poredi_duzine);
    printf("Niske sortirane po duzini:\n");
122    for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
124    printf("\n");

126    /* Oslobadja se zauzeta memorija */
    for (i = 0; i < n; i++)
128        free(niske[i]);

130    exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.35

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

```
4 #include <search.h>
6 #define MAX 500
8 /* Struktura sa svim informacijama o pojedinacnom studentu */
9 typedef struct {
10     char ime[21];
11     char prezime[21];
12     int bodovi;
13 } Student;
14
15 /* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
16     istim brojem bodova se dodatno sortiraju leksikografski po
17     prezimenu */
18 int poredi1(const void *a, const void *b)
19 {
20     Student *prvi = (Student *) a;
21     Student *drugi = (Student *) b;
22
23     if (prvi->bodovi > drugi->bodovi)
24         return -1;
25     else if (prvi->bodovi < drugi->bodovi)
26         return 1;
27     else
28         /* 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
34     bodova. Prvi parametar je ono sto se trazi u nizu (broj bodova),
35     a drugi 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 porede. */
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];
```

### 3 Algoritmi pretrage i sortiranja

```
56  int i;
57  size_t br_studenata = 0;
58  Student *nadjen = NULL;
59  FILE *fp = NULL;
60  int bodovi;
61  char prezime[21];
62
63  /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64   informacija korisniku kako se program koristi i prekida se
65   izvršavanje. */
66  if (argc < 2) {
67      fprintf(stderr, "Greska: Program se poziva sa %s datoteka\n",
68              argv[0]);
69      exit(EXIT_FAILURE);
70  }
71
72  /* Otvara se datoteka */
73  if ((fp = fopen(argv[1], "r")) == NULL) {
74      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s\n",
75              argv[1]);
76      exit(EXIT_FAILURE);
77  }
78
79  /* Ucitava se sadrzaj */
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  /* Zatvara se datoteka */
86  fclose(fp);
87  br_studenata = i;
88
89  /* Sortira se niz studenata po broju bodova, gde se unutar grupe
90   studenata sa istim brojem bodova sortiranje vrši po prezimenu */
91  qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
92
93  printf("Studenti sortirani po broju poena opadajuće, ");
94  printf("pa po prezimenu rasteće:\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  /* Pretražuju se studenati po broju bodova binarnom pretragom jer
100   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);
```

```

108  if (nadjen != NULL)
        printf
110      ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
        nadjen->ime, nadjen->prezime, nadjen->bodovi);
112  else
        printf("Nema studenta sa unetim brojem bodova\n");
114
        /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116         sortiran po bodovima. */
        printf("Unesite prezime: ");
118  scanf("%s", prezime);

120  nadjen =
        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
122         &poredi3);

124  if (nadjen != NULL)
        printf
126      ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
        nadjen->ime, nadjen->prezime, nadjen->bodovi);
128  else
        printf("Nema studenta sa unetim prezimenom\n");
130
        exit(EXIT_SUCCESS);
132 }

```

### Rešenje 3.36

```

#include <stdio.h>
2  #include <string.h>
#include <stdlib.h>
4
#define MAX 128
6
/* Funkcija poredi dva karaktera */
8  int uporedi_char(const void *pa, const void *pb)
{
10     return *(char *) pa - *(char *) pb;
}
12
/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14  int anagrami(char s[], char t[])
{
16     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
    if (strlen(s) != strlen(t))
18         return 0;

20     /* Sortiraju se karakteri u niskama */
    qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);

```



### 3 Algoritmi pretrage i sortiranja

```
24  /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
      suprotnom, nisu */
26  return !strcmp(s, t);
   }

28  int main()
30  {
      char s[MAX], t[MAX];

32

34  /* Unose se niske */
      printf("Unesite prvu nisku: ");
      scanf("%s", s);
36  printf("Unesite drugu nisku: ");
      scanf("%s", t);

38

40  /* Ispituje se da li su niske anagrami */
      if (anagrami(s, t))
          printf("jesu\n");
42  else
          printf("nisu\n");

44

46  return 0;
   }
```

#### Rešenje 3.37

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

5  #define MAX 10
   #define MAX_DUZINA 32

7

9  /* Funkcija poredjenja */
   int uporedi_niske(const void *pa, const void *pb)
   {
11     return strcmp((char *) pa, (char *) pb);
   }

13

15  int main()
   {
17     int i, n;
       char S[MAX][MAX_DUZINA];

19     /* Unosi se broj niski */
       printf("Unesite broj niski:");
21     scanf("%d", &n);

23     /* Unosi se niz niski */
       printf("Unesite niske:\n");
25     for (i = 0; i < n; i++)
```

```

    scanf("%s", S[i]);
27
    /* Sortira se niz niski */
29    qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

31    /*****
    Ovaj deo je iskomentarisano jer se u zadatku ne trazi ispis
33    sortiranih niski. Koriscen je samo u fazi testiranja programa.

    printf("Sortirane niske su:\n");
35    for(i = 0; i < n; i++)
37        printf("%s ", S[i]);
    *****/

39
41    /* Ako postoje dve iste niske u nizu, onda ce one nakon
    sortiranja niza biti jedna do druge */
43    for (i = 0; i < n - 1; i++)
45        if (strcmp(S[i], S[i + 1]) == 0) {
47            printf("ima\n");
49            return 0;
        }
    printf("nema\n");
    return 0;
}

```

### Rešenje 3.38

```

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

5  #define MAX 21

7  /* Struktura koja predstavlja jednog studenta */
   typedef struct student {
9      char nalog[8];
   char ime[MAX];
11  char prezime[MAX];
   int poeni;
13 } Student;

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

23 /* Funkcija poredi studente prvo prema godini, zatim prema smeru i

```

### 3 Algoritmi pretrage i sortiranja

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

49 /* Funkcija poredjenja po nalogu za upotrebu u biblioteckoj
        funkciji bsearch */
51 int uporedi_bsearch(const void *a, const void *b)
    {
53     /* Nalog studenta koji se trazi */
        char *nalog = (char *) a;
55     /* Kljuc pretrage */
        Student s = *(Student *) b;
57
59     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
        int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
61     char smer1 = nalog[1];
        char smer2 = s.nalog[1];
63     int indeks1 =
        (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + (nalog[6] -
        '0');
65     int indeks2 =
        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
67     (s.nalog[6] - '0');
        if (godina1 != godina2)
69     return godina1 - godina2;
        else if (smer1 != smer2)
71     return smer1 - smer2;
        else
73     return indeks1 - indeks2;
75 }
```

```
int main(int argc, char **argv)
77 {
    Student *nadjen = NULL;
79     char nalog_trazeni[8];
    Student niz_studenata[100];
81     int i = 0, br_studenata = 0;
    FILE *in = NULL, *out = NULL;
83
    /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
85     korisnik nije ispravno pokrenuo program. */
    if (argc != 2 && argc != 3) {
87         fprintf(stderr,
            "Greska: Program se poziva sa %s -opcija [nalog]\n",
89             argv[0]);
        exit(EXIT_FAILURE);
91     }

93     /* Otvara se datoteka za citanje */
    in = fopen("studenti.txt", "r");
95     if (in == NULL) {
        fprintf(stderr,
97             "Greska: Neuspesno otvaranje datoteke studenti.txt!\n");
        exit(EXIT_FAILURE);
99     }

101    /* Otvara se datoteka za pisanje */
    out = fopen("izlaz.txt", "w");
103    if (out == NULL) {
        fprintf(stderr,
105             "Greska: Neuspesno otvaranje datoteke izlaz.txt!\n");
        exit(EXIT_FAILURE);
107    }

109    /* Ucitavaju se studenti iz ulazne datoteke sve do njenog kraja */
    while (fscanf
111        (in, "%s %s %s %d", niz_studenata[i].nalog,
            niz_studenata[i].ime, niz_studenata[i].prezime,
113            &niz_studenata[i].poeni) != EOF)
        i++;
115
    br_studenata = i;
117

    /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
119    if (strcmp(argv[1], "-p") == 0)
        qsort(niz_studenata, br_studenata, sizeof(Student),
            &uporedi_poeni);
121    /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
123    else if (strcmp(argv[1], "-n") == 0)
        qsort(niz_studenata, br_studenata, sizeof(Student),
125            &uporedi_nalog);

127    /* Sortirani studenti se ispisuju u izlaznu datoteku */
```

### 3 Algoritmi pretrage i sortiranja

---

```
129     for (i = 0; i < br_studenata; i++)
        fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
            niz_studenata[i].ime, niz_studenata[i].prezime,
131             niz_studenata[i].poeni);

133     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
        studenta... */
135     if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
        strcpy(nalog_trazeni, argv[2]);

137         /* ... pronalazi se student sa tim nalogom. */
139         nadjen =
            (Student *) bsearch(nalog_trazeni, niz_studenata,
141                             br_studenata, sizeof(Student),
                                &uporedi_bsearch);

143         if (nadjen == NULL)
145             printf("Nije nadjen!\n");
        else
147             printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
                                nadjen->prezime, nadjen->poeni);
149     }

151     /* Zatvaraju se datoteke */
        fclose(in);
153     fclose(out);

155     exit(EXIT_SUCCESS);
}
```

## 4

# Dinamičke strukture podataka

## 4.1 Liste

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

- (a) Definisati strukturu `Cvor` kojom se predstavlja čvor liste, a koja 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`.

#### 4 Dinamičke strukture podataka

---

- (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 element u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje vrednosti u čvorovima liste uokvirene zagradama `[, ]` i međusobno razdvojene zaptetama.
- (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`. Funkcija vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.
- (k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.
- (l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.
- (n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ako je sve bilo u redu, odnosno 1, ako 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. Zatim se unosi ceo broj koji se traži u unetoj listi i na standardni izlaz se ispisuje rezultat pretrage.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CTRL+D za kraj unosa):
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 traži: 5
Trazeni broj 5 je u listi!

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CTRL+D za kraj unosa):
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [35, 14, 23]

Unesite broj koji se traži: 8
Broj 8 se ne nalazi u listi!

```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Zatim se unosi ceo broj čija se sva pojavljivanja u listi brišu i na standardni izlaz se ispisuje sadržaj liste nakon brisanja.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CTRL+D za kraj unosa):
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 briše: 3
Lista nakon brisanja: [2, 14, 17]

```

*Primer 2*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CTRL+D za kraj unosa):
23
Lista: [23]
14
Lista: [23, 14]
35
Lista: [23, 14, 35]

Unesite broj koji se briše: 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 standardni izlaz 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.*



## 4 Dinamičke strukture podataka

---

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CTRL+D za kraj unosa):
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 traži: 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 (CTRL+D za kraj unosa):
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [14, 23, 35]

Unesite broj koji se traži: 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 main funkcije 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. Etiketne smeštati u listu, a za formiranje liste koristiti strukturu `Element` koja sadrži neoznačen broj pojavljivanja etiketi, nisku karaktera koja može da prihvati etiketu veličine do 20 karaktera i pokazivač na sledeći element liste.

### Test 2

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML
<html>
<head><title>Primer</title></head>
<body>
  <h1>Naslov</h1>
  Danas je lep i sunčan dan. <br>
  A sutra ce biti jos lepsi.
  <a link='http://www.google.com'> Link 1</a>
  <a link='http://www.math.rs'> Link 2</a>
</body>
</html>
```

```
IZLAZ:
a - 4
br - 1
h1 - 2
body - 2
title - 2
head - 2
html - 2
```

*Test 1*

```

POKRETANJE: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje
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 učitanom 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. Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.

*Primer 1*

```

POKRETANJE: ./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*

```

POKRETANJE: ./a.out studenti.txt

DATOTEKA STUDENTI.TXT JE PRAZNA

INTERAKCIJA SA PROGRAMOM:
3/2014 ne
235/2008 ne
41/2009 ne

```

[Rešenje 4.4]

\* **Zadatak 4.5** Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

- Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.
- Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

## 4 Dinamičke strukture podataka

---

Napisati program koji u datoteku rezultat.txt upisuje nađeni strogo rastući podniz.

```
Test 1                                Test 2                                Test 3
|| BROJEVI.TXT                        || DATOTEKA BROJEVI.TXT              || DATOTEKA BROJEVI.TXT JE PRAZNA
|| 43 12 15 16 4 2 8                 || NE POSTOJI.                      ||
|| IZLAZ:                             || IZLAZ ZA GREŠKE:                 || IZLAZ:
|| REZULTAT.TXT                       || Greska: Neuspesno otvaranje     || REZULTAT.TXT
|| 12 15 16                           || datoteke brojevi.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.

```
Test 1                                Test 2
|| POKRETANJE: ./a.out dat1.txt dat2.txt || POKRETANJE: ./a.out dat1.txt dat2.txt
|| DAT1.TXT                            || DAT1.TXT                          ||
|| 2 4 6 10 15                         || 2 4 6 10 15                      ||
|| DAT2.TXT                            || DATOTEKA DAT2.TXT NE POSTOJI.    ||
|| 5 6 11 12 14 16                    || IZLAZ ZA GREŠKE:                 ||
|| IZLAZ:                               || Greska: Neuspesno otvaranje     ||
|| [2, 4, 5, 6, 6, 10, 11, 12, 14, 15, 16] || datoteke dat2.txt.              ||

Test 3                                Test 4
|| POKRETANJE: ./a.out dat1.txt dat2.txt || POKRETANJE: ./a.out dat1.txt
|| DATOTEKA DAT1.TXT JE PRAZNA         || IZLAZ ZA GREŠKE:                 ||
|| DAT2.TXT                            || Greska: Program se poziva sa:   ||
|| 5 6 11 12 14 16                    || ./a.out dat1.txt dat2.txt!      ||
|| IZLAZ:                               ||
|| [5, 6, 11, 12, 14, 16]              ||
```

[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: *Koristiti testove 2 - 6 za zadatak 4.6.*

```

Test 1
POKRETANJE: ./a.out dat1.txt dat2.txt
DAT1.TXT
 2 4 6 10 15
DAT2.TXT
 5 6 11 12 14 16
IZLAZ:
 2 5 4 6 6 11 10 12 15 14 16

```

**Zadatak 4.8** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka (engl. *stack*) utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

```

Test 1
IZRAZ.TXT
 {[23 + 5344] * (24 - 234)} - 23
IZLAZ:
 Zagrade su ispravno uparene.

```

```

Test 2
IZRAZ.TXT
 {[23 + 5] * (9 * 2)} - {23}
IZLAZ:
 Zagrade su ispravno uparene.

```

```

Test 3
IZRAZ.TXT
 {[2 + 54] / (24 * 87)} + (234 + 23)
IZLAZ:
 Zagrade nisu ispravno uparene.

```

```

Test 4
IZRAZ.TXT
 {(2 - 14) / (23 + 11)} * (2 + 13)
IZLAZ:
 Zagrade nisu ispravno uparene.

```

```

Test 5
DATOTEKA IZRAZ.TXT JE PRAZNA
IZLAZ:
 Zagrade su ispravno uparene.

```

```

Test 6
DATOTEKA IZRAZ.TXT NE POSTOJI.
IZLAZ ZA GREŠKE:
 Greska: Neuspesno otvaranje
 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*

## 4 Dinamičke strukture podataka

problema koristiti stek implementiran preko liste čiji čvorovi sadrže HTML etikete.

### Test 1

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML
<html>
<head>
  <title>Primer</title>
</head>
<body>
</body>
```

```
IZLAZ:
Etikete nisu pravilno uparene
(etiketa <html> nije zatvorena)
```

### Test 2

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML
<head>
  <title>Primer</title>
</head>
<body>
</body>
</html>
```

```
IZLAZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html>
koja nije otvorena)
```

### Test 3

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML
<html>
<head>
  <title>Primer</title>
</head>
<body>
  <h1>Naslov</h1>
  Danas je lep i sunčan dan. <br>
  Sutra ce biti jos lepsi.
  <a link='http://www.math.rs'>Link</a>
</body>
</html>
```

```
IZLAZ:
Etikete su pravilno uparene!
```

### Test 4

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML
<html>
<head>
  <title>Primer</title>
</head>
<body>
</html>
```

```
IZLAZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html>, a
poslednja otvorena je <body>)
```

### Test 5

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA DATOTEKA.HTML NE POSTOJI.
```

```
IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje
datoteke datoteka.html.
```

### Test 6

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML JE PRAZNA
```

```
IZLAZ:
Etikete su pravilno uparene!
```

[Rešenje 4.9]

**Zadatak 4.10** Napisati program koji pomaže službeniku u radu na šalteru. Službenik najpre evidentira sve korisničke *JMBG* brojeve (niske koje sadrže po 13 karaktera) i zahteve (niske koja sadrže 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 vratate 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 (engl. queue) implementiran korišćenjem listi.*

## 4 Dinamičke strukture podataka

---

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Sluzbenik evidentira korisnicke zahteve:
Novi zahtev [CTRL+D za kraj]
JMBG: 1234567890123
Opis problema: Otvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 2345678901234
Opis problema: Podizanje novca

Novi zahtev [CTRL+D za kraj]
JMBG: 3456789012345
Opis problema: Reklamacija

Novi zahtev [CTRL+D za kraj]
JMBG:

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 2345678901234
i zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

IZVESTAJ.TXT
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 3456789012345 Zahtev: Reklamacija
JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

[Rešenje 4.10]

**Zadatak 4.11** Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novom definicijom cvore i funkcijama.

(a) Definisati strukturu Cvor kojom se predstavlja čvor liste, a koja sadrži ceo

broj vrednost, pokazivače na sledeći i prethodni čvor liste.

- (b) 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`.
- (c) 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: *Koristiti 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 kostimima 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 stavlja 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 povezanu kružnu listu.*

Test 1

```

|| ULAZ:
|| 5 3
||
|| IZLAZ:
|| 3 1 5 2 4

```

Test 2

```

|| ULAZ:
|| 8 4
||
|| IZLAZ:
|| 4 8 5 2 1 3 7 6

```

Test 3

```

|| ULAZ:
|| 3 8
||
|| IZLAZ ZA GREŠKE:
|| Greska: n mora biti uvek
|| vece 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. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

```
Test 1          Test 2          Test 3
|||            |||            |||
|| ULAZ:      || ULAZ:      || ULAZ:
|| 5 3        || 8 4        || 5 8
||           ||           ||
|| IZLAZ:     || IZLAZ:     || IZLAZ ZA GREŠKE:
|| 3 5 4 2 1  || 4 8 5 7 6 3 2 1 || Greska: n mora biti uvek
||           ||           || vece od k, a 5 < 8!
```

## 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 alokira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.
- Napisati funkciju `int dodaj_u_stablo(Cvor ** adresa_korena, int broj)` koja u stablo na koje pokazuje argument `adresa_korena` dodaje ceo broj `broj`. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.
- Napisati funkciju `Cvor *pretrazi_stablo(Cvor * koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili `NULL` ukoliko takav čvor ne postoji.

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

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

*Primer 1*

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

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuce stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

## 4 Dinamičke strukture podataka

**Zadatak 4.15** Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati na standardni izlaz za greške poruku *Nedostaje ime ulazne datoteke!*. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

```
Test 1
| POKRETANJE: ./a.out test.txt
| TEST.TXT
| Sunce utorak raCunar SUNCE programiranje
| jabuka PROGRAMiranje sunCE JABUka
| IZLAZ:
| jabuka: 2
| programiranje: 2
| racunar: 1
| sunce: 3
| utorak: 1
| Najcesca rec: sunce (pojavljuje se 3 puta)

Test 2
| POKRETANJE: ./a.out suma.txt
| SUMA.TXT
| lipa zova hrast ZOVA breza LIPA
| IZLAZ:
| breza: 1
| hrast: 1
| lipa: 2
| zova: 2
| Najcesca rec: lipa
| (pojavljuje se 2 puta)

Test 3
| POKRETANJE: ./a.out ulaz.txt
| DATOTEKA ULAZ.TXT NE POSTOJI
| IZLAZ ZA GREŠKE:
| Greska: Neuspesno otvaranje datoteke ulaz.txt.

Test 4
| POKRETANJE: ./a.out
| IZLAZ ZA GREŠKE:
| Greska: 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 međusobno razdvojeni blanko znakom, na primer *Milos Peric* 064/123 – 4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči *KRAJ*, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

*Primer 1*

```

IMENIK.TXT
Milos 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: Milos Peric
Broj je: 011/3240-987
Unesite ime i prezime: Marko Markovic
Broj nije u imeniku!
Unesite ime i prezime: KRAJ

```

*Primer 2*

```

DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik1.txt
IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje datoteke
imenik1.txt.

```

[Rešenje 4.16]

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

*Test 1*

```

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

IZLAZ:
1. Maja Agic 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9

```

*Test 2*

```

DATOTEKA PRIJEMNI.TXT NE POSTOJI

IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje datoteke
prijemni.txt.

```

[Rešenje 4.17]

\* **Zadatak 4.18** Napisati program koji implementira podsetnik za rođendan. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije. Svaka linija datoteke je formata *Ime Prezime DD.MM.* i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. 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 datumima - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i *DD.MM.* formata. Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

### Primer 1

```
POKRETANJE: ./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

```
POKRETANJE: ./a.out rodjendani.txt

DATOTEKA RODJENDANI.TXT NE POSTOJI

IZLAZ ZA GREŠKE:
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. Napisati 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 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
|| Greška: Pogresna dimenzija niza!

```

[Rešenje 4.21]

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

(a) Napisati funkciju koja izračunava broj čvorova stabla.

## 4 Dinamičke strukture podataka

---

- (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

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

### Test 2

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

[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.*

Test 1	Test 2	Test 3
<pre> ULAZ:  10 5 15 3 2 4 30 12 14 13  IZLAZ: 0.nivo: 10 1.nivo: 5 15 2.nivo: 3 12 30 3.nivo: 2 4 14 4.nivo: 13           </pre>	<pre> ULAZ:  6 11 8 3 -2  IZLAZ: 0.nivo: 6 1.nivo: 3 11 2.nivo: -2 8           </pre>	<pre> ULAZ:  24 53 61 9 7 55 20 16  IZLAZ: 0.nivo: 24 1.nivo: 9 53 2.nivo: 7 20 61 3.nivo: 16 55           </pre>

[Rešenje 4.23]

\* **Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Primer 1	Primer 2
<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 14 30 1 0 Stabla su slicna kao u ogledalu.           </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 20 15 0 Stabla nisu slicna kao u ogledalu.           </pre>

**Zadatak 4.25** AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor * koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i 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.*



## 4 Dinamičke strukture podataka

Test 1

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

Test 2

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

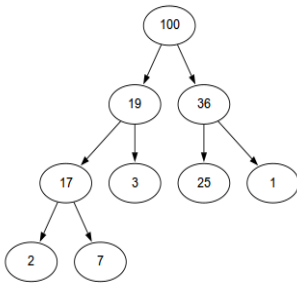
[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int hip(Cvor * koren)` koja proverava da li je dato binarno stablo celih brojeva hip. Napisati zatim i program koji kreira stablo zadato slikom 4.1, poziva funkciju `hip` i ispisuje rezultat na standardni izlaz. NAPOMENA: Za alokaciju i oslobađanje memorije koristiti funkcije `napravi_cvor` i `oslobodi_stablo` iz zadatka 4.14.

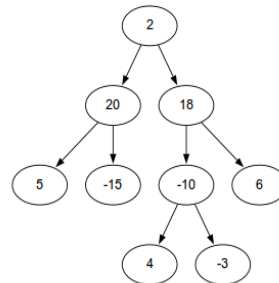
Test 1

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

[Rešenje 4.26]



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

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

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

*Test 1*

```

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

```

## 4.3 Rešenja

### Rešenje 4.1

*lista.h*

```

1 #ifndef _LISTA_H_
2 #define _LISTA_H_
3
4 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
5    podatak vrednost i pokazivac na sledeci cvor liste */
6 typedef struct cvor {
7     int vrednost;
8     struct cvor *sledeci;
9 } Cvor;
10
11 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na
12    broj, dok pokazivac na sledeci cvor postavlja na NULL. Vraca
13    pokazivac na novokreirani cvor ili NULL ako alokacija nije bila
14    uspesna. */
15 Cvor *napravi_cvor(int broj);
16
17 /* Funkcija oslobadja dinamicnu memoriju zauzetu za cvorove liste
18    ciji se pokazivac glava nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave);
20

```

## 4 Dinamičke strukture podataka

```
22  /* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
    greske pri alokaciji memorije, inace vraca 0. */
    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. */
    Cvor *pronadji_poslednji(Cvor * glava);
28
    /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo
    greske pri alokaciji memorije, inace vraca 0. */
    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. */
    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. */
    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. */
    int dodaj_sortirano(Cvor ** adresa_glave, int broj);
44
    /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
    broju. Vraca pokazivac na cvor liste u kome je sadržan traženi
    broj ili NULL u slucaju da takav cvor ne postoji u listi. */
    Cvor *pretrazi_listu(Cvor * glava, int broj);
50
    /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
    broju. Vraca pokazivac na cvor liste u kome je sadržan traženi
    broj ili NULL ako da takav cvor ne postoji. U pretrazi oslanja
    se na cinjenicu se pretrazuje neopadajuće sortirana lista. */
    Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
56
    /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
    Azurira pokazivac na glavu liste, koji može biti promenjen u
    slucaju da se obrise stara glava. */
    void obrisi_cvor(Cvor ** adresa_glave, int broj);
60
    /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na cinjenicu da je prosledjena lista sortirana
    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
    promenjen ukoliko se obrise stara glava liste. */
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
66
    /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
    kraju liste, razdvojene zapetama i uokvirene zagradama. */
    void ispisi_listu(Cvor * glava);
70
72 #endif
```

*lista.c*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  Cvor *napravi_cvor(int broj)
6  {
7      /* Alokacija memorije za novi cvor uz proveru uspesnosti
8         alokacije. */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;
12
13     /* Inicijalizacija polja strukture */
14     novi->vrednost = broj;
15     novi->sledeci = NULL;
16
17     /* Vraca se adresa novog cvora */
18     return novi;
19 }
20
21 void oslobodi_listu(Cvor ** adresa_glave)
22 {
23     Cvor *pomocni = NULL;
24
25     /* Ako lista nije prazna, onda treba osloboditi memoriju */
26     while (*adresa_glave != NULL) {
27         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
28            osloboditi cvor koji predstavlja glavu liste */
29         pomocni = (*adresa_glave)->sledeci;
30         free(*adresa_glave);
31
32         /* Sledeci cvor je nova glava liste */
33         *adresa_glave = pomocni;
34     }
35 }
36
37 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
38 {
39     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
40     Cvor *novi = napravi_cvor(broj);
41     if (novi == NULL)
42         return 1;
43
44     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
45     novi->sledeci = *adresa_glave;
46     *adresa_glave = novi;
47
48     /* Vraca se indikator uspesnog dodavanja */
49     return 0;
50 }
```

## 4 Dinamičke strukture podataka

```
50 }
52 Cvor *pronadji_poslednji(Cvor * glava)
53 {
54     /* U praznoj listi nema cvorova pa se vraća NULL */
55     if (glava == NULL)
56         return NULL;
57
58     /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
59     glava se pomera na sledeći cvor. Nakon izlaska iz petlje,
60     glava će pokazivati na cvor liste koji nema sledbenika, tj. na
61     poslednji cvor liste, pa se vraća vrednost pokazivaca glava.
62     Pokazivac glava je argument funkcije i njegove promene neće se
63     odraziti na vrednost pokazivaca glava u pozivajućoj funkciji. */
64     while (glava->sledeci != NULL)
65         glava = glava->sledeci;
66
67     return glava;
68 }
69
70 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
71 {
72     /* Kreira se novi cvor i proverava se uspešnost kreiranja */
73     Cvor *novi = napravi_cvor(broj);
74     if (novi == NULL)
75         return 1;
76
77     /* Ako je lista prazna */
78     if (*adresa_glave == NULL) {
79         /* Glava nove liste je upravo novi cvor */
80         *adresa_glave = novi;
81     } else {
82         /* Ako lista nije prazna, pronalazi se poslednji cvor i novi
83         cvor se dodaje na kraj liste kao sledbenik poslednjeg */
84         Cvor *poslednji = pronadji_poslednji(*adresa_glave);
85         poslednji->sledeci = novi;
86     }
87
88     /* Vraća se indikator uspešnog dodavanja */
89     return 0;
90 }
91
92 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
93 {
94     /* U praznoj listi nema takvog mesta i vraća se NULL */
95     if (glava == NULL)
96         return NULL;
97
98     /* Pokazivac glava se pomera na sledeći cvor sve dok ne bude
99     pokazivao na cvor čiji sledeći ili ne postoji ili ima vrednost
100     veću ili jednaku vrednosti novog cvora. */
101     /* Zbog izračunavanja izraza u C-u prvi deo konjunkcije mora biti
```

```
102     provera da li se doslo do poslednjeg cvora liste pre nego sto
103     se proveru 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
109     poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci
110     ima vrednost vecu od broj. */
111     return glava;
112 }

113
114 int dodaj_iza(Cvor * tekuci, int broj)
115 {
116     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
117     Cvor *novi = napravi_cvor(broj);
118     if (novi == NULL)
119         return 1;
120
121     /* Novi cvor se dodaje iza tekuceg cvora. */
122     novi->sledeci = tekuci->sledeci;
123     tekuci->sledeci = novi;
124
125     /* Vraca se indikator uspesnog dodavanja */
126     return 0;
127 }

128
129 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
130 {
131     /* Ako je lista prazna */
132     if (*adresa_glave == NULL) {
133         /* Glava nove liste je novi cvor */
134         /* Kreiranje novog cvora uz proveru uspesnost kreiranja */
135         Cvor *novi = napravi_cvor(broj);
136         if (novi == NULL)
137             return 1;
138
139         *adresa_glave = novi;
140
141         /* Vraca se indikator uspesnog dodavanja */
142         return 0;
143     }
144
145     /* Inace, ako je broj manji ili jednak vrednosti u glavi liste,
146     onda ga treba dodati na pocetak liste. */
147     if ((*adresa_glave)->vrednost >= broj) {
148         return dodaj_na_pocetak_liste(adresa_glave, broj);
149     }
150
151     /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
152     tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
153     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
```

## 4 Dinamičke strukture podataka

---

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

```
206     }
207
208     /* Ako je nakon ovog brisanja lista prazna, vraća se iz funkcije */
209     if (*adresa_glave == NULL)
210         return;
211
212     /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
213        pokazuje na cvor čija je vrednost različita od traženog broja.
214        Isto važi i za sve cvorove levo od tekućeg. Poredi se vrednost
215        sledećeg cvora (ako postoji) sa traženim brojem. Cvor se briše
216        ako je jednak, a ako je različit, prelazi se na sledeći. Ovaj
217        postupak se ponavlja dok se ne dođe do poslednjeg cvora. */
218     tekuci = *adresa_glave;
219     while (tekuci->sledeci != NULL)
220     {
221         if (tekuci->sledeci->vrednost == broj) {
222             /* tekuci->sledeci treba obrisati, zbog toga se njegova
223                adresa prvo čuva u promenljivoj pomocni. */
224             pomocni = tekuci->sledeci;
225             /* Tekućem se preusmerava pokazivač sledeći, preskakanjem
226                njegovog trenutnog sledećeg. Njegov novi sledeći će biti
227                sledeći od cvora koji se briše. */
228             tekuci->sledeci = pomocni->sledeci;
229             /* Sada treba osloboditi cvor sa vrednošću broj. */
230             free(pomocni);
231         } else {
232             /* Inače, ne treba brisati sledećeg od tekućeg i pokazivač se
233                pomera na sledeći. */
234             tekuci = tekuci->sledeci;
235         }
236     }
237     return;
238 }
239
240 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
241 {
242     Cvor *tekuci = NULL;
243     Cvor *pomocni = NULL;
244
245     /* Sa početka liste se brišu svi cvorovi koji su jednaki datom
246        broju i azurira se pokazivač na glavu liste. */
247     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
248     {
249         /* Adresu repa liste treba sačuvati pre oslobađanja cvora na
250            adresi adresa_glave. */
251         pomocni = (*adresa_glave)->sledeci;
252         free(*adresa_glave);
253         *adresa_glave = pomocni;
254     }
255
256     /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
257        prekida. Isto se radi i ukoliko glava liste sadrži vrednost
258        koja je veća od broja, jer kako je lista sortirana rastuće
259        nema potrebe broj tražiti u repu liste. */
```



## 4 Dinamičke strukture podataka

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

*main\_a.c*

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

```

11  /* Testiranje funkcije za dodavanja novog broja na pocetak liste */
13  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
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 osloboditi pre napustanja programa. */
18      if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
19          fprintf(stderr,
20                 "Greska: Neuspesna alokacija memorije za cvor.\n");
21          oslobodi_listu(&glava);
22          exit(EXIT_FAILURE);
23      }
24      printf("\tLista: ");
25      ispisi_listu(glava);
26  }

27  /* Testiranje funkcije za pretragu liste */
29  printf("\nUnesite broj koji se trazi: ");
30  scanf("%d", &broj);

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

36  /* Oslobadja se memorija zauzeta listom */
37  oslobodi_listu(&glava);

40  exit(EXIT_SUCCESS);
41  }

```

*main\_b.c*

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  int main()
6  {
7      /* Lista je prazna na pocetku */
8      Cvor *glava = NULL;
9      int broj;
10
11     /* Testira se funkcija za dodavanja novog broja na kraj liste */
12     printf("Unesite brojeve: (za kraj CTRL+D)\n");
13     while (scanf("%d", &broj) > 0) {
14         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
15            memorije za nov cvor. Memoriju alociranu za cvorove liste
16            treba osloboditi pre napustanja programa. */

```

## 4 Dinamičke strukture podataka

```
18     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
19         fprintf(stderr,
20             "Greska: Neuspesna alokacija memorije za cvor.\n");
21         oslobodi_listu(&glava);
22         exit(EXIT_FAILURE);
23     }
24     printf("\tLista: ");
25     ispisi_listu(glava);
26 }
27
28 /* Testira se funkcije kojom se brise cvor liste */
29 printf("\nUnesite broj koji se brise: ");
30 scanf("%d", &broj);
31
32 /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
33 obrisi_cvor(&glava, broj);
34
35 printf("Lista nakon brisanja: ");
36 ispisi_listu(glava);
37
38 /* Oslobadja se memorija zauzeta listom */
39 oslobodi_listu(&glava);
40
41 exit(EXIT_SUCCESS);
42 }
```

main.c.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 int main()
6 {
7     /* Lista je prazna na pocetku */
8     Cvor *glava = NULL;
9     Cvor *trazeni = NULL;
10    int broj;
11
12    /* Testira se funkcija za dodavanje vrednosti u listu tako da
13       bude uredjena neopadajuće */
14    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
15    while (scanf("%d", &broj) > 0) {
16        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17           memorije za nov cvor. Memoriju alociranu za cvorove liste
18           treba osloboditi pre napustanja programa. */
19        if (dodaj_sortirano(&glava, broj) == 1) {
20            fprintf(stderr,
21                "Greska: Neuspesna alokacija memorije za cvor.\n");
22            oslobodi_listu(&glava);
23            exit(EXIT_FAILURE);
24        }
25    }
26 }
```

```

    }
25     printf("\tLista: ");
    ispisi_listu(glava);
27 }

29 /* Testira se funkcija za pretragu liste */
    printf("\nUnesite broj koji se traži: ");
31     scanf("%d", &broj);

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

39 /* Testira se funkcija kojom se brise cvor liste */
    printf("\nUnesite broj koji se brise: ");
41     scanf("%d", &broj);

43 /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
    obrisi_cvor_sortirane_liste(&glava, broj);

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

47

49 /* Oslobadja se memorija zauzeta listom */
    oslobodi_listu(&glava);

51     exit(EXIT_SUCCESS);
53 }

```

## Rešenje 4.2

*lista.h*

```

1  #ifndef _LISTA_H_
    #define _LISTA_H_
3
    /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
5     podatak vrednost i pokazivac na sledeci cvor liste. */
    typedef struct cvor {
7         int vrednost;
        struct cvor *sledeci;
9     } Cvor;

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

```

## 4 Dinamičke strukture podataka

```
17 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
18    ciji se pokazivac glava nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave);

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);

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

29 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova
30    lista ostane sortirana. Vraca 1 ukoliko je bilo greske pri
31    alokaciji 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
35    broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
36    broj ili NULL u slucaju da takav cvor ne postoji u listi. */
37 Cvor *pretrazi_listu(Cvor * glava, int broj);

39 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
40    broju. U pretrazi oslanja se na cinjenicu da je lista koja se
41    pretrazuje neopadajuce sortirana. Vraca pokazivac na cvor liste
42    u kome je sadrzan trazeni broj ili NULL ako takav cvor ne
43    postoji. */
44 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
45
46 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
47    Azurira pokazivac na glavu liste, koji moze biti promenjen u
48    slucaju da se obrise stara glava liste. */
49 void obrisi_cvor(Cvor ** adresa_glave, int broj);

51 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
52    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
53    neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
54    promenjen ukoliko se obrise stara glava liste. */
55 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

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

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

65 #endif
```

*lista.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 Cvor *napravi_cvor(int broj)
6 {
7     /* Alokacija memorije za novi cvor uz proveru uspesnosti */
8     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9     if (novi == NULL)
10         return NULL;
11
12     /* Inicijalizacija polja strukture */
13     novi->vrednost = broj;
14     novi->sledeci = NULL;
15
16     /* Vraca se adresa novog cvora */
17     return novi;
18 }
19
20 void oslobodi_listu(Cvor ** adresa_glave)
21 {
22     /* Ako je lista vec prazna */
23     if (*adresa_glave == NULL)
24         return;
25
26     /* Ako lista nije prazna, treba osloboditi memoriju. Treba
27        osloboditi rep, pre oslobadjanja memorije za glavu liste. */
28     oslobodi_listu(&(*adresa_glave)->sledeci);
29     /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
30        glava u pozivajucoj funkciji tako da odgovara praznoj listi */
31     free(*adresa_glave);
32     *adresa_glave = NULL;
33 }
34
35 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
36 {
37     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
38     Cvor *novi = napravi_cvor(broj);
39     if (novi == NULL)
40         return 1;
41
42     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
43     novi->sledeci = *adresa_glave;
44     *adresa_glave = novi;
45
46     /* Vraca se indikator uspesnog dodavanja */
47     return 0;
48 }
49
```

## 4 Dinamičke strukture podataka

```
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
51 {
    /* Ako je lista prazna */
53     if (*adresa_glave == NULL) {
        /* Novi cvor postaje glava liste */
        Cvor *novi = napravi_cvor(broj);
57         /* Ako je bilo greske pri kreiranju novog cvora, vraca se 1 */
        if (novi == NULL)
59             return 1;
        /* Azuriranjem vrednosti na koju pokazuje adresa_glave, ujedno
61         se azurira i pokazivacka promenljiva u pozivajucoj funkciji */
63         *adresa_glave = novi;
        /* Vraca se indikator uspesnog dodavanja */
65         return 0;
67     }
69     /* Ako lista nije prazna, broj se dodaje u rep liste. */
    /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
71     novi broj se dodaje u rep liste u rekurzivnom pozivu.
    Informaciju o uspesnosti alokacije u rekurzivnom pozivu
73     funkcija prosledjuje visem rekurzivnom pozivu koji tu
    informaciju vraca u rekurzivni poziv iznad, sve dok se ne
75     vrati u main. Tek je iz main funkcije moguće pristupiti pravom
    pocetku liste i osloboditi je celu, ako ima potrebe. Ako je
77     funkcija vratila 0, onda nije bilo greske. */
    return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
79 }

int dodaj_sortirano(Cvor ** adresa_glave, int broj)
81 {
    /* Ako je lista prazna */
83     if (*adresa_glave == NULL) {
        /* Novi cvor postaje glava liste */
85         Cvor *novi = napravi_cvor(broj);
87
        /* Ako je bilo greske pri kreiranju novog cvora, vraca se 1 */
89         if (novi == NULL)
91             return 1;
93         /* Azurira se glava liste */
        *adresa_glave = novi;
95
        /* Vraca se indikator uspesnog dodavanja */
97         return 0;
99     }
101     /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
    glavi liste, onda se dodaje na pocetak liste */
```

```
103     if ((*adresa_glave)->vrednost >= broj)
        return dodaj_na_pocetak_liste(adresa_glave, broj);
105     /* Inace, broj treba dodati u rep liste, tako da rep i sa novim
        cvorom bude sortirana lista. */
107     return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
    }
109
    Cvor *pretrazi_listu(Cvor * glava, int broj)
111    {
        /* U praznoj listi nema vrednosti */
113     if (glava == NULL)
        return NULL;
115
        /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
117     if (glava->vrednost == broj)
        return glava;
119
        /* Inace, pretraga se nastavlja u repu liste */
121     return pretrazi_listu(glava->sledeci, broj);
    }
123
    Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
125    {
        /* Trazenog broja nema ako je lista prazna ili broj manji od
        vrednosti u glavi liste, jer je lista neopadajuce sortirana */
127     if (glava == NULL || glava->vrednost > broj)
        return NULL;
129
        /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
131     if (glava->vrednost == broj)
        return glava;
133
        /* Inace, pretraga se nastavlja u repu liste */
135     return pretrazi_listu(glava->sledeci, broj);
    }
137
    void obrisi_cvor(Cvor ** adresa_glave, int broj)
139    {
        /* U praznoj listi nema cvorova za brisanje. */
141     if (*adresa_glave == NULL)
        return;
143
        /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
        obrisi_cvor(&(*adresa_glave)->sledeci, broj);
147
        /* Preostaje provera da li glavu liste treba obrisati */
149     if ((*adresa_glave)->vrednost == broj) {
        /* Pomocni pokazuje na cvor koji treba da se obrise */
        Cvor *pomocni = *adresa_glave;
151     /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi
        i brise se cvor koji je bio glava liste. */
153     }
```



## 4 Dinamičke strukture podataka

```
    *adresa_glave = (*adresa_glave)->sledeci;
155     free(pomocni);
    }
157 }

159 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
161     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
        veca od broja, kako je lista sortirana rastuce nema potrebe
163     broj traziti u repu liste i zato se funkcija prekida */
    if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
165         return;

167     /* Brisu se cvorovi iz repa koji imaju vrednost broj */
    obrisi_cvor(&(*adresa_glave)->sledeci, broj);
169

171     /* Preostaje provera da li glavu liste treba obrisati */
    if ((*adresa_glave)->vrednost == broj) {
173         /* Pomocni pokazuje na cvor koji treba da se obrise */
        Cvor *pomocni = *adresa_glave;
175         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi
            i brise se cvor koji je bio glava liste */
        *adresa_glave = (*adresa_glave)->sledeci;
177         free(pomocni);
    }
179 }

181 void ispisi_vrednosti(Cvor * glava)
{
183     /* Prazna lista */
    if (glava == NULL)
185         return;

187     /* Ispisuje se vrednost u glavi liste */
    printf("%d", glava->vrednost);
189

191     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
        se poziva ista funkcija za ispis ostalih. */
    if (glava->sledeci != NULL) {
193         printf(", ");
        ispisi_vrednosti(glava->sledeci);
195     }
}

197 void ispisi_listu(Cvor * glava)
199 {
201     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
203     na glavu liste iz pozivajuće funkcije. Ona ispisuje samo
        zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
205     rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
        elemente razdvojene zapedom i razmakom. */
```

```

207     putchar('[');
        ispisi_vrednosti(glava);
        printf("]\n");
209 }

```

### Rešenje 4.3

```

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

5  #define MAX_DUZINA 20

7  /* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
   broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
9  typedef struct _Cvor {
   char etiketa[20];
11  unsigned broj_pojavljivanja;
   struct _Cvor *sledeci;
13 } Cvor;

15 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
   ili NULL ako alokacija nije uspesno izvršena */
17 Cvor *napravi_cvor(unsigned br, char *etiketa)
   {
19     /* Alokacija memorije za cvor uz proveru uspesnosti alokacije */
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
21     if (novi == NULL)
        return NULL;

23     /* Inicijalizacija polja strukture */
25     novi->broj_pojavljivanja = br;
   strcpy(novi->etiketa, etiketa);
27     novi->sledeci = NULL;

29     /* Vraca se adresa novog cvora */
   return novi;
31 }

33 /* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
   void oslobodi_listu(Cvor ** adresa_glave)
35 {
   Cvor *pomocni = NULL;
37
   /* Sve dok lista ni bude prazna, brise se tekuca glava liste i
   azurira se vrednost glave liste */
39   while (*adresa_glave != NULL) {
41     pomocni = (*adresa_glave)->sledeci;
   free(*adresa_glave);
43     *adresa_glave = pomocni;
   }

```

## 4 Dinamičke strukture podataka

---

```
45  /* Pokazivac glava u main funkciji, na adresi adresa_glave, bice
46     postavljen na NULL vrednost po izlasku iz petlje. */
47  }
48
49  /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
50     do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
51  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
52                             char *etiketa)
53  {
54     /* Kreira se novi cvor i proverava se uspesnost alokacije */
55     Cvor *novi = napravi_cvor(br, etiketa);
56     if (novi == NULL)
57         return 1;
58
59     /* Dodaje se novi cvor na pocetak liste */
60     novi->sledeci = *adresa_glave;
61     *adresa_glave = novi;
62
63     /* Vraca se indikator uspesnog dodavanja */
64     return 0;
65  }
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
73     /* Obilazi se cvor po cvor liste */
74     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
75         /* Ako tekuci cvor sadrzi trazenu etiketu, vraca se njegova
76            vrednost */
77         if (strcmp(tekuci->etiketa, etiketa) == 0)
78             return tekuci;
79
80     /* Cvor nije pronadjen */
81     return NULL;
82  }
83
84  /* Funkcija ispisuje sadrzaj liste */
85  void ispisi_listu(Cvor * glava)
86  {
87     /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete
88        i broj njihovog pojavljivanja u HTML datoteci. */
89     for (; glava != NULL; glava = glava->sledeci)
90         printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
91  }
92
93  int main(int argc, char **argv)
94  {
95     /* Proverava se da li je program pozvan sa ispravnim brojem
96        argumenata komandne linije. */
```

```
97  if (argc != 2) {
98      fprintf(stderr,
99          "Greska: Program se poziva sa: ./a.out datoteka.html\n");
100      exit(EXIT_FAILURE);
101  }
102
103  /* Priprema datoteke za citanje */
104  FILE *in = NULL;
105  in = fopen(argv[1], "r");
106  if (in == NULL) {
107      fprintf(stderr,
108          "Greska: Neuspesno otvaranje datoteke %s!\n", argv[1]);
109      exit(EXIT_FAILURE);
110  }
111
112  char c;
113  int i = 0;
114  char procitana[MAX_DUZINA];
115  Cvor *glava = NULL;
116  Cvor *trazeni = NULL;
117
118  /* Cita se datoteka, karakter po karakter, dok se ne procita
119     karakter za kraj sadrzaja datoteke. */
120  while ((c = fgetc(in)) != EOF) {
121      /* Proverava se da li se pocinje sa citanjem nove etikete */
122      if (c == '<') {
123          /* Proverava se da li se cita zatvarajuca etiketa */
124          if ((c = fgetc(in)) == '/') {
125              i = 0;
126              while ((c = fgetc(in)) != '>')
127                  procitana[i++] = c;
128          }
129          /* Cita se otvarajuca etiketa */
130          else {
131              i = 0;
132              procitana[i++] = c;
133              while ((c = fgetc(in)) != ' ' && c != '>')
134                  procitana[i++] = c;
135          }
136          procitana[i] = '\0';
137
138          /* Trazi se procitana etiketa medju postojećim cvorovima
139             liste. Ako ne postoji, dodaje se novi cvor za ucitanu
140             etiketu sa brojem pojavljivanja 1. Inace se uvecava broj
141             pojavljivanja etikete. */
142          trazeni = pretrazi_listu(glava, procitana);
143          if (trazeni == NULL) {
144              if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
145                  fprintf(stderr,
146                      "Greska: Neuspesna alokacija memorije za nov cvor\n
147                      ");
148                  oslobodi_listu(&glava);

```

## 4 Dinamičke strukture podataka

```
        exit(EXIT_FAILURE);
149     }
        } else
151     trazeni->broj_pojavljivanja++;
    }
153 }

155 /* Zatvara se datoteka */
fclose(in);

157 /* Ispisuje se sadržaj cvorova liste */
159 ispisi_listu(glava);

161 /* Oslobadja se memorija zauzeta listom */
oslobodi_listu(&glava);
163

165 exit(EXIT_SUCCESS);
}
```

### Rešenje 4.4

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

4
#define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21

8 /* Struktura kojom se predstavlja cvor liste koji sadrži podatke o
   studentu */
10 typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
12     char ime[MAX_IME_PREZIME];
    char prezime[MAX_IME_PREZIME];
14     struct _Cvor *sledeci;
} Cvor;
16

18 /* Funkcija kreira i inicijalizuje cvor liste i vraća pokazivač na
   novi cvor ili NULL ukoliko je došlo do greške */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
    /* Alokacija memorije za cvor uz proveru uspešnosti alokacije */
22     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
24         return NULL;

26     /* Inicijalizacija polja strukture */
    strcpy(novi->broj_indeksa, broj_indeksa);
28     strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
30     novi->sledeci = NULL;
```

```
32  /* Vraca se adresa novog cvora */
    return novi;
34 }

36 /* Funkcija oslobadja memoriju zauzetu cvorovima liste */
void oslobodi_listu(Cvor ** adresa_glave)
38 {
    /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
40  if (*adresa_glave == NULL)
        return;

42  /* Rekurzivnim pozivom se oslobadja rep liste */
44  oslobodi_listu(&(*adresa_glave)->sledeci);

46  /* Potom se oslobadja i glava liste */
    free(*adresa_glave);

48  /* Lista se proglašava praznom */
50  *adresa_glave = NULL;
}

52 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
54  do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
56  char *ime, char *prezime)
{
58  /* Kreira se novi cvor i proverava se uspesnost alokacije */
    Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
60  if (novi == NULL)
        return 1;

62  /* Dodaje se novi cvor na pocetak liste */
64  novi->sledeci = *adresa_glave;
    *adresa_glave = novi;

66  /* Vraca se indikator uspesnog dodavanja */
68  return 0;
}

70 /* Funkcija ispisuje sadrzaj cvorova liste. */
72 void ispisi_listu(Cvor * glava)
{
74  /* Pocevsi od glave liste */
    for (; glava != NULL; glava = glava->sledeci)
76  printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
        glava->prezime);
78 }

80 /* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj
    indeksa. U suprotnom funkcija vraca NULL */
82 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
```

## 4 Dinamičke strukture podataka

```
{
84  /* Ako je lista prazna, ne postoji trazeni cvor */
    if (glava == NULL)
86      return NULL;

88  /* Poredi se trazeni broj indeksa sa indeksom u glavi liste */
    if (!strcmp(glava->broj_indeksa, broj_indeksa))
90      return glava;

92  /* Ukoliko u glavi liste nije trazeni indeks, pretraga se
    nastavlja u repu liste */
94  return pretrazi_listu(glava->sledeci, broj_indeksa);
}

96 int main(int argc, char **argv)
98 {
    /* Argumenti komandne linije su neophodni jer se iz komandne
100     linije dobija ime datoteke sa informacijama o studentima */
    if (argc != 2) {
102        fprintf(stderr,
            "Greska: Program se poziva sa: ./a.out ime_datoteke\n");
104        exit(EXIT_FAILURE);
    }

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

116    /* Deklaracije pomocnih promenljiva za citanje vrednosti koje
    treba smestiti u listu */
118    char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
    char broj_indeksa[MAX_INDEKS];
120    Cvor *glava = NULL;
    Cvor *trazeni = NULL;

122    /* Ucitavanje vrednosti u listu */
124    while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
        if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
126            fprintf(stderr,
                "Greska: Neuspesna alokacija memorije za nov cvor\n");
128            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
130        }

132    /* Datoteka vise nije potrebna i zatvara se. */
    fclose(in);
134
```

```

136 /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
while (scanf("%s", broj_indeksa) != EOF) {
    trazeni = pretrazi_listu(glava, broj_indeksa);
138     if (trazeni == NULL)
        printf("ne\n");
140     else
        printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
142 }

144 /* Oslobadja se memorija zauzeta za cvorove liste. */
oslobodi_listu(&glava);

146
148 exit(EXIT_SUCCESS);
}

```

### Rešenje 4.6

NAPOMENA: *Rešenje koristi biblioteku za rad sa listama iz zadatka 4.1.*

```

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

5  /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave
   * nalaze na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
   * pokazivaca postojećih cvorova listi */
7  Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9  {
11     /* Pokazivaci na pocetne cvorove listi koje se prevezuju */
    Cvor *lista1 = *adresa_glave_1;
    Cvor *lista2 = *adresa_glave_2;

13

15     /* Pokazivac na pocetni cvor rezultujuće liste */
    Cvor *rezultujuca = NULL;
    Cvor *tekuci = NULL;

17

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

21

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

25

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

29

31     /* Odredjuje se prvi cvor rezultujuće liste - to je ili prvi cvor
   * liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
   * koji sadrzi manju vrednost */

```



## 4 Dinamičke strukture podataka

```
33  if (lista1->vrednost < lista2->vrednost) {
34      rezultujuca = lista1;
35      lista1 = lista1->sledeci;
36  } else {
37      rezultujuca = lista2;
38      lista2 = lista2->sledeci;
39  }
40  /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste,
41     ne sme joj se menjati vrednost. Zato se koristi pokazivac
42     tekuci koji sadrzi adresu trenutnog cvora rezultujuce liste */
43  tekuci = rezultujuca;
44
45  /* U svakoj iteraciji petlje rezultujucoj listi se dodaje novi
46     cvor tako da bude uredjena neopadajuce. Pokazivac na listu iz
47     koje se uzima cvor se azurira tako da pokazuje na sledeci. */
48  while (lista1 != NULL && lista2 != NULL) {
49      if (lista1->vrednost < lista2->vrednost) {
50          tekuci->sledeci = lista1;
51          lista1 = lista1->sledeci;
52      } else {
53          tekuci->sledeci = lista2;
54          lista2 = lista2->sledeci;
55      }
56      tekuci = tekuci->sledeci;
57  }
58
59  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
60     rezultujucoj listu treba nadovezati ostatak druge liste */
61  if (lista1 == NULL)
62      tekuci->sledeci = lista2;
63  else
64      /* U suprotnom treba nadovezati ostatak prve liste */
65      tekuci->sledeci = lista1;
66
67  /* Preko adresa glava polaznih listi vrednosti pokazivaca u
68     pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
69     prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
70     se moze doci prateci pokazivace iz glave rezultujuce liste, tako
71     da stare pokazivace treba postaviti na NULL. */
72  *adresa_glave_1 = NULL;
73  *adresa_glave_2 = NULL;
74
75  return rezultujuca;
76  }
77
78  int main(int argc, char **argv)
79  {
80      /* Argumenti komandne linije su neophodni */
81      if (argc != 3) {
82          fprintf(stderr,
83              "Greska: Program se poziva sa: ./a.out dat1.txt dat2.txt\
84              n");
```

```
    exit(EXIT_FAILURE);
85 }

87 /* Otvaraju se datoteke u kojima se nalaze elementi listi */
FILE *in1 = NULL;
89 in1 = fopen(argv[1], "r");
if (in1 == NULL) {
91     fprintf(stderr,
93         "Greska: Neuspesno otvaranje datoteke %s.\n", argv[1]);
    exit(EXIT_FAILURE);
}

95 FILE *in2 = NULL;
97 in2 = fopen(argv[2], "r");
if (in2 == NULL) {
99     fprintf(stderr,
101         "Greska: Neuspesno otvaranje datoteke %s.\n", argv[2]);
    exit(EXIT_FAILURE);
}

103 /* Liste su na pocetku prazne */
105 int broj;
Cvor *lista1 = NULL;
107 Cvor *lista2 = NULL;

109 /* Ucitavanje listi */
while (fscanf(in1, "%d", &broj) != EOF)
111     dodaj_na_kraj_liste(&lista1, broj);

113 while (fscanf(in2, "%d", &broj) != EOF)
    dodaj_na_kraj_liste(&lista2, broj);
115

117 /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
fclose(in1);
fclose(in2);

119 /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
objedinjavanjem listi */
121 Cvor *rezultat = objedini(&lista1, &lista2);

123 /* Ispis rezultujuce liste. */
125 ispisi_listu(rezultat);

127 /* Lista rezultat dobijena je prevezivanjem cvorova polaznih
listi. Njenim oslobadjanjem oslobadja se sva zauzeta memorija.
*/
129 oslobodi_listu(&rezultat);

131 exit(EXIT_SUCCESS);
133 }
```

### Rešenje 4.8

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
5    predstavlja vidjenu zagradu i pokazivac na sledeci cvor liste */
6 typedef struct cvor {
7     char zagrada;
8     struct cvor *sledeci;
9 } Cvor;
10
11 /* Funkcija koja oslobadja memoriju zauzetu stekom */
12 void oslobodi_stek(Cvor ** stek)
13 {
14     Cvor *tekuci;
15     Cvor *pomocni;
16
17     /* Oslobadja se cvor po cvor steka */
18     tekuci = *stek;
19     while (tekuci != NULL) {
20         pomocni = tekuci->sledeci;
21         free(tekuci);
22         tekuci = pomocni;
23     }
24
25     /* Stek se proglašava praznim */
26     *stek = NULL;
27 }
28
29 int main()
30 {
31     /* Stek je na pocetku prazan */
32     Cvor *stek = NULL;
33     FILE *ulaz = NULL;
34     char c;
35     Cvor *pomocni = NULL;
36
37     /* Otvaranje datoteke za citanje izraza */
38     ulaz = fopen("izraz.txt", "r");
39     if (ulaz == NULL) {
40         fprintf(stderr,
41             "Greska: Neuspesno otvaranje datoteke izraz.txt!\n");
42         exit(EXIT_FAILURE);
43     }
44
45     /* Cita karakter po karakter iz datoteke */
46     while ((c = fgetc(ulaz)) != EOF) {
47         /* Ako je ucitana otvorena zagrada, stavlja se na stek */
48         if (c == '(' || c == '{' || c == '[') {
49             /* Alocira se memorija za novi cvor liste i proverava se
50                uspesnost alokacije */
```

```
51     pomocni = (Cvor *) malloc(sizeof(Cvor));
52     if (pomocni == NULL) {
53         fprintf(stderr, "Greska: Neuspesna alokacija memorije!\n");
54         /* Oslobadjanje memorije zauzete stekom */
55         oslobodi_stek(&stek);
56         /* Prekid izvršavanja programa */
57         exit(EXIT_FAILURE);
58     }
59
60     /* Inicijalizacija polja strukture */
61     pomocni->zagrada = c;
62
63     /* Promena vrha steka */
64     pomocni->sledeci = stek;
65     stek = pomocni;
66 }
67 /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
68 prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
69 otvorena zagrada */
70 else {
71     if (c == ')' || c == '}' || c == ']') {
72         if (stek != NULL && ((stek->zagrada == '(' && c == ')')
73             || (stek->zagrada == '{' && c == '}')
74             || (stek->zagrada == '[' && c == ']')))
75         {
76             /* Sa vrha steka se uklanja otvorena zagrada */
77             pomocni = stek->sledeci;
78             free(stek);
79             stek = pomocni;
80         } else {
81             /* Dakle, zagrade u izrazu nisu ispravno uparene */
82             break;
83         }
84     }
85 }
86
87 /* Procitana je cela datoteka i treba je zatvoriti. */
88 fclose(ulaz);
89
90 /* Ako je stek prazan i procitana je cela datoteka, zagrade su
91 ispravno uparene. */
92 if (stek == NULL && c == EOF)
93     printf("Zagrade su ispravno uparene.\n");
94 else {
95     /* U suprotnom se zakljucuje da zagrade nisu ispravno uparene. */
96     printf("Zagrade nisu ispravno uparene.\n");
97     /* Oslobadja se memorija koja je ostala zauzeta stekom. */
98     oslobodi_stek(&stek);
99 }
100
101 exit(EXIT_SUCCESS);
```

```
}  
}
```

### Rešenje 4.9

*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 sadrzi ime etikete i
19    pokazivac na sledeci 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
26    njegovu adresu ili NULL ako alokacija nije bila uspesna */
27 Cvor *napravi_cvor(char *etiketa);
28
29 /* Funkcija oslobadja memoriju zauzetu stekom */
30 void oslobodi_stek(Cvor ** adresa_vrha);
31
32 /* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske
33    pri alokaciji 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
39    u suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan
40    (pa samim tim nije bilo moguće skinuti vrednost sa steka) ili 1 u
41    suprotnom */
42 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);
43
44 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
```

```

    steka. Ukoliko je stek prazan, vraca NULL */
46 char *vrh_steka(Cvor * vrh);

48 /* Funkcija prikazuje stek od vrha prema dnu */
void prikazi_stek(Cvor * vrh);

50
52 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
etiketu, i upisuje je u nisku na koju pokazuje pokazivac
etiketa. Vraca EOF u slucaju da se dodje do kraja datoteke pre
54 nego sto se procita etiketa. Vraca OTVORENA, ako je procitana
otvorena etiketa, odnosno ZATVORENA, ako je procitana zatvorena
56 etiketa */
int uzmi_etiketu(FILE * f, char *etiketa);

58
#endif

```

stek.c

```

1 #include "stek.h"

3 Cvor *napravi_cvor(char *etiketa)
{
5     /* Alokacija memorije za novi cvor uz proveru uspesnosti */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
7     if (novi == NULL)
        return NULL;
9
11    /* Inicijalizacija polja u novom cvoru */
    if (strlen(etiketa) >= MAX) {
        fprintf(stderr, "Greska: Etiketa je preduga, bice skracena.\n");
13        etiketa[MAX - 1] = '\0';
    }
15    strcpy(novi->etiketa, etiketa);
    novi->sledeci = NULL;
17
19    /* Vraca se adresa novog cvora */
    return novi;
}

21 void oslobodi_stek(Cvor ** adresa_vrha)
23 {
    Cvor *pomocni;
25
27    /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
    while (*adresa_vrha != NULL) {
        /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
29        osloboditi cvor koji predstavlja vrh steka */
        pomocni = *adresa_vrha;
31        /* Sledeci cvor je novi vrh steka */
        *adresa_vrha = (*adresa_vrha)->sledeci;
33        free(pomocni);
    }
}

```

## 4 Dinamičke strukture podataka

```
    }
35
    /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
37     *adresa_vrha ce pokazivati na NULL. */
    }
39
    int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
41 {
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43     Cvor *novi = napravi_cvor(etiketa);
    if (novi == NULL)
45         return 1;

    /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
47     novi->sledeci = *adresa_vrha;
    *adresa_vrha = novi;
49     return 0;
51 }

53 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
    {
55     Cvor *pomocni;

57     /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom
    i vraća se 0 */
59     if (*adresa_vrha == NULL)
        return 0;

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

67     /* Element sa vrha steka se uklanja */
    pomocni = *adresa_vrha;
    *adresa_vrha = (*adresa_vrha)->sledeci;
69     free(pomocni);

71     /* Vraća se indikator uspesno izvršene radnje */
73     return 1;
    }

75
77 char *vrh_steka(Cvor * vrh)
    {
79     /* Prazan stek nema cvor koji je vrh i vraća se NULL */
    if (vrh == NULL)
        return NULL;

81     /* Inace, vraća se pokazivac na nisku etiketa koja je polje cvora
83     koji je na vrhu steka. */
    return vrh->etiketa;
85 }
```

```
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,
107     dok to u C-u ne vazii. Zato ce sve etikete biti prevedene u
108     zapis 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;
119                 } else {
120                     if (isalpha(c)) {
121                         /* Cita se otvorena etiketa */
122                         tip = OTVORENA;
123                         etiketa[i++] = tolower(c);
124                     }
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
133                     smanjuje i smesta u etiketu */
134                     etiketa[i++] = tolower(c);
135                 } else {
136                     /* Inace, staje se sa citanjem etikete. Korektno se
137                     zavrшава niska koja sadrzi procitanu etiketu i vraća se
```



## 4 Dinamičke strukture podataka

```
        njen tip */
139     etiketa[i] = '\0';
        return tip;
141     }
        break;
143     }
    }
145     /* Doslo se do kraja datoteke pre nego sto je procitana naredna
        etiketa i vraca se EOF. */
147     return EOF;
}
```

*main.c*

```
#include "stek.h"
2
int main(int argc, char **argv)
4 {
    /* Na pocetku, stek je prazan i etikete su uparene jer nijedna
6     jos nije procitana. */
    Cvor *vrh = NULL;
    char etiketa[MAX];
    int tip;
10    int uparene = 1;
    FILE *f = NULL;
12
    /* Ime datoteke se preuzima iz komandne linije. */
14    if (argc < 2) {
        fprintf(stderr, "Greska:");
16        fprintf(stderr,
            "Program se poziva sa:\n %s ime_html_datoteke\n",
18            argv[0]);
        exit(EXIT_FAILURE);
20    }

22    /* Datoteka se otvara za citanje */
    if ((f = fopen(argv[1], "r")) == NULL) {
24        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
            argv[1]);
26        exit(EXIT_FAILURE);
    }
28
    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
30    while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
        /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
32        etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
        potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
34        koje nemaju sadrzaj (npr link). Zbog jednostavnosti
        pretpostavlja se da njih nema u HTML dokumentu. */
36        if (tip == OTVORENA) {
            if (strcmp(etiketa, "br") != 0
```

```

38     && strcmp(etiketa, "hr") != 0
39     && strcmp(etiketa, "meta") != 0)
40     if (potisni_na_stek(&vrh, etiketa) == 1) {
41         fprintf(stderr,
42             "Greska: Neuspesna alokacija memorije za nov cvor\n
43 ");
44         oslobodi_stek(&vrh);
45         exit(EXIT_FAILURE);
46     }
47     /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da
48     je u pitanju zatvaranje etikete koja je poslednja otvorena,
49     a jos uvek nije zatvorena. Ona se mora nalaziti na vrhu
50     steka. Ako je taj uslov ispunjen, skida se sa steka, jer je
51     upravo zatvorena. U suprotnom, pronadjena je nepravilnost i
52     etikete nisu pravilno uparene. */
53     else if (tip == ZATVORENA) {
54         if (vrh_steka(vrh) != NULL
55             && strcmp(vrh_steka(vrh), etiketa) == 0)
56             skini_sa_steka(&vrh, NULL);
57         else {
58             printf("Etikete nisu pravilno uparene\n");
59             printf("(nadjena je etiketa </%s>", etiketa);
60             if (vrh_steka(vrh) != NULL)
61                 printf(", a poslednja otvorena je </%s>\n",
62                     vrh_steka(vrh));
63             else
64                 printf(" koja nije otvorena)\n");
65             uparene = 0;
66             break;
67         }
68     }
69 }
70 /* Završeno je citanje i datoteka se zatvara */
71 fclose(f);
72
73 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi
74 trebalo da bude prazan. Ukoliko nije, tada postoje etikete
75 koje su ostale otvorene */
76 if (uparene) {
77     if (vrh_steka(vrh) == NULL)
78         printf("Etikete su pravilno uparene!\n");
79     else {
80         printf("Etikete nisu pravilno uparene\n");
81         printf("(etiketa </%s> nije zatvorena)\n", vrh_steka(vrh));
82         /* Oslobadja se memorija zauzeta stekom */
83         oslobodi_stek(&vrh);
84     }
85 }
86
87 exit(EXIT_SUCCESS);
88 }

```

### Rešenje 4.10

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
11    i opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
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
25    sa poslate adrese i vraca adresu novog cvora ili NULL ako je
26    doslo do greske pri alokaciji. Prosledjuje joj se pokazivac na
27    zahtev koji treba smestiti u novi cvor zbog smestanja manjeg
28    podatka na sistemski stek. Pokazivac na strukturu Zahtev je
29    manje velicine u bajtovima(B) u odnosu na strukturu Zahtev. */
30 Cvor *napravi_cvor(Zahtev * zahtev);
31
32 /* Funkcija prazni red oslobadjajuci memoriju koji je red zauzimao */
33 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
34
35 /* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo
36    do greske pri alokaciji memorije za novi cvor, inace vraca 0. */
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
38                Zahtev * zahtev);
39
40 /* Funkcija skida sa pocetka reda zahtev. Ako je argument zahtev
41    pokazivac razlicit od NULL, u strukturu na koju on pokazuje
42    upisuje se zahtev upravo skinut sa reda, inace se ne upisuje
43    nista. Funkcija vraca 0, ako je red bio prazan, inace vraca 1. */
44 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
45                  Zahtev * zahtev);
46
47 /* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev
```

```

korisnika na pocetku reda. Ako je red prazan, vraca NULL. */
49 Zahtev *pocetak_reda(Cvor * pocetak);

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

53 #endif

```

*red.c*

```

#include "red.h"
2
Cvor *napravi_cvor(Zahtev * zahtev)
4 {
    /* Alokacija memorije za novi cvor uz proveru uspesnosti */
6     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
8         return NULL;

10     /* Inicijalizacija polja strukture */
    novi->nalog = *zahtev;
12     novi->sledeci = NULL;

14     /* Vraca se adresa novog cvora */
    return novi;
16 }

18 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
{
20     Cvor *pomocni = NULL;

22     /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
    while (*pocetak != NULL) {
24         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
            osloboditi cvor sa pocetka reda */
26         pomocni = *pocetak;
            *pocetak = (*pocetak)->sledeci;
28         free(pomocni);
    }

30     /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
        treba postaviti na NULL. */
32     *kraj = NULL;
}

34
36 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                Zahtev * zahtev)
{
38     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(zahtev);
40     if (novi == NULL)
        return 1;
}

```

## 4 Dinamičke strukture podataka

```
42
44 /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
46 kraj, to je efikasno koliko i dodavanje na pocetak liste */
48 if (*adresa_kraja != NULL) {
46     (*adresa_kraja)->sledeci = novi;
48     *adresa_kraja = novi;
48 } else {
50     /* Ako je red bio ranije prazan */
50     *adresa_pocetka = novi;
52     *adresa_kraja = novi;
52 }

54 /* Vraca se indikator uspesnog dodavanja */
56 return 0;
56 }

58 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
60 Zahtev * zahtev)
62 {
62     Cvor *pomocni = NULL;
62
64     /* Ako je red prazan */
64     if (*adresa_pocetka == NULL)
66         return 0;
66
68     /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
68     zahtev koji je na pocetku reda. */
68     if (zahtev != NULL)
70         *zahtev = (*adresa_pocetka)->nalog;
70
72     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i azurira
72     se pokazivac na adresi adresa_pocetka da pokazuje na sledeci
74     cvor u redu. */
74     pomocni = *adresa_pocetka;
76     *adresa_pocetka = (*adresa_pocetka)->sledeci;
76     free(pomocni);
78
80     /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,
80     potrebno je azurirati i vrednost pokazivaca na adresi
82     adresa_kraja na NULL */
82     if (*adresa_pocetka == NULL)
84         *adresa_kraja = NULL;
84
86     return 1;
86 }

88 Zahtev *pocetak_reda(Cvor * pocetak)
90 {
90     /* U praznom redu nema zahteva */
92     if (pocetak == NULL)
92         return NULL;
```

```

94  /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
    return &(pocetak->nalog);
96  }

98  void prikazi_red(Cvor * pocetak)
    {
100  /* Prikazuje se sadrzaj reda od pocetka prema kraju */
    for (; pocetak != NULL; pocetak = pocetak->sledeci)
102      printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);

104  printf("\n");
    }

```

*main.c*

```

1  #include <stdio.h>
    #include <stdlib.h>
3  #include <string.h>
    #include "red.h"
5
6  #define VREME_ZA_PAUZU 5
7
8  int main(int argc, char **argv)
9  {
10     /* Red je prazan. */
11     Cvor *pocetak = NULL, *kraj = NULL;
    Zahtev nov_zahtev;
13     Zahtev *sledeci = NULL;
    char odgovor[3];
15     int broj_usluzenih = 0;

17     /* Sluzbenik evidentira korisnicke zahteve unosnjem njihovog
        JMBG broja i opisa potrebne usluge. */
19     printf("Sluzbenik evidentira korisnicke zahteve:\n");
    while (1) {
21         /* Ucitava se JMBG broj */
        printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
23         if (scanf("%s", nov_zahtev.jmbg) == EOF)
            break;

25         /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
            JMBG broja procitao i da bi fgets nakon toga procitala
            ispravan red sa opisom zahteva */
27         getchar();

31         /* Ucitava se opis problema */
        printf("\tOpis problema: ");
33         fgets(nov_zahtev.opis, MAX - 1, stdin);
        /* Ako je poslednji karakter nov red, eliminiše se */
35         if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
            nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';

```

## 4 Dinamičke strukture podataka

```
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,
41             "Greska: Neuspesna alokacija memorije za nov cvor\n");
42         oslobodi_red(&pocetak, &kraj);
43         exit(EXIT_FAILURE);
44     }
45 }
46
47 /* Otvaranje datoteke za dopisivanje izvestaja */
48 FILE *izlaz = fopen("izvestaj.txt", "a");
49 if (izlaz == NULL) {
50     fprintf(stderr,
51         "Greska: Neuspesno otvaranje datoteke izvestaj.txt\n");
52     exit(EXIT_FAILURE);
53 }
54
55 /* Dokle god ima korisnika u redu, treba ih uslužiti */
56 while (1) {
57     sledeci = pocetak_reda(pocetak);
58     /* Ako nema nikog vise u redu, prekida se petlja */
59     if (sledeci == NULL)
60         break;
61
62     printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
63     printf("i zahtevom: %s\n", sledeci->opis);
64
65     skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
66
67     broj_usluzenih++;
68
69     printf("\tDa li ga vratate na kraj reda? [Da/Ne] ");
70     scanf("%s", odgovor);
71
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);
77
78     if (broj_usluzenih == VREME_ZA_PAUZU) {
79         printf("\nDa li je kraj smene? [Da/Ne] ");
80         scanf("%s", odgovor);
81
82         if (strcmp(odgovor, "Da") == 0)
83             break;
84         else
85             broj_usluzenih = 0;
86     }
87 }
```

```

89  /******
    Usluzivanje korisnika moze da se izvrši i na sledeci način:  */
91  /******
    while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
93      printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
              nov_zahtev.jmbg);
95      printf("sa zahtevom: %s\n", nov_zahtev.opis);
      broj_usluzenih++;
97
      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
99      scanf("%s", odgovor);
      if (strcmp(odgovor, "Da") == 0)
101         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
      else
103         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
                  nov_zahtev.jmbg, nov_zahtev.opis);
105
      if (broj_usluzenih == VREME_ZA_PAUZU) {
107         printf("\nDa li je kraj smene? [Da/Ne] ");
         scanf("%s", odgovor);
109         if (strcmp(odgovor, "Da") == 0)
             break;
111         else
             broj_usluzenih = 0;
113     }
    }
115  /******
117  /* Datoteka vise nije potrebna i treba je zatvoriti. */
    fclose(izlaz);
119
    /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
121     neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
        koju zauzima red sa neobradjenim zahtevima korisnika. */
123     oslobodi_red(&pocetak, &kraj);
125     exit(EXIT_SUCCESS);
}

```

### Rešenje 4.11

*dvostruko\_povezana\_lista.h*

```

1  #ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
2  #define _DVOSTRUKO_POVEZANA_LISTA_H_
3
4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
   vrednost i pokazivace na sledeci i prethodni cvor liste. */
6  typedef struct cvor {
    int vrednost;

```



## 4 Dinamičke strukture podataka

```
8   struct cvor *sledeci;
9   struct cvor *prethodni;
10  } Cvor;

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

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

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

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

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

33  /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
34     dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
35  int dodaj_iza(Cvor * tekuci, int broj);

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

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

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

51  /* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi
```

```

60     ciji pokazivac glava se nalazi na adresi adresa_glave. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja,
62                 Cvor * tekuci);

64 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
Azurira pokazivac na glavu liste, koji može biti promenjen u
66 slucaju da se obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
68                 broj);

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

76
78 /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka
kraju liste, razdvojene zaptama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

80
82 /* Funkcija prikazuje vrednosti cvorova liste počevsi od kraja ka
glavi liste, razdvojene zaptama i uokvirene zagradama. */
void ispisi_listu_unazad(Cvor * kraj);

84
#endif

```

*dvostruko\_povezana\_lista.c*

```

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

5  Cvor *napravi_cvor(int broj)
   {
7     /* Alokacija memorije za novi cvor uz proveru uspesnosti */
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9     if (novi == NULL)
       return NULL;

11
   /* Inicijalizacija polja strukture */
13     novi->vrednost = broj;
   novi->sledeci = NULL;

15
   /* Vraca se adresa novog cvora */
17     return novi;
   }

19
21 void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
   {
   Cvor *pomocni = NULL;

```

## 4 Dinamičke strukture podataka

```
23
24 /* Ako lista nije prazna, onda treba osloboditi memoriju */
25 while (*adresa_glave != NULL) {
26     /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
27        osloboditi memoriju cvora koji predstavlja glavu liste */
28     pomocni = (*adresa_glave)->sledeci;
29     free(*adresa_glave);
30     /* Sledeci cvor je nova glava liste */
31     *adresa_glave = pomocni;
32 }
33 /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj
34    liste treba postaviti na NULL */
35 *adresa_kraja = NULL;
36 }
37
38 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
39                            adresa_kraja, int broj)
40 {
41     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
42     Cvor *novi = napravi_cvor(broj);
43     if (novi == NULL)
44         return 1;
45
46     /* Sledbenik novog cvora je glava stare liste */
47     novi->sledeci = *adresa_glave;
48
49     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
50        treba da bude novi cvor. Inace, novi cvor je u isto vreme i
51        pocetni i krajnji. */
52     if (*adresa_glave != NULL)
53         (*adresa_glave)->prethodni = novi;
54     else
55         *adresa_kraja = novi;
56
57     /* Novi cvor je nova glava liste */
58     *adresa_glave = novi;
59
60     /* Vraca se indikator uspesnog dodavanja */
61     return 0;
62 }
63
64 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
65                         int broj)
66 {
67     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
68     Cvor *novi = napravi_cvor(broj);
69     if (novi == NULL)
70         return 1;
71
72     /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
73        ujedno i cela lista. Azuriraju se vrednosti na koje pokazuju
74        adresa_glave i adresa_kraja */

```

```
75  if (*adresa_glave == NULL) {
76      *adresa_glave = novi;
77      *adresa_kraja = novi;
78  } else {
79      /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
80         sledbenik poslednjeg cvora i azurira se samo pokazivac na
81         kraj liste */
82      (*adresa_kraja)->sledeci = novi;
83      novi->prethodni = (*adresa_kraja);
84      *adresa_kraja = novi;
85  }
86
87  /* Vraca se indikator uspesnog dodavanja */
88  return 0;
89  }
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     /* Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
101        provera da li se doslo do poslednjeg cvora liste pre nego sto
102        se proveru vrednost u sledecem cvoru jer u slucaju poslednjeg,
103        sledeci ne postoji pa ni njegova vrednost. */
104     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
105         glava = glava->sledeci;
106
107     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
108        poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci
109        ima vrednost vecu od broj */
110     return glava;
111  }
112
113  int dodaj_iza(Cvor * tekuci, int broj)
114  {
115     /* Kreira se novi cvor i proveru se uspesnost kreiranja */
116     Cvor *novi = napravi_cvor(broj);
117     if (novi == NULL)
118         return 1;
119
120     novi->sledeci = tekuci->sledeci;
121     novi->prethodni = tekuci;
122
123     /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje
124        prethodnik, a potom i tekuci dobija novog sledeceg
125        postavljanjem pokazivaca na ispravne adrese */
126     if (tekuci->sledeci != NULL)
```

## 4 Dinamičke strukture podataka

```
127     tekuci->sledeci->prethodni = novi;
128     tekuci->sledeci = novi;
129
130     /* Vraca se indikator uspesnog dodavanja */
131     return 0;
132 }
133
134 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
135                    broj)
136 {
137     /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor */
138     if (*adresa_glave == NULL) {
139         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
140         Cvor *novi = napravi_cvor(broj);
141         if (novi == NULL)
142             return 1;
143
144         /* Azuriraju se vrednosti pocetka i kraja liste */
145         *adresa_glave = novi;
146         *adresa_kraja = novi;
147
148         /* Vraca se indikator uspesnog dodavanja */
149         return 0;
150     }
151
152     /* Ukoliko je vrednost glave liste veca ili jednaka od nove
153        vrednosti onda novi cvor treba staviti na pocetak liste */
154     if ((*adresa_glave)->vrednost >= broj) {
155         return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
156     }
157
158     /* Pronazi se cvor iza koga treba uvezati novi cvor */
159     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
160     /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
161     if (dodaj_iza(pomocni, broj) == 1)
162         return 1;
163
164     /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
165        novi cvor poslednji u listi. */
166     if (pomocni == *adresa_kraja)
167         *adresa_kraja = pomocni->sledeci;
168
169     return 0;
170 }
171
172 Cvor *pretrazi_listu(Cvor * glava, int broj)
173 {
174     /* Obilaze se cvorovi liste */
175     for (; glava != NULL; glava = glava->sledeci)
176         /* Ako je vrednost tekućeg cvora jednaka zadatom broju,
177            pretraga se obustavlja */
178         if (glava->vrednost == broj)
179             return glava;
```

```
179     /* Nema trazenog broja u listi i vraca se NULL */
181     return NULL;
183 }
184
185 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
186 {
187     /* Obilaze se cvorovi liste */
188     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
189     for (; glava != NULL && glava->vrednost <= broj;
190          glava = glava->sledeci)
191         /* Ako je vrednost tekuceg cvora jednaka zadatom broju,
192          pretraga se obustavlja */
193         if (glava->vrednost == broj)
194             return glava;
195
196     /* Nema trazenog broja u listi i bice vrateno NULL */
197     return NULL;
198 }
199
200 /* Funkcija brise cvor zadat argumentom tekuci. Brisanje odredjenog
201    cvora dvostruko povezane liste moze se lako realizovati jer cvor
202    sadrzi pokazivace na svog sledbenika i prethodnika u listi. */
203 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja,
204                  Cvor * tekuci)
205 {
206     /* Ako je tekuci NULL pokazivac, nema potrebe za brisanjem */
207     if (tekuci == NULL)
208         return;
209
210     /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da
211        njegov sledbenik bude sledbenik tekuceg cvora */
212     if (tekuci->prethodni != NULL)
213         tekuci->prethodni->sledeci = tekuci->sledeci;
214
215     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
216        treba da bude prethodnik tekuceg cvora */
217     if (tekuci->sledeci != NULL)
218         tekuci->sledeci->prethodni = tekuci->prethodni;
219
220     /* Ako je glava cvor koji se brise, nova glava liste ce biti
221        sledbenik stare glave. */
222     if (tekuci == *adresa_glave)
223         *adresa_glave = tekuci->sledeci;
224
225     /* Ako je cvor koji se brise poslednji u listi, azurira se i
226        pokazivac na kraj liste. */
227     if (tekuci == *adresa_kraja)
228         *adresa_kraja = tekuci->prethodni;
229
230     /* Oslobadja se dinamicki alocirani prostor za cvor tekuci. */
231     free(tekuci);
232 }
```

## 4 Dinamičke strukture podataka

```
231 }
233 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                int broj)
235 {
    Cvor *tekuci = *adresa_glave;
237
    /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
239     takvi cvorovi se brisu iz liste. */
    while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
241     obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}
243
244 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                adresa_kraja, int broj)
245 {
    Cvor *tekuci = *adresa_glave;
247
    /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
249     takvi cvorovi se brisu iz liste. */
    while ((tekuci =
251     pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
253     obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}
255
256 void ispisi_listu(Cvor * glava)
257 {
    putchar('[');
259     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
        pocetka prema kraju liste. */
    for (; glava != NULL; glava = glava->sledeci) {
261         printf("%d", glava->vrednost);
263         if (glava->sledeci != NULL)
            printf(", ");
265     }
267     printf("]\n");
}
269
270 void ispisi_listu_unazad(Cvor * kraj)
271 {
    putchar('[');
273     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
        kraja prema pocetku liste. */
    for (; kraj != NULL; kraj = kraj->prethodni) {
275         printf("%d", kraj->vrednost);
277         if (kraj->prethodni != NULL)
            printf(", ");
279     }
    printf("]\n");
281 }
```

main\_a.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"
4
5 int main()
6 {
7     /* Lista je prazna na pocetku */
8     /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
9     da bi operacije poput dodavanja na kraj liste i ispisivanja
10     liste unazad bile efikasne poput dodavanja na pocetak liste i
11     ispisivanja liste od pocetnog do poslednjeg cvora. */
12     Cvor *glava = NULL;
13     Cvor *kraj = NULL;
14     Cvor *trazeni = NULL;
15     int broj;
16
17     /* Testira se funkcija za dodavanje novog broja na pocetak liste */
18     printf("Unesite brojeve (CTRL+D za kraj unosa): ");
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,
25                 "Greska: Neuspesna alokacija memorije za cvor.\n");
26             oslobodi_listu(&glava, &kraj);
27             exit(EXIT_FAILURE);
28         }
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 trazeni dobija vrednost rezultata pretrage */
38     trazeni = pretrazi_listu(glava, broj);
39     if (trazeni == NULL)
40         printf("Broj %d se ne nalazi u listi!\n", broj);
41     else
42         printf("Trazeni broj %d je u listi!\n", trazeni->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);
```



## 4 Dinamičke strukture podataka

---

```
51     exit(EXIT_SUCCESS);
    }
```

*main\_b.c*

```
#include <stdio.h>
2 #include <stdlib.h>
#include "dvostruko_povezana_lista.h"
4
int main()
6 {
    /* Lista je prazna na pocetku. */
8     Cvor *glava = NULL;
    Cvor *kraj = NULL;
10     int broj;

12     /* Testira se funkcija za dodavanje novog broja na kraj liste */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
14     while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
16         memorije za novi cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa */
18         if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
            fprintf(stderr,
20                 "Greska: Neuspesna alokacija memorije za cvor.\n");
            oslobodi_listu(&glava, &kraj);
22             exit(EXIT_FAILURE);
        }
24         printf("\tLista: ");
        ispisi_listu(glava);
26     }

28     /* Testira se funkcija za brisanje elemenata iz liste */
    printf("\nUnesite broj koji se brise iz liste: ");
30     scanf("%d", &broj);

32     /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
    obrisi_cvor(&glava, &kraj, broj);
34

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

38     /* Ispisuje se lista unazad */
    printf("\nLista ispisana u nazad: ");
40     ispisi_listu_unazad(kraj);

42     /* Oslobadja se memorija zauzeta za cvorove liste */
    oslobodi_listu(&glava, &kraj);
44

    exit(EXIT_SUCCESS);
}
```

46 }

---

*main.c.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"
4
5 int main()
6 {
7     /* Lista je prazna na pocetku */
8     Cvor *glava = NULL;
9     Cvor *kraj = NULL;
10    Cvor *trazeni = NULL;
11    int broj;
12
13    /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
14     bude uredjena neopadajuće */
15    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
16    while (scanf("%d", &broj) > 0) {
17        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
18         memorije za novi cvor. Memoriju alociranu za cvorove liste
19         treba osloboditi pre napustanja programa */
20        if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
21            fprintf(stderr,
22                "Greska: Neuspesna alokacija memorije za cvor.\n");
23            oslobodi_listu(&glava, &kraj);
24            exit(EXIT_FAILURE);
25        }
26        printf("\tLista: ");
27        ispisi_listu(glava);
28    }
29
30    /* Testira se funkcija za pretragu liste */
31    printf("\nUnesite broj koji se trazi u listi: ");
32    scanf("%d", &broj);
33
34    /* Pokazivac trazeni dobija vrednost rezultata pretrage */
35    trazeni = pretrazi_listu(glava, broj);
36    if (trazeni == NULL)
37        printf("Broj %d se ne nalazi u listi!\n", broj);
38    else
39        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
40
41    /* Testira se funkcija za brisanje elemenata iz liste */
42    printf("\nUnesite broj koji se brise iz liste: ");
43    scanf("%d", &broj);
44
45    /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
46    obrisi_cvor_sortirane_liste(&glava, &kraj, broj);
```

## 4 Dinamičke strukture podataka

---

```
48  printf("Lista nakon brisanja: ");
    ispisi_listu(glava);
50
    /* Ispisuje se lista unazad */
52  printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);
54
    /* Oslobadja se memorija zauzeta za cvorove liste */
56  oslobodi_listu(&glava, &kraj);
58
    exit(EXIT_SUCCESS);
}
```

### Rešenje 4.14

*stabla.h*

```
#ifndef _STABLA_H_
2 #define _STABLA_H_ 1

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

12 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
    inicijalizuje polja strukture i vraća 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 najveću vrednost u
    stablu */
30 Cvor *pronadji_najveci(Cvor * koren);

32 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
void obrisi_element(Cvor ** adresa_korena, int broj);
```

```

34 /* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
36     postablo - Koren - Desno podstablo ) */
37 void ispis_i_stablo_infiksno(Cvor * koren);
38
39 /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji (Koren -
40     Levo podstablo - Desno podstablo ) */
41 void ispis_i_stablo_prefiksno(Cvor * koren);
42
43 /* j) Funkcija koja ispisuje stablo u postfiksnoj notaciji (Levo
44     podstablo - Desno postablo - Koren) */
45 void ispis_i_stablo_postfiksno(Cvor * koren);
46
47 /* k) Funkcija koja oslobadja memoriju zauzetu stablom */
48 void oslobodi_stablo(Cvor ** adresa_korena);
49
50 #endif

```

*stabla.c*

```

#include <stdio.h>
2 #include <stdlib.h>
#include "stabla.h"
4
Cvor *napravi_cvor(int broj)
6 {
8     /* Alocira se memorija za novi cvor i proverava se uspesnost
    alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
        return NULL;
12
    /* Inicijalizuju se polja novog cvora */
14     novi->broj = broj;
    novi->levo = NULL;
16     novi->desno = NULL;
18
    /* Vraca se adresa novog cvora */
    return novi;
20 }
22 int dodaj_u_stablo(Cvor ** adresa_korena, int broj)
{
24     /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
26
        /* Kreira se novi cvor */
28     Cvor *novi_cvor = napravi_cvor(broj);
30
        /* Proverava se uspesnost kreiranja */
        if (novi_cvor == NULL) {

```

## 4 Dinamičke strukture podataka

```
32     /* Ako je doslo do greske, vraca se odgovarajuca vrednost */
33     return 1;
34 }
35
36 /* Inace, novi cvor se proglašava korenom stabla */
37 *adresa_korena = novi_cvor;
38
39 /* I vraca se indikator uspesnosti kreiranja */
40 return 0;
41 }
42
43 /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za
44    zadati broj */
45
46 /* Ako je zadata vrednost manja od vrednosti korena */
47 if (broj < (*adresa_korena)->broj)
48     /* Broj se dodaje u levo podstablo */
49     return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
50 else
51     /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
52        dodaje u desno podstablo */
53     return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
54 }
55
56 Cvor *pretrazi_stablo(Cvor * koren, int broj)
57 {
58     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
59     if (koren == NULL)
60         return NULL;
61
62     /* Ako je trazena vrednost sadrzana u korenu */
63     if (koren->broj == broj) {
64         /* Prekida se pretraga */
65         return koren;
66     }
67
68     /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
69     if (broj < koren->broj)
70         /* Pretraga se nastavlja u levom podstablu */
71         return pretrazi_stablo(koren->levo, broj);
72     else
73         /* U suprotnom, pretraga se nastavlja u desnom podstablu */
74         return pretrazi_stablo(koren->desno, broj);
75 }
76
77 Cvor *pronadji_najmanji(Cvor * koren)
78 {
79     /* Ako je stablo prazno, prekida se pretraga */
80     if (koren == NULL)
81         return NULL;
82
83     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
```

```
84     levo od njega */
86     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
87        najmanju vrednost */
88     if (koren->levo == NULL)
89         return koren;
90
91     /* Inace, pretragu treba nastaviti u levom podstablu */
92     return pronadji_najmanji(koren->levo);
93 }
94
95 Cvor *pronadji_najveci(Cvor * koren)
96 {
97     /* Ako je stablo prazno, prekida se pretraga */
98     if (koren == NULL)
99         return NULL;
100
101     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
102        desno od njega */
103
104     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
105        najveću vrednost */
106     if (koren->desno == NULL)
107         return koren;
108
109     /* Inace, pretragu treba nastaviti u desnom podstablu */
110     return pronadji_najveci(koren->desno);
111 }
112
113 void obrisi_element(Cvor ** adresa_korena, int broj)
114 {
115     Cvor *pomocni_cvor = NULL;
116
117     /* Ako je stablo prazno, brisanje nije primenljivo */
118     if (*adresa_korena == NULL)
119         return;
120
121     /* Ako je vrednost koju treba obrisati manja od vrednosti u
122        korenu stabla, ona se eventualno nalazi u levom podstablu, pa
123        treba rekursivno primeniti postupak na levo podstablo. Koren
124        ovako modifikovanog stabla je nepromenjen. */
125     if (broj < (*adresa_korena)->broj) {
126         obrisi_element(&(*adresa_korena)->levo, broj);
127         return;
128     }
129
130     /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
131        stabla, ona se eventualno nalazi u desnom podstablu pa treba
132        rekursivno primeniti postupak na desno podstablo. Koren ovako
133        modifikovanog stabla je nepromenjen. */
134     if ((*adresa_korena)->broj < broj) {
135         obrisi_element(&(*adresa_korena)->desno, broj);
```

## 4 Dinamičke strukture podataka

```
136     return;
137 }
138
139 /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
140    jednaka broju koji se brise tj. slucaj kada treba obrisati
141    koren */
142
143 /* 1. Ako koren nema sinova, tada se on prosto brise, i rezultat
144    je prazno stablo (vraca se NULL) */
145 if ((*adresa_korena)->levo == NULL
146     && (*adresa_korena)->desno == NULL) {
147     free(*adresa_korena);
148     *adresa_korena = NULL;
149     return;
150 }
151
152 /* 2. Ako koren ima samo levog sina, tada se brisanje vrši tako
153    sto se brise koren, a novi koren postaje levi sin */
154 if ((*adresa_korena)->levo != NULL
155     && (*adresa_korena)->desno == NULL) {
156     pomocni_cvor = (*adresa_korena)->levo;
157     free(*adresa_korena);
158     *adresa_korena = pomocni_cvor;
159     return;
160 }
161
162 /* 3. Ako koren ima samo desnog sina, tada se brisanje vrši tako
163    sto se brise koren, a novi koren postaje desni sin */
164 if ((*adresa_korena)->desno != NULL
165     && (*adresa_korena)->levo == NULL) {
166     pomocni_cvor = (*adresa_korena)->desno;
167     free(*adresa_korena);
168     *adresa_korena = pomocni_cvor;
169     return;
170 }
171
172 /* 4. Ako koren ima oba sina, najpre se potrazi sledbenik korena
173    (u smislu poretka) u stablu. To je upravo po vrednosti
174    najmanji cvor u desnom podstablu koji se moze pronaci npr.
175    funkcijom pronadji_najmanji(). Potom se u koren smesti
176    vrednost pronadjenog cvora, a u taj cvor se smesti vrednost
177    korena (tj. broj koji se brise). Zatim se rekurzivno pozove
178    funkcija za brisanje nad desnim podstablom. S obzirom da u
179    njemu treba obrisati najmanji element, a on zasigurno ima
180    najvise jednog potomka, jasno je da ce njegovo brisanje biti
181    obavljeno na jedan od jednostavnijih nacina koji su gore
182    opisani. */
183 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
184 (*adresa_korena)->broj = pomocni_cvor->broj;
185 pomocni_cvor->broj = broj;
186 obrisi_element(&(*adresa_korena)->desno, broj);
187 }
```

```
188 void ispisi_stablo_infiksno(Cvor * koren)
190 {
192     /* Ako stablo nije prazno */
194     if (koren != NULL) {
196         /* Prvo se ispisuju svi cvorovi levo od korena */
198         ispisi_stablo_infiksno(koren->levo);
200         /* Zatim se ispisuje vrednost u korenu */
202         printf("%d ", koren->broj);
204         /* Na kraju se ispisuju cvorovi desno od korena */
206         ispisi_stablo_infiksno(koren->desno);
208     }
210 }
212
214 void ispisi_stablo_prefiksno(Cvor * koren)
216 {
218     /* Ako stablo nije prazno */
220     if (koren != NULL) {
222         /* Prvo se ispisuje vrednost u korenu */
224         printf("%d ", koren->broj);
226         /* Zatim se ispisuju svi cvorovi levo od korena */
228         ispisi_stablo_prefiksno(koren->levo);
230         /* Na kraju se ispisuju svi cvorovi desno od korena */
232         ispisi_stablo_prefiksno(koren->desno);
234     }
236 }
238
240 void ispisi_stablo_postfiksno(Cvor * koren)
242 {
244     /* Ako stablo nije prazno */
246     if (koren != NULL) {
248         /* Prvo se ispisuju svi cvorovi levo od korena */
250         ispisi_stablo_postfiksno(koren->levo);
252         /* Zatim se ispisuju svi cvorovi desno od korena */
254         ispisi_stablo_postfiksno(koren->desno);
256         /* Na kraju se ispisuje vrednost u korenu */
258         printf("%d ", koren->broj);
260     }
262 }
264
266 void oslobodi_stablo(Cvor ** adresa_korena)
268 {
```



## 4 Dinamičke strukture podataka

```
240  /* Ako je stablo prazno, nepotrebno je oslobadjeti memoriju */
241  if (*adresa_korena == NULL)
242      return;
244  /* Oslobadja se memorija zauzeta levim podstablom */
245  oslobodi_stablo(&(*adresa_korena)->levo);
246
247  /* Oslobadja se memorija zauzeta desnim podstablom */
248  oslobodi_stablo(&(*adresa_korena)->desno);
249
250  /* Oslobadja se memorija zauzeta korenom */
251  free(*adresa_korena);
252
253  /* Proglasava se stablo praznim */
254  *adresa_korena = NULL;
255  }
```

*main.c*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"
4
5  int main()
6  {
7      Cvor *koren;
8      int n;
9      Cvor *trazeni_cvor;
10
11     /* Proglasava se stablo praznim */
12     koren = NULL;
13
14     /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
15     dodavanja */
16     printf("Unesite brojeve (CTRL+D za kraj unosa): ");
17     while (scanf("%d", &n) != EOF) {
18         if (dodaj_u_stablo(&koren, n) == 1) {
19             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
20             oslobodi_stablo(&koren);
21             exit(EXIT_FAILURE);
22         }
23     }
24
25     /* Generisu se trazeni ispisi: */
26     printf("\nInfiksni ispisi: ");
27     ispisi_stablo_infiksno(koren);
28     printf("\nPrefiksni ispisi: ");
29     ispisi_stablo_prefiksno(koren);
30     printf("\nPostfiksni ispisi: ");
31     ispisi_stablo_postfiksno(koren);
32 }
```

```

33  /* Demonstrira se rad funkcije za pretragu */
    printf("\nTraži se broj: ");
35  scanf("%d", &n);
    trazeni_cvor = pretrazi_stablo(koren, n);
37  if (trazeni_cvor == NULL)
        printf("Broj se ne nalazi u stablu!\n");
39  else
        printf("Broj se nalazi u stablu!\n");
41
    /* Demonstrira se rad funkcije za brisanje */
43  printf("Briše se broj: ");
    scanf("%d", &n);
45  obrisi_element(&koren, n);
    printf("Rezultujuće stablo: ");
47  ispisi_stablo_infiksno(koren);
    printf("\n");
49
    /* Oslobadja se memorija zauzeta stablom */
51  oslobodi_stablo(&koren);
53
    exit(EXIT_SUCCESS);
}

```

### Rešenje 4.15

```

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

6  #define MAX 50

8  /* Struktura kojom se opisuje cvor stabla: sadrži rec, njen broj
    pojavljivanja i redom pokazuje na levo i desno podstablo */
10 typedef struct cvor {
    char *rec;
12     int broj;
    struct cvor *levo;
14     struct cvor *desno;
} Cvor;
16

/* Funkcija koja kreira novi cvor stabla */
18 Cvor *napravi_cvor(char *rec)
{
20     /* Alocira se memorija za novi cvor i proverava se uspešnost
        alokacije */
22     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
    if (novi_cvor == NULL)
24         return NULL;
26
    /* Alocira se memorija za zadatu rec: potrebno je rezervirati

```

## 4 Dinamičke strukture podataka

```
    memoriju za svaki karakter reci ukljucujuci i terminirajucu
28     nulu */
    novi_cvor->rec =
30     (char *) malloc((strlen(rec) + 1) * sizeof(char));
    if (novi_cvor->rec == NULL) {
32     free(novi_cvor);
        return NULL;
34     }

    /* Inicijalizuju se polja u novom cvoru */
    strcpy(novi_cvor->rec, rec);
36     novi_cvor->brojac = 1;
    novi_cvor->levo = NULL;
38     novi_cvor->desno = NULL;
40

    /* Vraca se adresa novog cvora */
    return novi_cvor;
42 }
44

46 /* Funkcija koja dodaje novu rec u stablo. Ako je dodavanje uspesno
    povratna vrednost je 0, u suprotnom povratna vrednost je 1. */
48 int dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
    {
50     /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
52     /* Kreira se cvor koji sadrzi zadatu rec */
        Cvor *novi_cvor = napravi_cvor(rec);
54     /* Proverava se uspesnost kreiranja novog cvora */
        if (novi_cvor == NULL) {
56     /* Ako je doslo do greske, vraca se odgovarajuca vrednost */
            return 1;
58     }

60     /* Inace, novi cvor se proglašava korenom stabla */
        *adresa_korena = novi_cvor;
62

64     /* I vraca se indikator uspesnog dodavanja */
        return 0;
66     }

68     /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novu
        rec */

70     /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
        levo podstablo */
72     if (strcmp(rec, (*adresa_korena)->rec) < 0)
        return dodaj_u_stablo(&(*adresa_korena)->levo, rec);
74     else
        /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
        desno podstablo */
76     if (strcmp(rec, (*adresa_korena)->rec) > 0)
78     return dodaj_u_stablo(&(*adresa_korena)->desno, rec);
```

```
80     else {
81         /* Ako je rec jednaka reci u korenu, uvecava se njen broj
82            pojavljivanja */
83         (*adresa_korena)->brojac++;
84
85         /* I vraca se indikator uspesnog dodavanja */
86         return 0;
87     }
88 }
89
90 /* Funkcija koja oslobadja memoriju zauzetu stablom */
91 void oslobodi_stablo(Cvor ** adresa_korena)
92 {
93     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
94     if (*adresa_korena == NULL)
95         return;
96
97     /* Oslobadja se memorija zauzeta levim podstablom */
98     oslobodi_stablo(&(*adresa_korena)->levo);
99
100    /* Oslobadja se memorija zauzeta desnim podstablom */
101    oslobodi_stablo(&(*adresa_korena)->desno);
102
103    /* Oslobadja se memorija zauzeta korenom */
104    free((*adresa_korena)->rec);
105    free(*adresa_korena);
106
107    /* Stablo se proglašava praznim */
108    *adresa_korena = NULL;
109 }
110
111 /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec
112    (rec sa najvećim brojem pojavljivanja) */
113 Cvor *nadj_i_najfrekventniju_rec(Cvor * koren)
114 {
115     Cvor *max, *max_levo, *max_desno;
116
117     /* Ako je stablo prazno, prekida se sa pretragom */
118     if (koren == NULL)
119         return NULL;
120
121     /* Pronalazi se najfrekventnija rec u levom podstablu */
122     max_levo = nadj_i_najfrekventniju_rec(koren->levo);
123
124     /* Pronalazi se najfrekventnija rec u desnom podstablu */
125     max_desno = nadj_i_najfrekventniju_rec(koren->desno);
126
127     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
128        podstabla, korena i desnog podstabla */
129     max = koren;
130     if (max_levo != NULL && max_levo->brojac > max->brojac)
131         max = max_levo;
```

## 4 Dinamičke strukture podataka

---

```
132     if (max_desno != NULL && max_desno->brojac > max->brojac)
133         max = max_desno;
134     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
135     return max;
136 }
137
138 /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
139    pracone brojem pojavljivanja */
140 void prikazi_stablo(Cvor * koren)
141 {
142     /* Ako je stablo prazno, završava se sa ispisom */
143     if (koren == NULL)
144         return;
145
146     /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz
147        levog podstabla */
148     prikazi_stablo(koren->levo);
149
150     /* Zatim rec iz korena */
151     printf("%s: %d\n", koren->rec, koren->brojac);
152
153     /* I nastavlja se sa ispisom reci iz desnog podstabla */
154     prikazi_stablo(koren->desno);
155 }
156
157 /* Funkcija ucitava sledeću rec iz zadate datoteke f i upisuje je u
158    niz rec. Maksimalna dužina reci je određena argumentom max.
159    Funkcija vraća EOF ako u datoteci nema više reci ili 0 u
160    suprotnom. Rec je niz malih ili velikih slova. */
161 int procitaj_rec(FILE * f, char rec[], int max)
162 {
163     /* Karakter koji se cita */
164     int c;
165
166     /* Indeks pozicije na koju se smesta procitani karakter */
167     int i = 0;
168
169     /* Sve dok ima mesta za još jedan karakter u nizu i dokle se god
170        nije stiglo do kraja datoteke */
171     while (i < max - 1 && (c = fgetc(f)) != EOF) {
172         /* Proverava se da li je procitani karakter slovo */
173         if (isalpha(c))
174             /* Ako jeste, smesta se u niz - pritom se vrši konverzija u
175                mala slova jer program treba da bude neosetljiv na razliku
176                izmedju malih i velikih slova */
177             rec[i++] = tolower(c);
178         else
179             /* Ako nije, proverava se da li je procitano barem jedno
180                slovo nove reci */
181             /* Ako jeste, prekida se sa citanjem */
182             if (i > 0)
```

```
        break;
184
    /* U suprotnom se ide na sledecu iteraciju */
186 }
188 /* Dodaje se na rec terminirajuca nula */
    rec[i] = '\0';
190
    /* Vraca se 0 ako je uspesno procitana rec, tj. EOF u suprotnom */
192 return i > 0 ? 0 : EOF;
    }
194
int main(int argc, char **argv)
196 {
    Cvor *koren = NULL, *max;
198 FILE *f;
    char rec[MAX];
200
    /* Proverava se da li je navedeno ime datoteke prilikom
202 pokretanja programa */
    if (argc < 2) {
204         fprintf(stderr, "Greska: Nedostaje ime ulazne datoteke!\n");
        exit(EXIT_FAILURE);
206     }

    /* Priprema se datoteka za citanje */
208 if ((f = fopen(argv[1], "r")) == NULL) {
210     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
        argv[1]);
212     exit(EXIT_FAILURE);
    }
214

    /* Ucitavaju se reci iz datoteke i smestaju u binarno stablo
216 pretrage uz proveru uspesnosti dodavanja */
    while (procitaj_rec(f, rec, MAX) != EOF) {
218         if (dodaj_u_stablo(&koren, rec) == 1) {
            fprintf(stderr, "Greska: Neuspelo dodavanje reci %s.\n", rec);
220             oslobodi_stablo(&koren);
            exit(EXIT_FAILURE);
222         }
    }
224

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

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

    /* Pronalazi se najfrekventnija rec */
232 max = najdi_najfrekventniju_rec(koren);
234
```

## 4 Dinamičke strukture podataka

```
236 /* Ako takve reci nema */
    if (max == NULL)
        /* Ispisuje se odgovarajuće obavestenje */
238     printf("U tekstu nema reci!\n");
    else
240     /* Inace, ispisuje se broj pojavljivanja reci */
        printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
242             max->rec, max->brojac);

244 /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);

246
    exit(EXIT_SUCCESS);
248 }
```

### Rešenje 4.16

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

19 /* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
21 {
    /* Alocira se memorija za novi cvor i proverava se uspesnost
23     alokacije */
    Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
25     if (novi_cvor == NULL)
        return NULL;
27
    /* Inicijalizuju se polja novog cvora */
29     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
    strcpy(novi_cvor->telefon, telefon);
31     novi_cvor->levo = NULL;
    novi_cvor->desno = NULL;
33
    /* Vraca se adresa novog cvora */
```

```

35     return novi_cvor;
36 }
37
38 /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo.
39    Ukoliko je dodavanje uspesno povratna vrednost funkcije je 0,
40    dok je u suprotnom povratna vrednost 1 */
41 int
42 dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
43               char *telefon)
44 {
45     /* Ako je stablo prazno */
46     if (*adresa_korena == NULL) {
47         /* Kreira se novi cvor */
48         Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
49         /* Proverava se uspesnost kreiranja novog cvora */
50         if (novi_cvor == NULL) {
51             /* Ako je doslo do greske, vraca se odgovarajuca vrednost */
52             return 1;
53         }
54
55         /* Inace, novi cvor se proglašava korenom stabla */
56         *adresa_korena = novi_cvor;
57
58         /* I vraca se indikator uspesnog dodavanja */
59         return 0;
60     }
61
62     /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
63        unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
64        treba da bude pretrazivacko po ovom polju.
65
66        Ako je zadato ime i prezime leksikografski manje od imena i
67        prezimena koje se nalazi u korenu, podaci se dodaju u levo
68        podstablo */
69     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
70         < 0)
71         return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
72                               telefon);
73     else
74         /* Ako je zadato ime i prezime leksikografski vece od imena i
75            prezimena sadržanog u korenu, podaci se dodaju u desno
76            podstablo */
77         if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
78             return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
79                                   telefon);
80
81     /* Pretostavka zadatka je da nema istih imena i prezimena u
82        datoteci, pa se sledeca naredba nikada neci ni izvršiti */
83     return 0;
84 }
85
86 /* Funkcija koja oslobadja memoriju zauzetu stablom */

```



## 4 Dinamičke strukture podataka

---

```
87 void oslobodi_stablo(Cvor ** adresa_korena)
88 {
89     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
90     if (*adresa_korena == NULL)
91         return;
92
93     /* Oslobadja se memorija zauzeta levim podstablom */
94     oslobodi_stablo(&(*adresa_korena)->levo);
95
96     /* Oslobadja se memorija zauzeta desnim podstablom */
97     oslobodi_stablo(&(*adresa_korena)->desno);
98
99     /* Oslobadja se memorija zauzeta korenom */
100    free(*adresa_korena);
101
102    /* Stablo se proglašava praznim */
103    *adresa_korena = NULL;
104 }
105
106 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
107 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
108    koristiti za proveru da li je stablo uspesno kreirano. */
109 void prikazi_stablo(Cvor * koren)
110 {
111     /* Ako je stablo prazno, završava se sa ispisom */
112     if (koren == NULL)
113         return;
114
115     /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
116        podstabla */
117     prikazi_stablo(koren->levo);
118
119     /* Zatim se ispisuju podaci iz korena */
120     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
121
122     /* I nastavlja se sa ispisom podataka iz desnog podstabla */
123     prikazi_stablo(koren->desno);
124 }
125
126 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
127    ime i prezime i broj telefona u odgovarajuće nizove. Vraca EOF
128    ako nema više kontakata ili 0 u suprotnom. Maksimalna dužina
129    imena i prezimena određena je konstantom MAX_IME_PREZIME, a
130    maksimalna dužina broja telefona konstantom MAX_CIFARA. */
131 int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
132 {
133     /* Karakter koji se cita */
134     int c;
135
136     /* Indeks pozicije na koju se smesta procitani karakter */
137     int i = 0;
```

```
139 /* Linije datoteke koje se obradjuju su formata Ime Prezime
    BrojTelefona */
141
142 /* Preskacu se eventualne praznine sa pocetka linije datoteke */
143 while ((c = fgetc(f)) != EOF && isspace(c));
144
145 /* Prvo procitano slovo se upisuje u ime i prezime */
146 if (!feof(f))
147     ime_i_prezime[i++] = c;
148
149 /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
    cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
    Pritom treba voditi racuna da li ima dovoljno mesta za
    smestanje procitanog karaktera i da se slucajno ne dodje do
    kraja datoteke */
150 while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
151     if (!isdigit(c))
152         ime_i_prezime[i++] = c;
153     else if (i > 0)
154         break;
155 }
156
157 /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
    blanko karaktera */
158 ime_i_prezime[--i] = '\0';
159
160 /* I pocinje se sa citanjem broja telefona */
161 i = 0;
162
163 /* Upisuje se cifra koja je vec procitana */
164 telefon[i++] = c;
165
166 /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
    karaktera cije prisustvo nije dozvoljeno u broju telefona */
167 while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
168     if (c == '/' || c == '-' || isdigit(c))
169         telefon[i++] = c;
170     else
171         break;
172
173 /* Upisuje se terminirajuca nula */
174 telefon[i] = '\0';
175
176 /* Vraca se 0 ako je uspesno procitan kontakt ili EOF u suprotnom
    */
177 return !feof(f) ? 0 : EOF;
178 }
179
180 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
    prezimenom */
181 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
182 {
```

## 4 Dinamičke strukture podataka

```
191  /* Ako je imenik prazan, završava se sa pretragom */
192  if (koren == NULL)
193      return NULL;

195  /* Ako je traženo ime i prezime sadržano u korenu, također se
196     završava sa pretragom */
197  if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
198      return koren;

199
201  /* Ako je zadato ime i prezime leksikografski manje od vrednosti
202     u korenu pretraga se nastavlja levo */
203  if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
204      return pretrazi_imenik(koren->levo, ime_i_prezime);
205  else
206      /* U suprotnom, pretraga se nastavlja desno */
207      return pretrazi_imenik(koren->desno, ime_i_prezime);
208  }

209  int main(int argc, char **argv)
210  {
211      char ime_datoteke[MAX_IME_DATOTEKE];
212      Cvor *koren = NULL;
213      Cvor *trazeni;
214      FILE *f;
215      char ime_i_prezime[MAX_IME_I_PREZIME];
216      char telefon[MAX_CIFARA];
217      char c;
218      int i;

219
220      /* Učitava se ime datoteke i vrši se njena priprema za citanje */
221      printf("Unesite ime datoteke: ");
222      scanf("%s", ime_datoteke);
223      getchar();
224      if ((f = fopen(ime_datoteke, "r")) == NULL) {
225          fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
226                  ime_datoteke);
227          exit(EXIT_FAILURE);
228      }

229
230      /* Citaju se podaci iz datoteke i smestaju u binarno stablo
231         pretrage uz proveru uspesnosti dodavanja */
232      while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
233          if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
234              fprintf(stderr,
235                      "Greska: Neuspelo dodavanje podataka za osobu %s.\n",
236                      ime_i_prezime);
237              oslobodi_stablo(&koren);
238              exit(EXIT_FAILURE);
239          }

240
241      /* Datoteka se zatvara */
242      fclose(f);
```

```

243 /* Omogucava se pretraga imenika */
244 while (1) {
245     /* Ucitava se ime i prezime */
246     printf("Unesite ime i prezime: ");
247     i = 0;
248     while ((c = getchar()) != '\n')
249         ime_i_prezime[i++] = c;
250     ime_i_prezime[i] = '\0';
251
252     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
253        pretraga */
254     if (strcmp(ime_i_prezime, "KRAJ") == 0)
255         break;
256
257     /* Inace se ispisuje rezultat pretrage */
258     trazeni = pretrazi_imenik(koren, ime_i_prezime);
259     if (trazeni == NULL)
260         printf("Broj nije u imeniku!\n");
261     else
262         printf("Broj je: %s \n", trazeni->telefon);
263 }
264
265 /* Oslobadja se memorija zauzeta imenikom */
266 oslobodi_stablo(&koren);
267
268 exit(EXIT_SUCCESS);
269 }

```

### 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 */

```

## 4 Dinamičke strukture podataka

---

```
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
    double matematika, double jezik)
23 {
    /* Alocira se memorija za novi cvor i proverava se uspesnost
25     alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
        return NULL;
29
    /* Inicijalizuju se polja strukture */
31     strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
33     novi->uspeh = uspeh;
    novi->matematika = matematika;
35     novi->jezik = jezik;
    novi->levo = NULL;
37     novi->desno = NULL;
39
    /* Vraca se adresa kreiranog cvora */
    return novi;
41 }
43
/* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo.
    Ukoliko je dodavanje uspesno, povratna vrednost funkcije je 0,
45     dok je u suprotnom povratna vrednost 1 */
int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
47     double uspeh, double matematika, double jezik)
{
49     /* Ako je stablo prazno */
    if (*koren == NULL) {
51         /* Kreira se novi cvor */
        Cvor *novi_cvor =
53             napravi_cvor(ime, prezime, uspeh, matematika, jezik);
        /* Proverava se uspesnost kreiranja novog cvora */
55         if (novi_cvor == NULL) {
            /* I ukoliko je doslo do greske, vraca se odgovarajuca
57             vrednost */
            return 1;
59         }
61
        /* Inace, novi cvor se proglašava korenom stabla */
        *koren = novi_cvor;
63
        /* I vraca se indikator uspesnog dodavanja */
65         return 0;
67     }
69
    /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
    sortirano po ukupnom broju poena */
    if (uspeh + matematika + jezik >
71         (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
        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 }
78
79 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
80 void oslobodi_stablo(Cvor ** koren)
81 {
82     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
83     if (*koren == NULL)
84         return;
85
86     /* Oslobadja se memorija zauzeta levim podstablom */
87     oslobodi_stablo(&(*koren)->levo);
88
89     /* Oslobadja se memorija zauzeta desnim podstablom */
90     oslobodi_stablo(&(*koren)->desno);
91
92     /* Oslobadja se memorija zauzeta korenom */
93     free(*koren);
94
95     /* Stablo se proglašava praznim */
96     *koren = NULL;
97 }
98
99
100 /* Funkcija ispisuje sadržaj stabla. Ukoliko je vrednost argumenta
101    položili jednaka 0, ispisuju se informacije o učenicima koji
102    nisu položili prijemni, a ako je vrednost argumenta različita od
103    nule, ispisuju se informacije o učenicima koji su položili
104    prijemni. */
105 void stampaj(Cvor * koren, int položili)
106 {
107     /* Stablo je prazno - prekida se sa ispisom */
108     if (koren == NULL)
109         return;
110
111     /* Stampaju se informacije iz levog podstabla */
112     stampaj(koren->levo, položili);
113
114     /* Stampaju se informacije iz korenog cvora */
115     if (položili && koren->matematika + koren->jezik >= 10)
116         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
117             koren->prezime, koren->uspeh, koren->matematika,
118             koren->jezik,
119             koren->uspeh + koren->matematika + koren->jezik);
120     else if (!položili && koren->matematika + koren->jezik < 10)
121         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
122             koren->prezime, koren->uspeh, koren->matematika,
123             koren->jezik,
124             koren->uspeh + koren->matematika + koren->jezik);
```

## 4 Dinamičke strukture podataka

```
125     /* Stampaju se informacije iz desnog podstabla */
127     stampaj(koren->desno, položili);
129 }
129 /* Funkcija koja određuje koliko studenata nije položilo prijemni
131    ispit */
131 int nisu_položili(Cvor * koren)
133 {
135     /* Ako je stablo prazno, broj onih koji nisu položili je 0 */
135     if (koren == NULL)
137         return 0;
139     /* Pretraga se vrši i u levom i u desnom podstablu. Ako uslov za
139        polaganje nije ispunjen za koreni cvor, broj studenata se
141        uvecava za 1 */
141     if (koren->matematika + koren->jezik < 10)
143         return 1 + nisu_položili(koren->levo) +
143             nisu_položili(koren->desno);
145     /* Inace, nastavlja se prebrojavanje u podstablima */
145     return nisu_položili(koren->levo) + nisu_položili(koren->desno);
147 }
149 int main(int argc, char **argv)
151 {
151     FILE *in;
151     Cvor *koren;
153     char ime[MAX], prezime[MAX];
153     double uspeh, matematika, jezik;
155     /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
157     in = fopen("prijemni.txt", "r");
157     if (in == NULL) {
159         fprintf(stderr,
159             "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
161         exit(EXIT_FAILURE);
163     }
165     /* Citaju se podaci i dodaju u stablo uz proveru uspesnosti
165        dodavanja */
165     koren = NULL;
167     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
167         &matematika, &jezik) != EOF) {
169         if (dodaj_u_stablo
169             (&koren, ime, prezime, uspeh, matematika, jezik)
171             == 1) {
171             fprintf(stderr,
173                 "Greska: Neuspelo dodavanje podataka za %s %s.\n",
173                 ime, prezime);
175             oslobodi_stablo(&koren);
175             exit(EXIT_FAILURE);

```

```

177     }
178 }
179
180 /* Zatvara se datoteka */
181 fclose(in);
182
183 /* Stampaju se prvo podaci o ucenicima koji su položili prijemni */
184 stampaj(koren, 1);
185
186 /* Linija se iscrtava samo ako postoje ucenici koji nisu položili
187 prijemni */
188 if (nisu_položili(koren) != 0)
189     printf("-----\n");
190
191 /* Stampaju se podaci o ucenicima koji nisu položili prijemni */
192 stampaj(koren, 0);
193
194 /* Oslobadja se memorija zauzeta stablom */
195 oslobodi_stablo(&koren);
196
197 exit(EXIT_SUCCESS);
198 }

```

### 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 rođenja 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;

```



## 4 Dinamičke strukture podataka

```
27  /* Inicijalizuju se polja strukture */
    strcpy(novi->ime, ime);
29  strcpy(novi->prezime, prezime);
    novi->dan = dan;
31  novi->mesec = mesec;
    novi->levo = NULL;
33  novi->desno = NULL;

35  /* Vraca se adresa novog cvora */
    return novi;
37  }

39  /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
    uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
41  dodavanje uspesno povratna vrednost funkcije je 0, u suprotnom
    povratna vrednost je 1 */
43  int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                    int dan, int mesec)
45  {
    /* Ako je stablo prazno */
47  if (*koren == NULL) {

49      /* Kreira se novi cvor */
        Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
51      /* Proverava se uspesnost kreiranja novog cvora */
        if (novi_cvor == NULL) {
53          /* I ukoliko je doslo do greske, vraca se odgovarajuca
            vrednost */
55          return 1;
        }
57      /* Inace, novi cvor se proglašava korenom stabla */
        *koren = novi_cvor;
59

        /* I vraca se indikator uspesnog dodavanja */
61        return 0;
    }

63

    /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
65    meseca */
    if (mesec < (*koren)->mesec)
67        return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan,
                            mesec);
69    else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
        return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan,
71                            mesec);
73    else
        return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan,
                            mesec);
75  }

77  /* Funkcija vrši pretragu stabla i vraca cvor sa traženim datumom */
    Cvor *pretrazi(Cvor * koren, int dan, int mesec)
```

```

79 {
81  /* Ako je stablo prazno, obustavlja se pretraga */
82  if (koren == NULL)
83      return NULL;
84
85  /* Ako je trazeni datum u korenu */
86  if (koren->dan == dan && koren->mesec == mesec)
87      /* Vraca se njegova vrednost */
88      return koren;
89
90  /* Ako je mesec trazenog datuma manji od meseca sadrzanog u
91  korenu ili ako su meseci isti, ali je dan trazenog datuma
92  manji od aktuelnog datuma, pretrazuje se levo podstablo. Pre
93  toga se svakako proverava da li leva grana postoji. Ako ne
94  postoji treba vratiti prvi sledeci, a to je bas vrednost
95  uocenog korena */
96  if (mesec < koren->mesec
97      || (mesec == koren->mesec && dan < koren->dan)) {
98      if (koren->levo == NULL)
99          return koren;
100     else
101         return pretrazi(koren->levo, dan, mesec);
102 }
103
104 /* Inace se nastavlja pretraga u desnom delu */
105 return pretrazi(koren->desno, dan, mesec);
106 }
107
108 /* Funkcija koja pronalazi najmanji datum u stablu */
109 Cvor *pronadji_najmanji_datum(Cvor * koren)
110 {
111     /* Ako je stablo prazno, obustavlja se pretraga */
112     if (koren == NULL)
113         return NULL;
114
115     /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
116     sadrzi najmanji datum */
117     if (koren->levo == NULL)
118         return koren;
119     else
120         /* Inace, trazi se manji datum u levom podstablu */
121         return pronadji_najmanji_datum(koren->levo);
122 }
123
124 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
125 */
126 void datum_u_nisku(int dan, int mesec, char datum[])
127 {
128     if (dan < 10) {
129         datum[0] = '0';
130         datum[1] = dan + '0';
131     } else {

```

## 4 Dinamičke strukture podataka

```
131     datum[0] = dan / 10 + '0';
132     datum[1] = dan % 10 + '0';
133 }
134 datum[2] = '.';
135
136 if (mesec < 10) {
137     datum[3] = '0';
138     datum[4] = mesec + '0';
139 } else {
140     datum[3] = mesec / 10 + '0';
141     datum[4] = mesec % 10 + '0';
142 }
143 datum[5] = '.';
144 datum[6] = '\\0';
145 }
146
147 /* Funkcija koja oslobadja memoriju zauzetu stablom */
148 void oslobodi_stablo(Cvor ** adresa_korena)
149 {
150     /* Stablo je prazno */
151     if (*adresa_korena == NULL)
152         return;
153
154     /* Ako postoji levo podstablo, oslobadja se memorija koju zauzima
155     */
156     if ((*adresa_korena)->levo)
157         oslobodi_stablo(&(*adresa_korena)->levo);
158
159     /* Ako postoji desno podstablo, oslobadja se memorija koju
160     zauzima */
161     if ((*adresa_korena)->desno)
162         oslobodi_stablo(&(*adresa_korena)->desno);
163
164     /* Oslobadja se memorija zauzeta korenom */
165     free(*adresa_korena);
166
167     /* Stablo se proglašava praznim */
168     *adresa_korena = NULL;
169 }
170
171 int main(int argc, char **argv)
172 {
173     FILE *in;
174     Cvor *koren;
175     Cvor *slavljenik;
176     char ime[MAX_NISKA], prezime[MAX_NISKA];
177     int dan, mesec;
178     char datum[7];
179
180     /* Provera da li je zadato ime ulazne datoteke */
181     if (argc < 2) {
182         /* Ako nije, ispisuje se poruka i prekida se sa izvršavanjem
```

```
183     programa */
184     fprintf(stderr, "Greska: Nedostaje ime ulazne datoteke!\n");
185     exit(EXIT_FAILURE);
186 }
187
188 /* Inace, priprema se datoteka za citanje */
189 in = fopen(argv[1], "r");
190 if (in == NULL) {
191     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
192             argv[1]);
193     exit(EXIT_FAILURE);
194 }
195
196 /* Stablo se popunjava podacima uz proveru uspesnosti dodavanja */
197 koren = NULL;
198 while (fscanf
199     (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
200     if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
201         fprintf(stderr,
202             "Greska: Neuspelo dodavanje podataka za %s %s.\n",
203             ime, prezime);
204         oslobodi_stablo(&koren);
205         exit(EXIT_FAILURE);
206     }
207
208 /* Zatvara se datoteka */
209 fclose(in);
210
211 /* Omogucuje se pretraga podataka */
212 while (1) {
213     /* Ucitava se novi datum */
214     printf("Unesite datum: ");
215     if (scanf("%d.%d.", &dan, &mesec) == EOF)
216         break;
217
218     /* Pretrazuje se stablo */
219     slavljenik = pretrazi(koren, dan, mesec);
220
221     /* Ispisuju se pronadjeni podaci */
222
223     /* Ako slavljenik nije pronadjeni, to moze znaci da: */
224     /* 1. Drvo je prazno */
225     if (slavljenik == NULL && koren == NULL) {
226         printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
227         continue;
228     }
229     /* 2. Posle datuma koji je unesen, nema podataka u stablu - u
230     ovom slucaju se pretraga vrši pocevši od naredne godine i
231     ispisuje se najmanji datum */
232     if (slavljenik == NULL) {
233         slavljenik = pronadji_najmanji_datum(koren);
234         datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
235     }
236 }
```

## 4 Dinamičke strukture podataka

```
235     printf("Slavljenik: %s %s %s\n", slavljenik->ime,
236           slavljenik->prezime, datum);
237     continue;
238 }
239
240 /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
241 /* 1. Pronadjeni su tacni podaci */
242 if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
243     printf("Slavljenik: %s %s\n", slavljenik->ime,
244           slavljenik->prezime);
245     continue;
246 }
247
248 /* 2. Pronadjeni su podaci o prvom sledecem rođendanu */
249 datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
250 printf("Slavljenik: %s %s %s\n", slavljenik->ime,
251       slavljenik->prezime, datum);
252 }
253
254 /* Oslobadja se memorija zauzeta stablom */
255 oslobodi_stablo(&koren);
256
257 exit(EXIT_SUCCESS);
258 }
```

### Rešenje 4.19

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```
#include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija koja proverava da li su dva stabla koja sadrže cele
8    brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
9    odnosno 0 ako nisu */
10 int identitet(Cvor * koren1, Cvor * koren2)
11 {
12     /* Ako su oba stabla prazna, identična su */
13     if (koren1 == NULL && koren2 == NULL)
14         return 1;
15
16     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
17        identična */
18     if (koren1 == NULL || koren2 == NULL)
19         return 0;
20
21     /* Ako su oba stabla neprazna i u korenima se nalaze različite
```

```
22     vrednosti, moze se zakljuciti da se razlikuju */
23     if (koren1->broj != koren2->broj)
24         return 0;

26     /* Inace, proverava se da li vazi identitet i levih i desnih
27        podstabala */
28     return (identitet(koren1->levo, koren2->levo)
29             && identitet(koren1->desno, koren2->desno));
30 }

32 int main()
33 {
34     int broj;
35     Cvor *koren1, *koren2;

36     /* Ucitavaju se elementi prvog stabla */
37     koren1 = NULL;
38     printf("Prvo stablo: ");
39     scanf("%d", &broj);
40     while (broj != 0) {
41         if (dodaj_u_stablo(&koren1, broj) == 1) {
42             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
43                     broj);
44             oslobodi_stablo(&koren1);
45             exit(EXIT_FAILURE);
46         }
47         scanf("%d", &broj);
48     }

50     /* Ucitavaju se elementi drugog stabla */
51     koren2 = NULL;
52     printf("Drugo stablo: ");
53     scanf("%d", &broj);
54     while (broj != 0) {
55         if (dodaj_u_stablo(&koren2, broj) == 1) {
56             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
57                     broj);
58             oslobodi_stablo(&koren2);
59             exit(EXIT_FAILURE);
60         }
61         scanf("%d", &broj);
62     }

64     /* Poziva se funkcija koja ispituje identitet stabala i ispisuje
65        se njen rezultat */
66     if (identitet(koren1, koren2))
67         printf("Stabla jesu identicna.\n");
68     else
69         printf("Stabla nisu identicna.\n");

72     /* Oslobadja se memorija zauzeta stablima */
    oslobodi_stablo(&koren1);
```

## 4 Dinamičke strukture podataka

```
74  oslobodi_stablo(&koren2);
76  exit(EXIT_SUCCESS);
}
```

### Rešenje 4.20

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

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

4 /* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

6
8 /* Funkcija kreira novo stablo identično stablu koje je dato
   korenom. Povratna vrednost funkcije je 0 ukoliko je kopiranje
   uspesno, odnosno 1 ukoliko je doslo do greske */
10 int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
   {
12     /* Izlaz iz rekurzije */
   if (koren == NULL) {
14         *duplikat = NULL;
   return 0;
16     }

18     /* Duplira se koren stabla i postavlja da bude koren novog stabla
   */
20     *duplikat = napravi_cvor(koren->broj);
   if (*duplikat == NULL) {
22         return 1;
   }

24
26     /* Rekurzivno se dupliraju levo i desno podstablo i njihove adrese
   se cuvaju redom u pokazivacima na levo i desno podstablo korena
   duplikata */
28     int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
   int kopija_desno =
30         kopiraj_stablo(koren->desno, &(*duplikat)->desno);

32     /* Ako je uspesno duplirano i levo i desno podstablo */
   if (kopija_levo == 0 && kopija_desno == 0)
34         /* Uspesno je duplirano i celo stablo */
   return 0;
36     /* Inace, prijavljuje se da je doslo do greske */
   return 1;
38 }

40 /* Funkcija izracunava uniju dva skupa predstavljena stablima -
   rezultujući skup tj. stablo se dobija modifikacijom prvog
```

```

42     stabla. Povratna vrednost funkcije je 0 ukoliko je kreiranje
      unije uspesno, odnosno 1 ukoliko je doslo do greske */
44 int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
    {
46     /* Ako drugo stablo nije prazno */
      if (koren2 != NULL) {
48         /* 1. Dodaje se njegov koren u prvo stablo */
          if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
50             return 1;
          }
52
          /* 2. Rekurzivno se racuna unija levog i desnog podstabla
54             drugog stabla sa prvim stablom */
          int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
56         int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);
58
          /* Ako je unija podstabala uspesno kreirana */
          if (unija_levo == 0 && unija_desno == 0)
60             /* Uspesno je kreirana i unija stabala */
              return 0;
62
          /* U suprotnom se prijavljuje da je doslo do greske */
          return 1;
64     }
66
        /* Ako je drugo stablo prazno, nista se ne preduzima */
68     return 0;
    }
70
    /* Funkcija izracunava presek dva skupa predstavljana stablima -
72     rezultujući skup tj. stablo se dobija modifikacijom prvog
      stabla. Povratna vrednost funkcije je 0 ukoliko je kreiranje
74     preseka uspesno, odnosno 1 ukoliko je doslo do greske */
    int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
    {
76     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
      if (*adresa_korena1 == NULL)
78         return 0;
80
          /* Inace, kreira se presek levog i desnog podstabla sa drugim
82             stablom, tj. iz levog i desnog podstabla prvog stabla brisu se
              svi oni elementi koji ne postoje u drugom stablu */
84     int presek_levo =
          kreiraj_presek(&(*adresa_korena1)->levo, koren2);
86     int presek_desno =
          kreiraj_presek(&(*adresa_korena1)->desno, koren2);
88     if (presek_levo == 0 && presek_desno == 0) {
          /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se
90             on uklanja iz prvog stabla */
          if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
92             obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
    }

```



## 4 Dinamičke strukture podataka

```
94     /* U ovom slucaju je presek stabala uspesno kreiran */
95     return 0;
96 }
97 /* Inace, prijavljuje se da je doslo do greske */
98 return 1;
99 }
100
101 /* Funkcija izracunava razliku dva skupa predstavljena stablima -
102 rezultujući skup tj. stablo se dobija modifikacijom prvog
103 stabla. Povratna vrednost funkcije je 0 ukoliko je kreiranje
104 razlike uspesno, odnosno 1 ukoliko je doslo do greske */
105 int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
106 {
107     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
108     if (*adresa_korena1 == NULL)
109         return 0;
110
111     /* Inace, kreira se razlika levog i desnog podstabla sa drugim
112 stablom, tj. iz levog i desnog podstabla prvog stabla se brisu
113 svi oni elementi koji postoje i u drugom stablu */
114     int razlika_levo =
115         kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
116     int razlika_desno =
117         kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
118     if (razlika_levo == 0 && razlika_desno == 0) {
119         /* Ako se koren prvog stabla nalazi i u drugom stablu tada se
120 on uklanja se iz prvog stabla */
121         if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
122             obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
123
124         /* Razlika stabala je uspesno kreirana */
125         return 0;
126     }
127
128     /* Inace, prijavljuje se da je doslo do greske */
129     return 1;
130 }
131
132 int main()
133 {
134     Cvor *skup1;
135     Cvor *skup2;
136     Cvor *pomocni_skup = NULL;
137     int n;
138
139     /* Ucitavaju se elementi prvog skupa */
140     skup1 = NULL;
141     printf("Prvi skup: ");
142     while (scanf("%d", &n) != EOF) {
143         if (dodaj_u_stablo(&skup1, n) == 1) {
144             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
145             oslobodi_stablo(&skup1);
146         }
147     }
148 }
```

```
146     exit(EXIT_FAILURE);
147 }
148 }
149
150 /* Ucitavaju se elementi drugog skupa */
151 skup2 = NULL;
152 printf("Drugi skup: ");
153 while (scanf("%d", &n) != EOF) {
154     if (dodaj_u_stablo(&skup2, n) == 1) {
155         fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
156         oslobodi_stablo(&skup2);
157         exit(EXIT_FAILURE);
158     }
159 }
160
161 /* Kreira se unija skupova. Pre svega, napravi se kopija prvog
162 skupa kako bi se polazni skup mogao iskoristiti i za preostale
163 operacije */
164 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
165     oslobodi_stablo(&skup1);
166     oslobodi_stablo(&pomocni_skup);
167     exit(EXIT_FAILURE);
168 }
169 if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
170     oslobodi_stablo(&pomocni_skup);
171     oslobodi_stablo(&skup2);
172     exit(EXIT_FAILURE);
173 }
174 printf("Unija: ");
175 ispisi_stablo_infiksno(pomocni_skup);
176 putchar('\n');
177
178 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
179 prethodne operacije */
180 oslobodi_stablo(&pomocni_skup);
181
182 /* Kreira se presek skupova. Prvo se napravi kopija prvog skupa
183 kako bi se polazni skup mogao iskoristiti i za preostale
184 operacije */
185 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
186     oslobodi_stablo(&skup1);
187     oslobodi_stablo(&pomocni_skup);
188     exit(EXIT_FAILURE);
189 }
190 if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
191     oslobodi_stablo(&pomocni_skup);
192     oslobodi_stablo(&skup2);
193     exit(EXIT_FAILURE);
194 }
195 printf("Presek: ");
196 ispisi_stablo_infiksno(pomocni_skup);
197 putchar('\n');
```

## 4 Dinamičke strukture podataka

```
198      /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
200         prethodne operacije */
      oslobodi_stablo(&pomocni_skup);

202
      /* Kreira se razlika skupova. Prvo se napravi kopija prvog skupa
204         kako bi se polazni skup mogao iskoristiti i za preostale
         operacije */
206      if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
          oslobodi_stablo(&skup1);
208          oslobodi_stablo(&pomocni_skup);
          exit(EXIT_FAILURE);
210      }
      if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
212          oslobodi_stablo(&pomocni_skup);
          oslobodi_stablo(&skup2);
214          exit(EXIT_FAILURE);
      }
216      printf("Razlika: ");
      ispisi_stablo_infiksno(pomocni_skup);
218      putchar('\n');

220      /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
         prethodne operacije */
222      oslobodi_stablo(&pomocni_skup);

224      /* Oslobadja se memorija zauzeta polaznim skupovima */
      oslobodi_stablo(&skup1);
226      oslobodi_stablo(&skup2);

228      exit(EXIT_SUCCESS);
  }
```

### Rešenje 4.21

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
7  #define MAX 50
9  /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
   cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11  su smestene u niz */
   int kreiraj_niz(Cvor * koren, int a[])
13  {
```

```

15     int r, s;
16
17     /* Stablo je prazno - u niz je smesteno 0 elemenata */
18     if (koren == NULL)
19         return 0;
20
21     /* Dodaju se u niz elementi iz levog podstabla */
22     r = kreiraj_niz(koren->levo, a);
23
24     /* Tekuca vrednost promenljive r je broj elemenata koji su
25        upisani u niz i na osnovu nje se moze odrediti indeks novog
26        elementa */
27
28     /* Smesta se vrednost iz korena */
29     a[r] = koren->broj;
30
31     /* Dodaju se elementi iz desnog podstabla */
32     s = kreiraj_niz(koren->desno, a + r + 1);
33
34     /* Racuna se indeks na koji treba smestiti naredni element */
35     return r + s + 1;
36 }
37
38 /* Funkcija sortira niz tako sto najpre elemente niza smesti u
39    stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
40    desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
41    kreiran i sortiran, a 1 ukoliko je doslo do greske.
42
43    Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu"
44    kao sto je to slucaj sa ostalim opisanim algoritmima sortiranja
45    jer se sortiranje vrshi u pomocnoj dinamicnoj strukturi, a ne
46    razmenom elemenata niza. */
47 int sortiraj(int a[], int n)
48 {
49     int i;
50     Cvor *koren;
51
52     /* Kreira se stablo smestanjem elemenata iz niza u stablo */
53     koren = NULL;
54     for (i = 0; i < n; i++) {
55         if (dodaj_u_stablo(&koren, a[i]) == 1) {
56             oslobodi_stablo(&koren);
57             return 1;
58         }
59     }
60     /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u
61        niz a */
62     kreiraj_niz(koren, a);
63
64     /* Stablo vise nije potrebno pa se oslobadja zauzeta memorija */
65     oslobodi_stablo(&koren);

```

## 4 Dinamičke strukture podataka

```
67  /* Vraca se indikator uspesnog sortiranja */
    return 0;
}

69
int main()
71 {
    int a[MAX];
73     int n, i;

75     /* Ucitavaju se dimenzija i elementi niza */
    printf("n: ");
77     scanf("%d", &n);
    if (n < 0 || n > MAX) {
79         printf("Greska: Pogresna dimenzija niza!\n");
        exit(EXIT_FAILURE);
81     }

83     printf("a: ");
    for (i = 0; i < n; i++)
85         scanf("%d", &a[i]);

87     /* Poziva se funkcija za sortiranje */
    if (sortiraj(a, n) == 0) {
89         /* Ako je niz uspesno sortiran, ispisuje se rezultujući niz */
        for (i = 0; i < n; i++)
91             printf("%d ", a[i]);
        printf("\n");
93     } else {
        /* Inace, obavestava se korisnik da je doslo do greske */
95         printf("Greska: Problem prilikom sortiranja niza!\n");
    }

97     exit(EXIT_SUCCESS);
99 }
```

### Rešenje 4.22

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

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

4  /* Uključuje se biblioteka za rad sa stablima */
    #include "stabla.h"

6

/* a) Funkcija koja izračunava broj cvorova stabla */
8 int broj_cvorova(Cvor * koren)
{
10     /* Ako je stablo prazno, broj cvorova je 0 */
    if (koren == NULL)
```

```
12     return 0;
14     /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova
16        u levom podstablu, broja cvorova u desnom podstablu i 1, zato
18        sto treba racunati i koren */
19     return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
20 }
21
22 /* b) Funkcija koja izracunava broj listova stabla */
23 int broj_listova(Cvor * koren)
24 {
25     /* Ako je stablo prazno, broj listova je nula */
26     if (koren == NULL)
27         return 0;
28
29     /* Proverava se da li je tekuci cvor list */
30     if (koren->levo == NULL && koren->desno == NULL)
31         /* Ako jeste vraća se 1 - ova vrednost će kasnije zbog
32            rekurzivnih poziva uvećati broj listova za 1 */
33         return 1;
34
35     /* U suprotnom se prebrojavaju listovi koje se nalaze u
36        podstablama */
37     return broj_listova(koren->levo) + broj_listova(koren->desno);
38 }
39
40 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
41 void pozitivni_listovi(Cvor * koren)
42 {
43     /* Slučaj kada je stablo prazno */
44     if (koren == NULL)
45         return;
46
47     /* Ako je cvor list i sadrži pozitivnu vrednost */
48     if (koren->levo == NULL && koren->desno == NULL
49         && koren->broj > 0)
50         /* Stampa se */
51         printf("%d ", koren->broj);
52
53     /* Nastavlja se sa stampanjem pozitivnih listova u podstablama */
54     pozitivni_listovi(koren->levo);
55     pozitivni_listovi(koren->desno);
56 }
57
58 /* d) Funkcija koja izracunava zbir cvorova stabla */
59 int zbir_svih_cvorova(Cvor * koren)
60 {
61     /* Ako je stablo prazno, zbir cvorova je 0 */
62     if (koren == NULL)
63         return 0;
64
65     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i
```

## 4 Dinamičke strukture podataka

```
64     svih elemenata u podstablama */
65     return koren->broj + zbir_svih_cvorova(koren->levo) +
66         zbir_svih_cvorova(koren->desno);
67 }
68
69 /* e) Funkcija koja izracunava najveći element stabla */
70 Cvor *najveći_element(Cvor * koren)
71 {
72     /* Ako je stablo prazno, obustavlja se pretraga */
73     if (koren == NULL)
74         return NULL;
75
76     /* Zbog prirode pretraživačkog stabla, vrednosti veće od korena
77        se nalaze u desnom podstablu */
78
79     /* Ako desnog podstabla nema */
80     if (koren->desno == NULL)
81         /* Najveća vrednost je koren */
82         return koren;
83
84     /* Inače, najveća vrednost se traži desno */
85     return najveći_element(koren->desno);
86 }
87
88 /* f) Funkcija koja izracunava dubinu stabla */
89 int dubina_stabla(Cvor * koren)
90 {
91     /* Dubina praznog stabla je 0 */
92     if (koren == NULL)
93         return 0;
94
95     /* Izracunava se dubina levog podstabla */
96     int dubina_levo = dubina_stabla(koren->levo);
97
98     /* Izracunava se dubina desnog podstabla */
99     int dubina_desno = dubina_stabla(koren->desno);
100
101     /* Dubina stabla odgovara većoj od dubina podstabala - 1 se
102        dodaje jer se računa i koren stabla */
103     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     /* Ako je stablo prazno, broj cvorova je 0 */
111     if (koren == NULL)
112         return 0;
113
114     /* Ako se stiglo do traženog nivoa, vraća se 1. Ova vrednost će
115        kasnije zbog rekurzivnih poziva uvećati broj cvorova za 1. */
```

```
116     if (i == 0)
117         return 1;
118
119     /* Inace, nastavlja se prebrojavanje na nivou nize i u levom i u
120        desnom postablu */
121     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
122        + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
123 }
124
125 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
126 void ispis_nivo(Cvor * koren, int i)
127 {
128     /* Ako je stablo prazno, nista se ne ispisuje */
129     if (koren == NULL)
130         return;
131
132     /* Ako se stiglo do trazenog nivoa, ispisuje se vrednost */
133     if (i == 0) {
134         printf("%d ", koren->broj);
135         return;
136     }
137     /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
138        desnom podstablu */
139     ispis_nivo(koren->levo, i - 1);
140     ispis_nivo(koren->desno, i - 1);
141 }
142
143 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
144    stabla */
145 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
146 {
147     /* Ako je stablo prazno, obustavlja se pretraga */
148     if (koren == NULL)
149         return NULL;
150
151     /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
152     if (i == 0)
153         return koren;
154
155     /* Pronalazi se maksimum na i-tom nivou levog podstabla */
156     Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
157
158     /* Pronalazi se maksimum na i-tom nivou desnog podstabla */
159     Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);
160
161     /* Trazi se i vraca maksimum izracunatih vrednosti */
162     if (a == NULL && b == NULL)
163         return NULL;
164     if (a == NULL)
165         return b;
166     if (b == NULL)
167         return a;
```



## 4 Dinamičke strukture podataka

```
168  /* Ako su obe vrednosti razlicite od NULL, veca od vrednosti se
169      nalazi u b cvoru jer je stablo pretrazivacko */
170  return b;
171  }
172
173  /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
174  int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
175  {
176      /* Ako je stablo prazno, zbir je 0 */
177      if (koren == NULL)
178          return 0;
179
180      /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
181      if (i == 0)
182          return koren->broj;
183
184      /* Inace, spustanje se nastavlja za jedan nivo nize i
185          izracunavaju se sume levog i desnog podstabla */
186      return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
187          + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
188  }
189
190  /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
191      manje ili jednake od date vrednosti x */
192  int zbir_manjih_od_x(Cvor * koren, int x)
193  {
194      /* Ako je stablo prazno, zbir je 0 */
195      if (koren == NULL)
196          return 0;
197
198      /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
199          prirode pretrazivackog stabla treba obici i levo i desno
200          podstablo */
201      if (koren->broj <= x)
202          return koren->broj + zbir_manjih_od_x(koren->levo, x) +
203              zbir_manjih_od_x(koren->desno, x);
204
205      /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
206          medju njima jedino moze biti onih koje zadovoljavaju uslov */
207      return zbir_manjih_od_x(koren->levo, x);
208  }
209
210  int main(int argc, char **argv)
211  {
212      /* Analiziraju se argumenti komandne linije */
213      if (argc != 3) {
214          fprintf(stderr,
215              "Greska: Program se poziva sa: ./a.out nivo
216              broj_za_pretragu\n");
217          exit(EXIT_FAILURE);
218      }
219      int i = atoi(argv[1]);
```

```
220     int x = atoi(argv[2]);
222     /* Kreira se stablo uz proveru uspesnosti dodavanja novih
223        vrednosti */
224     Cvor *koren = NULL;
225     int broj;
226     while (scanf("%d", &broj) != EOF) {
227         if (dodaj_u_stablo(&koren, broj) == 1) {
228             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
229                 broj);
230             oslobodi_stablo(&koren);
231             exit(EXIT_FAILURE);
232         }
233     }
234     /* Ispisuju se rezultati rada funkcija */
235     printf("Broj cvorova: %d\n", broj_cvorova(koren));
236     printf("Broj listova: %d\n", broj_listova(koren));
237     printf("Pozitivni listovi: ");
238     pozitivni_listovi(koren);
239     printf("\n");
240     printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
241     if (najveci_element(koren) == NULL)
242         printf("Najveci element: ne postoji\n");
243     else
244         printf("Najveci element: %d\n", najveci_element(koren)->broj);
245
246     printf("Dubina stabla: %d\n", dubina_stabla(koren));
247
248     printf("Broj cvorova na %d. nivou: %d\n", i,
249         broj_cvorova_na_itom_nivou(koren, i));
250     printf("Elementi na %d. nivou: ", i);
251     ispis_nivo(koren, i);
252     printf("\n");
253     if (najveci_element_na_itom_nivou(koren, i) == NULL)
254         printf("Nema elemenata na %d. nivou!\n", i);
255     else
256         printf("Maksimalni element na %d. nivou: %d\n", i,
257             najveci_element_na_itom_nivou(koren, i)->broj);
258
259     printf("Zbir elemenata na %d. nivou: %d\n", i,
260         zbir_cvorova_na_itom_nivou(koren, i));
261     printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
262         zbir_manjih_od_x(koren, x));
263
264     /* Oslobadja se memorija zauzeta stablom */
265     oslobodi_stablo(&koren);
266     exit(EXIT_SUCCESS);
267 }
268 }
```

### Rešenje 4.23

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```
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 izračunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
9  {
10     /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
12         return 0;
13
14     /* Izračunava se dubina levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
16
17     /* Izračunava se dubina desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);
19
20     /* Dubina stabla odgovara većoj od dubina podstabala uvećanoj za
21     1, jer se računa i koren stabla. */
22     return dubina_levo >
23         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }
25
26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
27 void ispisi_nivo(Cvor * koren, int i)
28 {
29     /* Ako nema više cvorova, nema ni spustanja niz stablo */
30     if (koren == NULL)
31         return;
32
33     /* Ako se stiglo do traženog nivoa, ispisuje se vrednost. */
34     if (i == 0) {
35         printf("%d ", koren->broj);
36         return;
37     }
38     /* Inace, vrši se spustanje za jedan nivo nize i u levom i u
39     desnom podstablu */
40     ispisi_nivo(koren->levo, i - 1);
41     ispisi_nivo(koren->desno, i - 1);
42 }
43
44 /* Funkcija koja ispisuje stablo po nivoima */
45 void ispisi_stablo_po_nivoima(Cvor * koren)
46 {
47     int i;
```

```

48  /* Prvo se izracunava dubina stabla */
49  int dubina;
50  dubina = dubina_stabla(koren);
51
52  /* Zatim se ispisuje nivo po nivo stabla */
53  for (i = 0; i < dubina; i++) {
54      printf("%d. nivo: ", i);
55      ispisi_nivo(koren, i);
56      printf("\n");
57  }
58  }
59
60  int main(int argc, char **argv)
61  {
62      Cvor *koren;
63      int broj;
64
65      /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
66       uspesnosti dodavanja */
67      koren = NULL;
68      while (scanf("%d", &broj) != EOF) {
69          if (dodaj_u_stablo(&koren, broj) == 1) {
70              fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
71                      broj);
72              oslobodi_stablo(&koren);
73              exit(EXIT_FAILURE);
74          }
75      }
76  }
77
78  /* Ispisuje se stablo po nivoima */
79  ispisi_stablo_po_nivoima(koren);
80
81  /* Oslobadja se memorija zauzeta stablom */
82  oslobodi_stablo(&koren);
83
84  exit(EXIT_SUCCESS);
85  }

```

### Rešenje 4.25

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija koja izracunava dubinu stabla */

```

## 4 Dinamičke strukture podataka

---

```
int dubina_stabla(Cvor * koren)
9 {
11     /* Dubina praznog stabla je 0 */
13     if (koren == NULL)
15         return 0;
17     /* Izracunava se dubina levog podstabla */
19     int dubina_levo = dubina_stabla(koren->levo);
21     /* Izracunava se dubina desnog podstabla */
23     int dubina_desno = dubina_stabla(koren->desno);
25     /* Dubina stabla odgovara vecoj od dubina podstabala uvecanoj za
27     1, jer se racuna i koren. */
29     return dubina_levo >
31         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
33 }
35
37 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
39 stablo */
41 int avl(Cvor * koren)
43 {
45     int dubina_levo, dubina_desno;
47     /* Ako je stablo prazno, zaustavlja se brojanje */
49     if (koren == NULL) {
51         return 0;
53     }
55     /* Izracunava se dubina levog podstabla korena */
57     dubina_levo = dubina_stabla(koren->levo);
59     /* Izracunava se dubina desnog podstabla korena */
61     dubina_desno = dubina_stabla(koren->desno);
63     /* Ako je uslov za AVL stablo ispunjen */
65     if (abs(dubina_desno - dubina_levo) <= 1) {
67         /* Racuna se broj AVL cvorova u levom i desnom podstablu i
69         uvecava za jedan iz razloga sto koren ispunjava uslov */
71         return 1 + avl(koren->levo) + avl(koren->desno);
73     } else {
75         /* Inace, racuna se samo broj AVL cvorova u podstablama */
77         return avl(koren->levo) + avl(koren->desno);
79     }
81 }
83
85 int main(int argc, char **argv)
87 {
89     Cvor *koren;
91     int broj;
93
95     /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
```

```

    uspesnosti dodavanja */
61 koren = NULL;
while (scanf("%d", &broj) != EOF) {
63     if (dodaj_u_stablo(&koren, broj) == 1) {
        fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
65             broj);
        oslobodi_stablo(&koren);
67         exit(EXIT_FAILURE);
    }
69 }

71 /* Racuna se i ispisuje broj AVL cvorova */
printf("%d\n", avl(koren));
73
75 /* Oslobadja se memorija zauzeta stablom */
oslobodi_stablo(&koren);
77
exit(EXIT_SUCCESS);
}

```

### Rešenje 4.26

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"
6
8  /* Funkcija koja kreira stablo prema zadatoj slici. Povratna
   vrednost funkcije je 0 ako je stablo uspesno kreirano, odnosno 1
   ukoliko je doslo do greske */
10 int kreiraj_hip(Cvor ** adresa_korena)
   {
12     /* Stablo se proglašava praznim */
    *adresa_korena = NULL;
14
16     /* Dodaje se cvor po cvor uz proveru uspesnosti dodavanja */
    if (((*adresa_korena) = napravi_cvor(100)) == NULL)
        return 1;
18     if (((*adresa_korena)->levo = napravi_cvor(19)) == NULL)
        return 1;
20     if (((*adresa_korena)->levo->levo = napravi_cvor(17)) == NULL)
        return 1;
22     if (((*adresa_korena)->levo->levo->levo =
        napravi_cvor(2)) == NULL)
24         return 1;
    if (((*adresa_korena)->levo->levo->desno =
26         napravi_cvor(7)) == NULL)

```

## 4 Dinamičke strukture podataka

```
    return 1;
28  if (((*adresa_korena)->levo->desno = napravi_cvor(3)) == NULL)
    return 1;
30  if (((*adresa_korena)->desno = napravi_cvor(36)) == NULL)
    return 1;
32  if (((*adresa_korena)->desno->levo = napravi_cvor(25)) == NULL)
    return 1;
34  if (((*adresa_korena)->desno->desno = napravi_cvor(1)) == NULL)
    return 1;
36
37  /* Vraca se indikator uspesnog kreiranja */
38  return 0;
39  }
40
41  /* Funkcija proverava da li je zadato binarno stablo celih
42  pozitivnih brojeva hip. Ideja koja ce biti implementirana u
43  osnovi ima pronalazenje maksimalne vrednosti levog i maksimalne
44  vrednosti desnog podstabla. Ako je vrednost u korenu veca od
45  izracunatih vrednosti, uoceni fragment stabla zadovoljava uslov
46  za hip. Zato ce funkcija vratiti maksimalne vrednosti iz uocenog
47  podstabla ili vrednost -1 ukoliko se zakljuci da stablo nije
48  hip. */
49  int hip(Cvor * koren)
50  {
51      int max_levo, max_desno;
52
53      /* Prazno sablo je hip. Kao rezultat se vraca 0, kao najmanji
54      pozitivan broj */
55      if (koren == NULL) {
56          return 0;
57      }
58      /* Ukoliko je stablo list... */
59      if (koren->levo == NULL && koren->desno == NULL) {
60          /* Vraca se njegova vrednost */
61          return koren->broj;
62      }
63
64      /* Inace, proverava se svojstvo za levo podstablo */
65      max_levo = hip(koren->levo);
66
67      /* Proverava se svojstvo za desno podstablo */
68      max_desno = hip(koren->desno);
69
70      /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
71      ni celo stablo */
72      if (max_levo == -1 || max_desno == -1) {
73          return -1;
74      }
75
76      /* U suprotnom proverava se da li svojstvo vazii za uoceni cvor */
77      if (koren->broj > max_levo && koren->broj > max_desno) {
78          /* Ako vazii, vraca se vrednost korena */
```

```
    return koren->broj;
80 }

82 /* U suprotnom se zaključuje da stablo nije hip */
    return -1;
84 }

86 int main(int argc, char **argv)
{
88     Cvor *koren;
    int hip_indikator;

90     /* Kreira se stablo prema zadatoj slici */
92     if (kreiraj_hip(&koren) == 1) {
        fprintf(stderr, "Greska: Neuspesno kreiranje hipa.\n");
94         oslobodi_stablo(&koren);
        exit(EXIT_FAILURE);
96     }

98     /* Poziva se funkcija kojom se proverava da li je stablo hip */
    hip_indikator = hip(koren);

100

102     /* Ispisuje se rezultat */
    if (hip_indikator == -1) {
        printf("Zadato stablo nije hip!\n");
104     } else {
        printf("Zadato stablo je hip!\n");
106     }

108     /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);

110     exit(EXIT_SUCCESS);
112 }
```



Elektronsko izdanje (2016)

# Dodatak A

## Ispitni rokovi

### A.1 Praktični deo ispita, jun 2015.

**Zadatak A.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 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 standardni izlaz u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom. U slučaju pojave bilo kakve greške na standardni izlaz za greške ispisati vrednost  $-1$  i prekinuti izvršavanje programa.

#### Test 1

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

```
ULAZ.TXT  
5  
Programiranje  
Matematika  
12345  
dInAmiCnArEc  
Ispit
```

```
IZLAZ:  
Ispit  
Matematika  
Programiranje
```

#### Test 2

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

```
ULAZ.TXT  
2  
maksimalano  
poena
```

```
IZLAZ:
```

### Test 3

```
POKRETANJE: ./a.out ulaz.txt
DATOTEKA ULAZ.TXT NE POSTOJI
IZLAZ ZA GREŠKE:
-1
```

### Test 4

```
POKRETANJE: ./a.out
IZLAZ ZA GREŠKE:
-1
```

[Rešenje A.1]

**Zadatak A.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 koje smešta u stablo. Smeštanje brojeva se prekida učitavanjem nule. Zatim se ispisuje rezultat pozivanja funkcije `sumiraj_n` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za greške ispisati  $-1$ . NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima 4.14.*

### 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 A.2]

**Zadatak A.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 takve 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.

<pre> 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         </pre>	<pre> Test 2 ULAZ: 2 3 0 0 -5 1 2 -4 IZLAZ: 2         </pre>	<pre> Test 3 ULAZ: -2 IZLAZ ZA GREŠKE: -1         </pre>
--	--	--

[Rešenje A.3]

## A.2 Praktični deo ispita, jul 2015.

**Zadatak A.4** Napisati program koji kao prvi arugment komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske *needle* u nisci *haystack*, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

<pre> Test 1 POKRETANJE: ./a.out ulaz.txt test ULAZ.TXT Ovo je test primer. U njemu se rec test javlja vise puta. testtesttest IZLAZ: 5         </pre>	<pre> Test 2 POKRETANJE: ./a.out IZLAZ ZA GREŠKE: -1         </pre>
<pre> Test 3 POKRETANJE: ./a.out ulaz.txt foo DATOTEKA ULAZ.TXT NE POSTOJI IZLAZ ZA GREŠKE: -1         </pre>	<pre> Test 4 POKRETANJE: ./a.out ulaz.txt . DATOTEKA ULAZ.TXT JE PRAZNA IZLAZ: 0         </pre>

[Rešenje A.4]

**Zadatak A.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. U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Ne praviti nikakve pretpostavke o broju trouglova u datoteci i proveriti da li je datoteka ispravno zadata. UPUTSTVO: *Koristiti Heronov obrazac:*  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla.

Test 3

```
DATOTEKA TROUGLOVI.TXT NE POSTOJI
IZLAZ ZA GREŠKE:
-1
```

Test 4

```
TROUGLOVI.TXT
0
IZLAZ:
```

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 GREŠKE:
-1
```

[Rešenje A.5]

**Zadatak A.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. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*



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 A.8]

**Zadatak A.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 standardni izlaz 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 greške. NAPOMENA: *Koristiti biblioteku za rad sa celobrojnim matricama iz zadatka 2.19.*

Test 1

```

|| ULAZ:
|| 4
|| 1 2 3 4
|| 2 1 4 3
|| 3 4 2 1
|| 4 3 1 2
|| IZLAZ:
|| 10
    
```

Test 2

```

|| ULAZ:
|| 3
|| 1 1 1
|| 1 1 1
|| 1 1 1
|| IZLAZ:
|| 3
    
```

Test 3

```

|| ULAZ:
|| 2
|| 1 1
|| 2 2
|| IZLAZ:
|| -
    
```

<p><i>Test 4</i></p> <pre> ULAZ:  2  1 2  1 2 IZLAZ:  -                 </pre>	<p><i>Test 5</i></p> <pre> ULAZ:  1  5 IZLAZ:  5                 </pre>	<p><i>Test 6</i></p> <pre> ULAZ:  0 IZLAZ ZA GREŠKE:  -1                 </pre>
--	---	---

[Rešenje A.9]

## A.4 Praktični deo ispita, januar 2016.

**Zadatak A.10** Napisati funkciju `unsigned int zamena(unsigned int x)` koja u datom broju `x` menja mesta prvom i četvrtom bajtu. Prvi bajt je sačinjen od 8 bitova najmanje težine. Napisati program koji testira funkciju `zamena` za ceo broj unet sa standardnog ulaza. U slučaju da je uneti broj negativan, na standardni izlaz za greške program ispisuje `-1`, a inače ispisuje na standardni izlaz broj dobijen primenom funkcije `zamena`.

<p><i>Test 1</i></p> <pre> ULAZ: 285278344 IZLAZ: 2281766929                 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 1024 IZLAZ: 1024                 </pre>	<p><i>Test 3</i></p> <pre> ULAZ:  1 IZLAZ: 16777216                 </pre>
<p><i>Test 4</i></p> <pre> ULAZ:  0 IZLAZ:  0                 </pre>	<p><i>Test 5</i></p> <pre> ULAZ: -63 IZLAZ ZA GREŠKE: -1                 </pre>	

[Rešenje A.10]

**Zadatak A.11** Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeva. Napisati funkciju `int najduzi_put (Cvor * koren)` koja za dato stablo izračunava dužinu najdužeg puta od korena do nekog lista. Ako je stablo prazno, povratna vrednost funkcije je `-1`. Ako stablo ima samo koren, dužina najdužeg puta je `0`. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva iz zadatka 4.14.*



Test 1

```

|| ULAZ:
|| 10 5 15 3 2 4 30 12 14 13
|| IZLAZ:
|| 4
    
```

Test 2

```

|| ULAZ:
|| 3
|| IZLAZ:
|| 0
    
```

Test 3

```

|| ULAZ:
|| 5 6
|| IZLAZ:
|| 1
    
```

Test 4

```

|| ULAZ:
|| 7 5 8
|| IZLAZ:
|| 1
    
```

Test 5

```

|| ULAZ:
|| 5 7 8
|| IZLAZ:
|| 2
    
```

[Rešenje A.11]

**Zadatak A.12** Sa standardnog ulaza zadaje se ime datoteke u kojoj se nalazi matrica realnih brojeva jednostruke tačnosti i jedan realan broj. Napisati program koji iz datoteke učitava matricu realnih brojeva, a zatim pronalazi i na standardni izlaz ispisuje indeks vrste matrice u kojoj se uneti realan broj pojavljuje najmanje puta. Ako postoji više takvih vrsta, ispisati indeks prve takve vrste. U datoteci su prvo navedena dva cela broja koja predstavljaju dimenzije matrice, redom broj vrsta i broj kolona, a zatim i elementi matrice vrstu po vrstu. U slučaju greške ispisati  $-1$  na standardni izlaz za greške. Pretpostaviti da ime datoteke neće biti duže od 30 karaktera. NAPOMENA: *U zadatku treba koristiti dinamičku alokaciju memorije.*

Test 1

```

|| ULAZ:
|| brojevi.txt 0
||
|| BROJEVI.TXT
|| 4 4
|| 0 0 0 1.2
|| 1 0 0.3 0.3
|| 0.5 0.5 0.9 -1
|| -2 0 0 0
||
|| IZLAZ:
|| 2
    
```

Test 2

```

|| ULAZ:
|| in.txt 2
||
|| IN.TXT
|| 3 3
|| 2 0 2
|| -1 2 -1
|| 2 5 3
||
|| IZLAZ:
|| 1
    
```

Test 3

```

|| ULAZ:
|| brojevi.txt 12
||
|| DATOTEKA BROJEVI.TXT JE PRAZNA
||
|| IZLAZ ZA GREŠKE:
|| -1
    
```

[Rešenje A.12]

## A.5 Rešenja

## Rešenje A.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define MAKS 50
5
6 /* Funkcija vrši dinamičku alokaciju memorije potrebne za n niski
7    od kojih nijedna nije duža od MAKS karaktera. */
8 char **alociranje_memorije(int n)
9 {
10     char **linije = NULL;
11     int i, j;
12     /* Alocira se prostor za niz vrsti matrice */
13     linije = (char **) malloc(n * sizeof(char *));
14     /* U slučaju neuspješnog otvaranja ispisuje se -1 na standardni
15        izlaz za greške i program završava. */
16     if (linije == NULL)
17         return NULL;
18     /* Alocira se prostor za svaku vrstu matrice. Niska nije duža od
19        MAKS karaktera, a 1 se dodaje zbog terminirajuće nule. */
20     for (i = 0; i < n; i++) {
21         linije[i] = malloc((MAKS + 1) * sizeof(char));
22         /* Ako alokacija nije prošla uspešno, oslobadjaju se svi
23            prethodno alocirani resursi, i povratna vrednost je NULL */
24         if (linije[i] == NULL) {
25             for (j = 0; j < i; j++) {
26                 free(linije[j]);
27             }
28             free(linije);
29             return NULL;
30         }
31     }
32     return linije;
33 }
34
35 /* Funkcija oslobadja dinamički alociranu memoriju */
36 char **oslobadjanje_memorije(char **linije, int n)
37 {
38     int i;
39     /* Oslobadja se prostor rezervisan za svaku vrstu */
40     for (i = 0; i < n; i++) {
41         free(linije[i]);
42     }
43     /* Oslobadja se memorija za niz pokazivaca na vrste */
44     free(linije);
45
46     /* Matrica postaje prazna, tj. nealocirana */
```

```

48     return NULL;
49 }
50 int main(int argc, char *argv[])
51 {
52     FILE *ulaz;
53     char **linije;
54     int i, n;
55
56     /* Proverava argumenata komandne linije. */
57     if (argc != 2) {
58         fprintf(stderr, "-1\n");
59         exit(EXIT_FAILURE);
60     }
61
62     /* Otvara se datoteka cije ime je navedeno kao argument komandne
63        linije neposredno nakon imena programa koji se poziva. U
64        slucaju neuspesnog otvaranja ispisuje se -1 na standardni
65        izlaz za greske i program zavrшава izvršavanje. */
66     ulaz = fopen(argv[1], "r");
67     if (ulaz == NULL) {
68         fprintf(stderr, "-1\n");
69         exit(EXIT_FAILURE);
70     }
71     /* Ucitava se broj linija. */
72     fscanf(ulaz, "%d", &n);
73
74     /* Alociranje memorije na osnovu ucitanog broja linija. */
75     linije = alociranje_memorije(n);
76
77     /* U slucaju neuspesne alokacije ispisuje se -1 na standardni
78        izlaz za greske i program završava. */
79     if (linije == NULL) {
80         fprintf(stderr, "-1\n");
81         exit(EXIT_FAILURE);
82     }
83
84     /* Iz datoteke se ucita svih n linija. */
85     for (i = 0; i < n; i++) {
86         fscanf(ulaz, "%s", linije[i]);
87     }
88
89     /* Ispisu se 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     /* Oslobadja se memorija koja je dinamički alocirana. */
97     linije = oslobadjanje_memorije(linije, n);
98

```

```

100  /* Zatvara se datoteka. */
      fclose(ulaz);
102  exit(EXIT_SUCCESS);
    }

```

## Rešenje A.2

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

*main.c*

```

#include <stdio.h>
2  #include <stdlib.h>
#include "stabla.h"
4
int sumiraj_n(Cvor * koren, int n)
6  {
    /* Ako je stablo prazno, suma je nula */
8   if (koren == NULL)
        return 0;
10
    /* Ako je n jednako nula, vraća se broj iz korena */
12  if (n == 0)
        return koren->broj;
14  /* Inace, izracunava se suma na (n-1)-om nivou u levom podstablu,
    kao i suma na (n-1)-om nivou u desnom podstablu i vraća se zbir
16  te dve izracunate vrednosti jer predstavlja zbir svih cvorova na
    n-tom nivou u pocetnom stablu */
18  return sumiraj_n(koren->levo, n - 1)
        + sumiraj_n(koren->desno, n - 1);
20 }

22 int main()
    {
24     Cvor *koren = NULL;
        int n;
26     int nivo;

28     /* Ucitava se vrednost nivoa */
        scanf("%d", &nivo);
30     while (1) {
        scanf("%d", &n);
32         /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
            if (n == 0)
34                 break;
        /* Ako nije, dodaje se procitani broj u stablo. */
36     if (dodaj_u_stablo(&koren, n) == 1) {
        fprintf(stderr, "-1\n");
    }
}

```

## A Ispitni rokovi

```
38     oslobodi_stablo(&koren);
39     exit(EXIT_FAILURE);
40 }
41 }
42
43 /* Ispisuje se rezultat rada trazene funkcije */
44 printf("%d\n", sumiraj_n(koren, nivo));
45
46 /* Oslobadja se memorija */
47 oslobodi_stablo(&koren);
48
49 exit(EXIT_SUCCESS);
50 }
```

### Rešenje A.3

```
#include <stdio.h>
2 #include <stdlib.h>
3 #define MAKS 50
4
5 /* Funkcija ucitava elemente 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
17    m 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 /* Funkcija vraca indeks kolone matrice m u kojoj se nalazi najvise
30    negativnih elemenata */
31 int maks_indeks(int m[][MAKS], int v, int k)
32 {
33     int j;
34     int broj_negativnih;
35     /* Inicijalizacija na nulu indeksa kolone sa maksimalnim brojem
```

```
36     negativnih (maks_indeks_kolone), kao i maksimalnog broja
37     negativnih elemenata u nekoj koloni (maks_broj_negativnih) */
38     int maks_indeks_kolone = 0;
39     int maks_broj_negativnih = 0;
40
41     for (j = 0; j < k; j++) {
42         /* Racuna se broj negativnih u j-oj koloni */
43         broj_negativnih = broj_negativnih_u_koloni(m, v, j);
44         /* Ukoliko broj negativnih u j-oj koloni veci od trenutnog
45            maksimalnog, azurira se trenutni maksimalni broj negativnih
46            kao i indeks kolone sa maksimalnim brojem negativnih */
47         if (maks_broj_negativnih < broj_negativnih) {
48             maks_indeks_kolone = j;
49             maks_broj_negativnih = broj_negativnih;
50         }
51     }
52     return maks_indeks_kolone;
53 }
54
55 int main()
56 {
57     int m[MAKS][MAKS];
58     int v, k;
59
60     /* Ucitava se broj vrsta matrice */
61     scanf("%d", &v);
62
63     /* Proverava se validnost broja vrsta */
64     if (v < 0 || v > MAKS) {
65         fprintf(stderr, "-1\n");
66         exit(EXIT_FAILURE);
67     }
68
69     /* Ucitava se broj kolona matrice */
70     scanf("%d", &k);
71
72     /* Proverava se validnost broja kolona */
73     if (k < 0 || k > MAKS) {
74         fprintf(stderr, "-1\n");
75         exit(EXIT_FAILURE);
76     }
77     /* Ucitaju se elementi matrice */
78     ucitaj_matricu(m, v, k);
79
80     /* Pronalazi se kolona koja sadrzi najveći broj negativnih
81        elemenata i ispisuje se rezultat */
82     printf("%d\n", maks_indeks(m, v, k));
83
84     exit(EXIT_SUCCESS);
85 }
```

Rešenje A.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;
11
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    /* Otvara se datoteka 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    /* Cita se sadrzaj 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, p je NULL i 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
40    /* Zatvara se datoteka */
41    fclose(f);
42
43    /* Ispisuje se vrednost brojaca */
44    printf("%d\n", brojac);
45
46    exit(EXIT_SUCCESS);
47 }
```

## Rešenje A.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
16 /* Funkcija racuna povrsinu trougla koristeći Heronov obrazac */
17 double povrsina(trougao t)
18 {
19     /* Racunaju se duzine stranica trougla */
20     double a = duzina(t.xb, t.yb, t.xc, t.yc);
21     double b = duzina(t.xa, t.ya, t.xc, t.yc);
22     double c = duzina(t.xa, t.ya, t.xb, t.yb);
23     /* Racuna se poluobim trougla */
24     double s = (a + b + c) / 2;
25     /* Primenom Heronovog obrasca racuna se povrsina trougla */
26     return sqrt(s * (s - a) * (s - b) * (s - c));
27 }
28
29 /* Funkcija poredi dva trougla. Ukoliko je povrsina trougla koji je
30 prvi argument funkcije manja od povrsine trougla koji je drugi
31 element funkcije funkcija vraca 1, ukoliko je veca -1, a ukoliko
32 su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
33 napisana tako da se moze proslediti funkciji qsort da se niz
34 trouglova sortira po povrsini opadajuće. */
35 int poredi(const void *a, const void *b)
36 {
37     trougao x = *(trougao *) a;
38     trougao y = *(trougao *) b;
39     double xp = povrsina(x);
40     double yp = povrsina(y);
41     if (xp < yp)
42         return 1;
43     if (xp > yp)
44         return -1;
45     return 0;
46 }
47
48 int main()
49 {
50     FILE *f;
```



```
52     int n, i;
53     trougao *niz;
54
55     /* Otvara se datoteka ciji je naziv trouglovi.txt */
56     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
57         fprintf(stderr, "-1\n");
58         exit(EXIT_FAILURE);
59     }
60
61     /* Ucitava se podatak o broju trouglova iz datoteke */
62     if (fscanf(f, "%d", &n) != 1) {
63         fprintf(stderr, "-1\n");
64         exit(EXIT_FAILURE);
65     }
66
67     /* Dinamicka alokacija memorije za niz trouglova duzine n
68        rezervise se memorijski prostor */
69     if ((niz = malloc(n * sizeof(trougao))) == NULL) {
70         fprintf(stderr, "-1\n");
71         exit(EXIT_FAILURE);
72     }
73
74     /* Ucitavaju se podaci u niz iz otvorene datoteke */
75     for (i = 0; i < n; i++) {
76         if (fscanf(f, "%lf%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
77                 &niz[i].xb, &niz[i].yb, &niz[i].xc,
78                 &niz[i].yc) != 6) {
79             fprintf(stderr, "-1\n");
80             exit(EXIT_FAILURE);
81         }
82     }
83
84     /* Poziva se funkcija qsort da sortira niz na osnovu funkcije
85        poredi */
86     qsort(niz, n, sizeof(trougao), &poredi);
87
88     /* Ispisuje se sortirani niz na standardni izlaz */
89     for (i = 0; i < n; i++)
90         printf("%g %g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
91             niz[i].yb, niz[i].xc, niz[i].yc);
92
93     /* Oslobadja se dinamicki alocirana memorija */
94     free(niz);
95
96     /* Zatvara se datoteka */
97     fclose(f);
98
99     exit(EXIT_SUCCESS);
100 }
```

## Rešenje A.6

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

main.c

```

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

4
/* Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
6  stablo. Funkcija vraca 1 u slucaju neuspesnog dodavanja elementa
   u stablo, a inace 0. */
8 int ucitaj_stablo(Cvor ** koren)
{
10  *koren = NULL;
   int x;
12  /* Sve dok ima brojeva na standardnom ulazu, ucitani brojevi se
   dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
14  je i povratna vrednost iz funkcije ucitaj_stablo 1. */
   while (scanf("%d", &x) == 1)
16     if (dodaj_u_stablo(koren, x) == 1)
       return 1;
18  return 0;
}

20
/* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
22 int prebroj_n(Cvor * koren, int n)
{
24  /* Ukoliko je stablo prazno, rezultat je nula. Takodje, ako je n
   negativan broj, na tom nivou nema cvorova i rezultat je nula. */
26  if (koren == NULL || n < 0)
     return 0;
28  /* Ukoliko je n = 0, na tom nivou je samo koren. Ukoliko ima
   jednog potomka funkcija vraca 1, inace 0 */
30  if (n == 0) {
     if (koren->levo == NULL && koren->desno != NULL)
32     return 1;
     if (koren->levo != NULL && koren->desno == NULL)
34     return 1;
     return 0;
36  }
38  /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
   (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om
   nivou levog podstabla */
40  return prebroj_n(koren->levo, n - 1)
     + prebroj_n(koren->desno, n - 1);
42 }

44 int main()
{

```

## A Ispitni rokovi

```
46  Cvor *koren;
    int n;
48  scanf("%d", &n);

50  /* Ucitavaju se elementi u stablo. U slucaju neuspesne alokacije
    oslobodja se alocirana memorija i izlazi se iz programa. */
52  if (ucitaj_stablo(&koren) == 1) {
    fprintf(stderr, "-1\n");
54  oslobodi_stablo(&koren);
    exit(EXIT_FAILURE);
56  }

58  /* Ispisuje se rezultat */
    printf("%d\n", prebroj_n(koren, n));
60
62  /* Oslobadja se dinamicki alocirana memorija za stablo */
    oslobodi_stablo(&koren);

64  exit(EXIT_SUCCESS);
}
```

### Rešenje A.7

```
#include <stdio.h>

2
/* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
4  zapis argumenta x rotira za n mesta udesno */
unsigned int rotiraj(unsigned int x, unsigned int n)
6  {
    int i;
    unsigned int maska = 1;
8  /* Formira se maska sa n jedinica na kraju, npr za n=4 maska bi
    izgledala: 000...00001111 */
10  /* Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
    nastavku je drugi nacin. */
12  for (i = 1; i < n; i++)
14  maska = (maska << 1) | 1;
    /* Kada se x pomeri za n mesta udesno x >> n, poslednjih n bitova
16  binarne reprezentacije broja x ce "ispasti". Za rotaciju je
    potrebno da se tih n bitova postavi na pocetak broja. Kreirana
18  maska omogucava da se tih n bitova izdvoji sa (maska & x), a
    zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta ulevo
20  tih n bitova postavlja na pocetak primenom bitovske
    disjunkcije i dobija se trazeni broj. */
22  return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
}

24
int main()
26 {
    unsigned int x, n;
28
```

```

30  /* Ucitaju se brojevi sa standardnog ulaza */
    scanf("%u%u", &x, &n);

32  /* Ispisuje se rezultat */
    printf("%u\n", rotiraj(x, n));
34  return 0;
}

```

## Rešenje A.8

*liste.h*

```

1  #ifndef _LISTE_H_
2  #define _LISTE_H_ 1

4  /* Struktura koja predstavlja cvor liste */
    typedef struct cvor {
6      double vrednost;
        struct cvor *sledeci;
8  } Cvor;

10 /* Pomocna funkcija koja kreira cvor. */
    Cvor *napravi_cvor(double broj);

12 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
14    ciji se pocetni cvor nalazi na adresi adresa_glave. */
    void oslobodi_listu(Cvor ** adresa_glave);

16 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
18    ili NULL kao je lista prazna */
    Cvor *pronadji_poslednji(Cvor * glava);

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

24 /* Funkcija prikazuje sve elemente liste pocev od glave ka kraju
26    liste. */
    void ispisi_listu(Cvor * glava);

28 #endif

```

*liste.c*

```

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

```

```
5  /* Pomocna funkcija koja kreira cvor. */
   Cvor *napravi_cvor(double broj)
6  {
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9   if (novi == NULL)
       return NULL;
11  /* inicijalizacija polja u novom cvoru */
   novi->vrednost = broj;
13  novi->sledeci = NULL;
       return novi;
15  }

17  /* Funkcija oslobadja dinamiccku memoriju zauzetu za elemente liste
      ciji se pocetni cvor nalazi na adresi adresa_glave. */
19  void oslobodi_listu(Cvor ** adresa_glave)
   {
21  Cvor *pomocni = NULL;
       while (*adresa_glave != NULL) {
23  pomocni = (*adresa_glave)->sledeci;
           free(*adresa_glave);
25  *adresa_glave = pomocni;
       }
27  }

29  /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
      ili NULL kao je lista prazna */
31  Cvor *pronadji_poslednji(Cvor * glava)
   {
33  /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
      funkcija vraca NULL. */
35  if (glava == NULL)
       return NULL;
37  while (glava->sledeci != NULL)
       glava = glava->sledeci;
39  return glava;
   }

41  /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
      greske pri alokaciji memorije, inace vraca 0. */
43  int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
   {
45  Cvor *novi = napravi_cvor(broj);
47  if (novi == NULL)
       return 1;
49  if (*adresa_glave == NULL) {
       *adresa_glave = novi;
51  return 0;
   }
53  Cvor *poslednji = pronadji_poslednji(*adresa_glave);
   poslednji->sledeci = novi;
55  return 0;
   }
```

```
57 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
58 */
59 void ispisi_listu(Cvor * glava)
60 {
61     for (; glava != NULL; glava = glava->sledeci)
62         printf("%.2lf ", glava->vrednost);
63     putchar('\n');
64 }
65 }
```

*main.c*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "liste.h"
4
5  /* Funkcija umece 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
15    vraca 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 inicijalizuje 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 cvorovima i kreira cvor sa tom vrednoscu.
29        U slucaju neuspele alokacije novog cvora, funkcija vraca 1.
30        Inace, novi cvor se umece izmedju dva cvora za koje je 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 umece novi cvor iza tekuceg cvora */
39         dodaj_iza(tekuci, novi);
```

```
41     /* Tekuci cvor se pomera na narednog u listi (to je
42        novoumetnuti cvor), a zatim jos jednom da bi pokazivao na
43        naredni cvor iz polazne liste */
44     tekuci = tekuci->sledeci;
45     tekuci = tekuci->sledeci;
46 }
47
48     return 0;
49 }
50
51 int main()
52 {
53     Cvor *glava = NULL;
54     double broj;
55
56     /* Dok se ne stigne do kraja ulaza, ucitavaju se elementi i
57        dodaju se na kraj liste */
58     while (scanf("%lf", &broj) > 0) {
59         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
60            memorije za nov cvor. Memoriju alociranu za cvorove liste
61            treba osloboditi. */
62         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
63             fprintf(stderr, "Greska: Neuspela alokacija za cvor %lf.\n",
64                     broj);
65             oslobodi_listu(&glava);
66             exit(EXIT_FAILURE);
67         }
68     }
69
70     /* Poziva se funkcija da dopuni listu. Ako je funkcija vratila 1,
71        onda je bilo greske pri alokaciji memorije za nov cvor.
72        Memoriju alociranu za cvorove liste treba osloboditi. */
73     if (dopuni_listu(&glava) == 1) {
74         fprintf(stderr, "Greska: Neuspela alokacija za cvor %lf.\n",
75                 broj);
76         oslobodi_listu(&glava);
77         exit(EXIT_FAILURE);
78     }
79
80     /* Ispisuju se elementi liste */
81     ispisi_listu(glava);
82
83     /* Oslobadja se memorija rezervisana za listu */
84     oslobodi_listu(&glava);
85
86     exit(EXIT_SUCCESS);
87 }
```

## Rešenje A.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 }
13
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
25 0. Na adresu zbir_magicnog bice upisan zbir elemenata neke vrste
26 ili kolone ukoliko je kvadrat magican. */
27 int magicni_kvadrat(int **M, int n, int *zbir_magicnog)
28 {
29     int i, j;
30     int zbir = 0, zbir_pom;
31     /* Promenljivu zbir inicijalizujemo na zbir 0-te vrste */
32     zbir = zbir_vrste(M, n, 0);
33
34     /* Racunaju se zbrovi 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
42     /* Racunaju se zbrovi u svim kolonama i ako neki razlikuje od
43 vrednosti promeljive zbir funkcija vraca 1 */
44     for (j = 0; j < n; j++) {
45         zbir_pom = zbir_kolone(M, n, j);
46         if (zbir_pom != zbir)
47             return 0;
48     }
49     /* Inace, zbrovi svih vrsta i kolona su jednaki i postavlja se
50 vrednost zbira na adresu zbir_magicnog i funkcija vraca 1. */
51     *zbir_magicnog = zbir;
```



```
    return 1;
52 }

54 int main()
55 {
56     int n;
57     int **matrica = NULL;
58     int zbir_magicnog;
59     scanf("%d", &n);
60
61     /* Provera da li je n strogo pozitivan */
62     if (n <= 0) {
63         printf("-1\n");
64         exit(EXIT_FAILURE);
65     }
66
67     /* Dinamicka alokacija kvadratne matrice dimenzije n */
68     matrica = alociraj_matricu(n, n);
69     if (matrica == NULL) {
70         printf("-1\n");
71         exit(EXIT_FAILURE);
72     }
73
74     /* Ucitavaju se elementi matrice sa standardnog ulaza */
75     ucitaj_matricu(matrica, n, n);
76
77     /* Ispisuje se rezultat na osnovu funkcije magicni_kvadrat */
78     if (magicni_kvadrat(matrica, n, &zbir_magicnog)) {
79         printf("%d\n", zbir_magicnog);
80     } else
81         printf("-\n");
82
83     /* Oslobadja se dinamicki alocirana memorija */
84     matrica = dealociraj_matricu(matrica, n);
85
86     exit(EXIT_SUCCESS);
87 }
```

### Rešenje A.10

```
#include <stdio.h>
2 #include <stdlib.h>
3
4 #define BITOVA_U_BAJTU 8
5
6 /* Funkcija u datom broju x menja mesta prvom i četvrtom bajtu */
7 unsigned int zamena(unsigned int x)
8 {
9     /* Deklaracija promenljivih za odgovarajuće maske i pomocne
10     promenljive */
11     unsigned maska_prvi_bajt, maska_cetvrti_bajt;
```

```

12 unsigned maska_prvi_bajt_komplement,
    maska_cetvrti_bajt_komplement;
14 unsigned prvi_bajt, cetvrti_bajt;
    unsigned i;

16
18 /* Maska_prvi_bajt odgovara broju cija je binarna reprezentacija
    00000...00000111111111 (8 bitova najmanje tezine su jedinice,
    a ostalo su nule). Moze se dobiti i tako sto se
20 maska_prvi_bajt postavi na heksadekadnu vrednost FF. Drugi
    nacin za inicijalizaciju maske maska_prvi_bajt je dodavanjem
22 jedinica sa desne strane: */
    maska_prvi_bajt = 1;
24 for (i = 1; i < BITOVA_U_BAJTU; i++)
    maska_prvi_bajt = maska_prvi_bajt << 1 | 1;

26
28 /* Maska_cetvrti_bajt odgovara broju cija je binarna
    reprezentacija 111111100000...00000 (8 bitova najvece tezine
    su jedinice, a ostalo su nule) i moze se dobiti pomeranjem
30 bitova prethodno kreirane maske maska_prvi_bajt tako da
    jedinice budu na poziciji bajta najvece tezine. */
32 maska_cetvrti_bajt =
    maska_prvi_bajt << ((sizeof(unsigned) - 1) * BITOVA_U_BAJTU);

34
36 /* Primenom operatora ~ na maska_prvi_bajt dobija se broj cija je
    binarna reprezentacija 11111...1111100000000 (8 bitova
    najmanje tezine su nule, a ostalo su jedinice) */
38 maska_prvi_bajt_komplement = ~maska_prvi_bajt;
    /* Primenom operatora ~ na maska_prvi_bajt dobija se broj cija je
40 binarna reprezentacija 0000000011111...11111 (8 bitova najvece
    tezine su nule, a ostalo su jedinice) */
42 maska_cetvrti_bajt_komplement = ~maska_cetvrti_bajt;

44 /* U promenljivu prvi_bajt se smesta broj koji se dobija kada se
    bitovi prvog bajta broja x pomere ulevo, tako da budu na
46 poziciji cetvrtog bajta */
    prvi_bajt =
48 (maska_prvi_bajt & x) << ((sizeof(unsigned) - 1) *
    BITOVA_U_BAJTU);

50 /* U promenljivu cetvrti_bajt se smesta broj koji se dobija kada
    se bitovi cetvrtog bajta broja x pomere udesno, tako da budu
52 na poziciji prvog bajta */
    cetvrti_bajt =
54 (maska_cetvrti_bajt & x) >> ((sizeof(unsigned) - 1) *
    BITOVA_U_BAJTU);

56
58 /* Na nule se postavlja 8 bitova najmanje tezine, a ostali bitovi
    ostaju nepromenjeni */
    x = x & maska_prvi_bajt_komplement;

60
62 /* Na nule se postavlja 8 bitova najvece tezine, a ostali bitovi
    ostaju nepromenjeni */
    x = x & maska_cetvrti_bajt_komplement;

```

```
64  /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi iz
66     prvog bajta */
68  x = x | prvi_bajt;

70  /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi iz
72     prvog bajta */
74  x = x | cetvrsti_bajt;

76  return x;
78  }

80  int main()
82  {
84     int x;

86     /* Sa standardnog ulaza se ucitava ceo broj */
88     scanf("%d", &x);

90     /* Provera da li je uneti broj negativan */
92     if (x < 0) {
94         fprintf(stderr, "-1\n");
96         exit(EXIT_FAILURE);
98     }

100    /* Ispisuje se rezultat primene funkcije zamena na uneti broj x */
102    printf("%u\n", zamena(x));

104    exit(EXIT_SUCCESS);
106  }
```

### Rešenje A.11

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"
4
5  /* Funkcija racuna duzinu najduzeg puta od korena do nekog lista */
6  int najduzi_put(Cvor * koren)
7  {
8     /* Pomocne promenljive */
9     int najduzi_u_levom, najduzi_u_desnom;
10
11    /* Ako je stablo prazno, povratna vrednost je -1 */
12    if (koren == NULL)
13        return -1;
14
15    /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
```

```

16 najduzi_u_levom = najduzi_put(koren->levo);
18 /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
   najduzi_u_desnom = najduzi_put(koren->desno);
20
22 /* Veca od prethodno izracunatih vrednosti za podstabla se
   uvecava za 1 i vraca kao konacan rezultat */
   return 1 + (najduzi_u_levom >
24             najduzi_u_desnom ? najduzi_u_levom :
26             najduzi_u_desnom);
}

28 int main()
   {
30   Cvor *stablo = NULL;
   int x;
32
34   /* U svakoj iteraciji se procitani broj dodaje u stablo. */
   while (scanf("%d", &x) == 1)
36     if (dodaj_u_stablo(&stablo, x) == 1) {
       fprintf(stderr, "-1\n");
       exit(EXIT_FAILURE);
38     }
40
42   /* Ispisuje se rezultat rada trazene funkcije */
   printf("%d\n", najduzi_put(stablo));
44
46   /* Oslobadja se memorija */
   oslobodi_stablo(&stablo);

   exit(EXIT_SUCCESS);
}

```

### Rešenje A.12

```

1 #include <stdio.h>
   #include <stdlib.h>
3 /* Ime datoteke nije duze od 30 karaktera */
   #define MAX 31
5
7 /* Funkcija alocira memorijski prostor za matricu sa n vrsta i m
   kolona */
   float **alociraj_matricu(int n, int m)
9 {
   float **matrica = NULL;
11   int i, j;
13
15   /* Alocira se prostor za niz vrsta matrice */
   matrica = (float **) malloc(n * sizeof(float *));
   /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije
   ce biti NULL, sto mora biti provereno u main funkciji */

```

```
17  if (matrica == NULL)
18      return NULL;
19
20  /* Alocira se prostor za svaku vrstu matrice */
21  for (i = 0; i < n; i++) {
22      matrica[i] = (float *) malloc(m * sizeof(float));
23      /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
24         prethodno alocirani resursi, i povratna vrednost je NULL */
25      if (matrica[i] == NULL) {
26          for (j = 0; j < i; j++)
27              free(matrica[j]);
28          free(matrica);
29          return NULL;
30      }
31  }
32
33  return matrica;
34 }
35
36 /* Funkcija oslobadja alocirani memorijski prostor */
37 float **deallociraj_matricu(float **matrica, int n)
38 {
39     int i;
40     /* Oslobadja se prostor rezervisan za svaku vrstu */
41     for (i = 0; i < n; i++)
42         free(matrica[i]);
43
44     /* Oslobadja se memorija za niz pokazivaca na vrste */
45     free(matrica);
46
47     /* Matrica postaje prazna, tj. nealocirana */
48     return NULL;
49 }
50
51 /* Funkcija prebrojava koliko se puta pojavljuje broj x u i-toj
52    vrsti matrice A, gde je m broj elemenata u vrsti */
53 int prebroj_u_itoj_vrsti(float **A, int i, int m, int x)
54 {
55     int j;
56     int broj = 0;
57     for (j = 0; j < m; j++) {
58         if (A[i][j] == x)
59             broj++;
60     }
61     return broj;
62 }
63
64 /* Funkcija vraca indeks vrste matrice A u kojoj se realan broj x
65    pojavljuje najmanje puta */
66 int indeks_vrste(float x, float **A, int n, int m)
67 {
68     /* Indeks vrste sa minimalnim brojem pojavljivanja broja x */
```

```
69  int min;
    /* Broj pojavljivanja broja x u vrsti sa indeksom min */
71  int min_broj;
    /* Promenljiva u kojoj ce se racunati broj pojavljivanja broja x
73  u tekucnoj vrsti */
    int broj_u_vrsti;
75  /* Pomocne promenljive */
    int i;
77
    /* Promenljiva min se inicijalizuje na nulu, a min_broj na broj
79  pojavljivanja broja x u nultoj vrsti */
    min = 0;
81  min_broj = prebroj_u_itoj_vrsti(A, 0, m, x);
83
    /* Za svaku vrstu (osim nulte) se racuna broj pojavljivanja broja
    x u njoj, pa ukoliko je taj broj manji od trenutno najmanjeg
85  azuriraju se promenljive min i min_broj */
    for (i = 1; i < n; i++) {
87        broj_u_vrsti = prebroj_u_itoj_vrsti(A, i, m, x);
        if (broj_u_vrsti < min_broj) {
89            min_broj = broj_u_vrsti;
            min = i;
91        }
    }
93
    /* Funkcija vraca odgovarajuci indeks vrste */
95  return min;
}
97
int main()
99 {
    FILE *in;
101  char datoteka[MAX];
    float broj;
103  float **A = NULL;
    int i, j, m, n;
105
    /* Sa standardnog ulaza se ucitava ime datoteke i realan broj */
107  scanf("%s", datoteka);
    scanf("%f", &broj);
109
    /* Otvara se datoteka za citanje */
111  in = fopen(datoteka, "r");
113
    /* Provera da li je datoteka uspesno otvorena */
    if (in == NULL) {
115        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
117    }
119
    /* Dimenzije matrice se ucitavaju iz datoteke (prva dva cela
    broja u datoteci). U slucaju neuspesnog ucitavanja, na
```

```
121     standardni izlaz za greske se ispisuje -1 i prekida se
      program. */
123     if (fscanf(in, "%d %d", &n, &m) == EOF) {
      fprintf(stderr, "-1\n");
125     exit(EXIT_FAILURE);
      }
127
      /* Provera da li su ucitani brojevi m i n pozitivni */
129     if (n <= 0 || m <= 0) {
      fprintf(stderr, "-1\n");
131     exit(EXIT_FAILURE);
      }
133
      /* Alokacija matrice */
135     A = alociraj_matricu(n, m);
137
      /* Provera da li je alokacija uspela */
139     if (A == NULL) {
      fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
141     }
143
      /* Ucitavaju se elementi matrice iz datoteke */
145     for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++)
147         fscanf(in, "%f", &A[i][j]);
      }
149
      /* Zatvara se datoteka */
      fclose(in);
151
      /* Ispisuje se rezultat poziva funkcije */
153     printf("%d\n", indeks_vrste(broj, A, n, m));
155
      /* Oslobadja se memorija koju je zauzimala matrica */
      A = dealociraj_matricu(A, n);
157
      exit(EXIT_SUCCESS);
159 }
```

Elektronsko izdanje 2015

