

## PROGRAMIRANJE 2



Univerzitet u Beogradu  
Matematički fakultet

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

# **PROGRAMIRANJE 2**

## **Zbirka zadataka sa rešenjima**

**Beograd  
2016.**

Autori:

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

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

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Anđelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu

Studentski trg 16, 11000 Beograd

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

Recenzenti:

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

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

Obrada teksta, crteži i korice: *autori*

Štampa: Skripta internacional, Beograd

Tiraž: 150

СІР Каталогизација у публикацији

Народна библиотека Србије, Београд

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

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



# 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 . . . . .	77
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 . . . . .	133
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 rešavanju problema, ispravnom radu sa pokazivačima i dinamički alociranom memorijom, osnovnim algoritmima pretraživanja i sortiranja, kao i radu sa dinamičkim strukturama podataka, poput listi i stabala. Zadaci koji se nalaze u ovoj zbirci predstavljaju objedinjen skup zadataka sa vežbi i praktikuma ovog kursa, kao i primere zadataka sa održanih ispita. Elektronska verzija zbirke i radni repozitorijum elektronskih verzija rešenja zadataka, dostupni su u okviru strane kursa [www.programiranje2.matf.bg.ac.rs](http://www.programiranje2.matf.bg.ac.rs), besplatno.

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.

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 u uobličavanju ovog materijala.

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

*Autori*

---

<sup>1</sup>Adrese autora su: milena, jgraovac, aspasic, mirko, andjelkaz, nina, 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) Definirati 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.

- (a) Definirati 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.
- (b) Napisati funkciju `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz, od najvišeg ka najnižem stepenu. Ispisati samo koeficijente koji su različiti od nule.
- (c) Napisati funkciju `Polinom ucitaj()` koja učitava polinom sa standardnog ulaza. Za polinom najpre uneti stepen, a zatim njegove koeficijente.
- (d) Napisati funkciju `double izracunaj(const Polinom * p, double x)` koja vraća vrednosti polinoma `p` u datoj tački `x` koristeći Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja vraća zbir dva polinoma `p` i `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja vraća proizvod dva polinoma `p` i `q`.
- (g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja vraća izvod polinoma `p`.
- (h) 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 najvećeg stepena do nultog):
3 1.2 3.5 2.1 4.2
Unesite polinom q (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
2 2.1 0 -3.9
Zbir polinoma je polinom z:
1.20x^3+5.60x^2+2.10x+0.30
Proizvod polinoma je polinom r:
2.52x^5+7.35x^4-0.27x^3-4.83x^2-8.19x-16.38
Unesite tacku u kojoj racunate vrednost polinoma 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) Definirati 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 vraća brojilac razlomka `r`.
- (e) Napisati funkciju `int imenilac(const Razlomak * r)` koja vraća 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: 0xf4f4f4f
Binarna reprezentacija: 01001111010011110100111101001111
Unesite broj tipa short: 0xf4f
Binarna reprezentacija: 0100111101001111
Unesite broj tipa char: 0xf
Binarna reprezentacija: 01001111
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkcije `prebroj_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 heksadekasnom formatu i redni broj funkcije koju treba primeniti ('1' ili '2'), a zatim na standardni izlaz ispisati broj jedinica u binarnom zapisu učitanoog broja pozivom izabrane funkcije. Ukoliko korisnik ne unese ispravnu vrednost za

## 1 Uvodni zadaci

---

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
```

### 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 GREŠKU:
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.

- (a) 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.
- (b) 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.
- (c) Napisati funkciju `unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)` koja vraća broj u kome se  $n$  bitova najmanje težine poklapa sa  $n$  bitova broja  $x$  počevši od pozicije  $p$ , dok su mu ostali bitovi postavljeni na 0.
- (d) Napisati funkciju `unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p, unsigned y)` koja vraća broj koji se dobija upisivanjem poslednjih  $n$  bitova najmanje težine broja  $y$  u broj  $x$ , počevši od pozicije  $p$ .
- (e) Napisati funkciju `unsigned invertuj(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija invertovanjem  $n$  bitova broja  $x$  počevši od pozicije  $p$ .

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve  $x$ ,  $n$ ,  $p$ ,  $y$  koji se unose sa standardnog ulaza. Na standardni izlaz ispisati binarne reprezenatacije brojeva  $x$  i  $y$ , a zatim i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija. NAPOMENA: *Bit najmanje težine je krajnji desni bit i njegova pozicija se označava nultom dok se pozicije ostalih bitova uvećavaju za jedan, sa desna na levo.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite neoznaceni ceo broj x: 235
Unesite neoznaceni ceo broj n: 9
Unesite neoznaceni ceo broj p: 24
Unesite neoznaceni ceo broj y: 127
x = 235                                     = 00000000000000000000000011101011
postavi_0( 235,      9,      24)          = 0000000000000000000000000011101011

x = 235                                     = 00000000000000000000000011101011
postavi_1( 235,      9,      24)          = 00000001111111110000000011101011

x = 235                                     = 00000000000000000000000011101011
vrati_bitove( 235,      9,      24)       = 000000000000000000000000000000000000

x = 235                                     = 00000000000000000000000011101011
y = 127                                    = 0000000000000000000000000011111111
postavi_1_n_bitove( 235,  9,  24,  127)   = 000000000111111110000000011101011

x = 235                                     = 00000000000000000000000011101011
invertuj( 235,      9,      24)           = 00000001111111110000000011101011
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite neoznaceni broj x: 2882398951
Unesite neoznaceni broj n: 5
Unesite neoznaceni broj p: 10
Unesite neoznaceni 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                  = 00000010000110000111001000101110
postavi_1_n_bitove(2882398951, 5, 10, 35156526) = 10101011110011011110101110100111

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

[Rešenje 1.8]

**Zadatak 1.9** Pod rotiranjem 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.

- (a) Napisati funkciju `unsigned rotiraj_ulevo(unsigned x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta ulevo datog celog neoznačenog broja `x`.
- (b) Napisati funkciju `unsigned rotiraj_udesno(unsigned x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta udesno datog celog neoznačenog broja `x`.
- (c) 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, tatim 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 broj x: ba11a7
Unesite neoznaceni 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:
000000000000000000000001001010101
10101010010000000000000000000000

```

### Test 2

```

ULAZ:
-15
IZLAZ:
11111111111111111111111111101011
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>
ULAZ: 123 10 IZLAZ: 4294967285	ULAZ: 3251 0 IZLAZ: 4294967295	ULAZ: 12541 1024 IZLAZ: 4294966271

**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>
ULAZ: 123 IZLAZ: 0	ULAZ: 3245 IZLAZ: 2	ULAZ: 100328 IZLAZ: 1

## 1.3 Rekurzija

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

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

Napisati program koji testira napisane 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.

### Primer 1

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

### Primer 2

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

[Rešenje 1.17]

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

- (a) funkciju `unsigned paran(unsigned n)` koja proverava da li je broj cifara broja `x` paran i vraća 1 ako jeste, a 0 inače;
- (b) 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:

$$F(0) = 0$$

$$F(1) = 1$$

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

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 clan niza se racuna: 5
|| F(5) = 5
```

*Primer 2*

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

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

$$F(0) = 0$$

$$F(1) = 1$$

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

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

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

Napisati program koji testira napisane 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 koju zelite:
1 - iterativna
2 - rekurzivna
3 - rekurzivna napredna
1
Unesite koeficijente: 2 3
Unesite koji clan niza se racuna: 5
F(5) = 61
```

### Primer 2

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

[Rešenje 1.21]

**Zadatak 1.22** Napisati rekurzivnu funkciju koja sabira dekadne cifre datog celog broja  $x$ . Napisati program koji testira ovu funkciju za broj koji se unosi sa standardnog ulaza.

### Test 1

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

### Test 2

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

### Test 3

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

### Test 4

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

### Test 5

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

[Rešenje 1.22]

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

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

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

### Primer 1

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

### Primer 2

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

[Rešenje 1.23]

**Zadatak 1.24** Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju za niz koji se unosi sa standardnog ulaza. 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  $x$  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 članovi niza.
```

### Primer 2

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

### Test 1

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

### Test 2

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

### Test 3

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



[Rešenje 1.28]

[illegible]

Test 1	Test 2	Test 3
ULAZ:	ULAZ:	ULAZ:
5	125	8
IZLAZ:	IZLAZ:	IZLAZ:
5	7	1

[Rešenje 1.30]

Test 1	Test 2	Test 3
ULAZ:	ULAZ:	ULAZ:
5	16	18
IZLAZ:	IZLAZ:	IZLAZ:
5	1	2

[Rešenje 1.31]

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: a IZLAZ: da </pre>	<pre> ULAZ: aa IZLAZ: da </pre>	<pre> ULAZ: aba IZLAZ: da </pre>
<i>Test 4</i>	<i>Test 5</i>	
<pre> ULAZ: programiranje IZLAZ: ne </pre>	<pre> ULAZ: anavolimilovana IZLAZ: da </pre>	

[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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 2 IZLAZ: 1 2 2 1 </pre>	<pre> ULAZ: 3 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1 </pre>	<pre> ULAZ: -5 Duzina permutacije mora biti broj iz intervala [0, 15]! </pre>

[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.

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

- (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 argument
14     funkcije */
15 void ucitaj_kompleksan_broj(KompleksanBroj * z)
16 {
17     /* Ucitavanje vrednosti 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 je
34 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 */

40 if (z.real != 0) {
    printf("%.2f", z.real);
42
    if (z.imag > 0)
44         printf(" + %.2f i", z.imag);
    else if (z.imag < 0)
46         printf(" - %.2f i", -z.imag);
    } else {
48         if (z.imag == 0)
            printf("0");
50         else
            printf("%.2f i", z.imag);
52     }

54 /* Završava se sa ispisom */
printf(")");
56
}

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

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

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

76 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
78 odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z)
80 {
```

## 1 Uvodni zadaci

---

```
82      /* Konjugovano kompleksan broj z se dobija tako sto se promeni znak
      imaginarnom delu kompleksnog broja */

84      KompleksanBroj z1 = z;

86      z1.imag *= -1;

88      return z1;
    }

90      /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
92      argumenata funkcije */
    KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
94    {
96      /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
      broj ciji je realan deo zbir realnih delova kompleksnih brojeva
      z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
98      brojeva z1 i z2 */

100     KompleksanBroj z = z1;

102     z.real += z2.real;
104     z.imag += z2.imag;

106     return z;
    }

108     /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
110     argumenata funkcije */
    KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
112    {
114     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
      broj ciji je realan deo razlika realnih delova kompleksnih
      brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
116     kompleksnih brojeva z1 i z2 */

118     KompleksanBroj z = z1;

120     z.real -= z2.real;
122     z.imag -= z2.imag;

124     return z;
    }

126     /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
      argumenata funkcije */
    KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
128    {
130     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
      broj ciji se realan i imaginaran deo racunaju po formuli za
      mnozenje kompleksnih brojeva z1 i z2 */

132
```

```

KompleksanBroj z;
134
    z.real = z1.real * z2.real - z1.imag * z2.imag;
136    z.imag = z1.real * z2.imag + z1.imag * z2.real;

138    return z;
}
140
/* Funkcija vraca argument zadatog kompleksnog broja */
142 float argument(KompleksanBroj z)
{
144     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
        iz biblioteke math.h */

146     return atan2(z.imag, z.real);
148 }

150 int main()
{
152     char c;

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

156
    /* Ucitavanje prvog kompleksnog broja, a potom i njegovo
        ispisivanje na standardni izlaz */
158    ucitaj_kompleksan_broj(&z1);
160    ispisi_kompleksan_broj(z1);
    printf("\n");

162
    /* Ucitavanje drugog kompleksnog broja, a potom njegovo ispisivanje
        na standardni izlaz */
164    ucitaj_kompleksan_broj(&z2);
166    ispisi_kompleksan_broj(z2);
    printf("\n");

168
    /* Ucitavanje i provera znaka na osnovu koga korisnik bira
        aritmeticku operaciju koja ce se izvrstiti nad kompleksnim
        brojevima */
170    getchar();
    printf("Unesite znak (+,-,*): ");
174    scanf("%c", &c);
    if (c != '+' && c != '-' && c != '*') {
176        printf("Greska: nedozvoljena vrednost operatora!\n");
        exit(EXIT_FAILURE);
178    }

180
    /* Analizira se uneti operator */
    if (c == '+') {
182        /* Racuna se zbir */
        z = saberi(z1, z2);
184    } else if (c == '-') {

```

## 1 Uvodni zadaci

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

### Rešenje 1.2

*kompleksan\_broj.h*

```
1  /* Zaglavlje kompleksan_broj.h sadrzi definiciju tipa KompleksanBroj
2     i
3     deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavlje
4     nikada ne treba da sadrzi definicije funkcija. Da bi neki program
5     mogao da koristi ove brojeve i funkcije iz ove biblioteke,
6     neophodno je da ukljuci ovo 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 u
11    narednoj pretprocesorskoj define direktivi. */
12 #ifndef _KOMPLEKSAN_BROJ_H
13 #define _KOMPLEKSAN_BROJ_H
14
15 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
```



```
15     koje se koriste u definicijama funkcija navedenim u
        kompleksan_broj.c */
17 #include <stdio.h>
#include <math.h>

19 /* Struktura KompleksanBroj */
typedef struct {
21     float real;
    float imag;
23 } KompleksanBroj;

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

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

33 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
    obliku (x + i y) */
35 void ispisi_kompleksan_broj(KompleksanBroj z);

37 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
float realan_deo(KompleksanBroj z);
39
/* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
41 float imaginaran_deo(KompleksanBroj z);

43 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
float moduo(KompleksanBroj z);
45
/* Funkcija vraca vrednost konjugovano kompleksnog broja koji
47 odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z);
49
/* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
51 argumenata funkcije */
KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
53
/* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
55 argumenata funkcije */
KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
57
/* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
59 argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
61
/* Funkcija vraca argument zadatog kompleksnog broja */
63 float argument(KompleksanBroj z);

65 /* Kraj zakljucanog dela */
```

## 1 Uvodni zadaci

---

```
#endif
```

*kompleksan\_broj.c*

```
/* Ukljucuje se zaglavlje za rad sa kompleksnim brojevima, jer je
2   neophodno da bude poznata definicija tipa KompleksanBroj. Takodje,
   time su ukljucena zaglavlja standardne biblioteke koja su navedena
4   u kompleksan_broj.h */
#include "kompleksan_broj.h"

6

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

16 void ispisi_kompleksan_broj(KompleksanBroj z)
{
18   /* Zapocinje se sa ispisom */
   printf("(");

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

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

32       if (z.imag > 0)
           printf(" + %.2f i", z.imag);
       else if (z.imag < 0)
           printf(" - %.2f i", -z.imag);
36   } else {
       if (z.imag == 0)
           printf("0");
       else
           printf("%.2f i", z.imag);
40   }

42 }

44 /* Zavrшава se sa ispisom */
   printf(")");
46 }
```

```
48 float realan_deo(KompleksanBroj z)
49 {
50     /* Vraca se vrednost realnog dela kompleksnog broja */
51     return z.real;
52 }
53
54 float imaginaran_deo(KompleksanBroj z)
55 {
56     /* Vraca se vrednost imaginarnog dela kompleksnog broja */
57     return z.imag;
58 }
59
60 float moduo(KompleksanBroj z)
61 {
62     /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
63     return sqrt(z.real * z.real + z.imag * z.imag);
64 }
65
66 KompleksanBroj konjugovan(KompleksanBroj z)
67 {
68     /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
69        se promeni znak imaginarnom delu kompleksnog broja */
70     KompleksanBroj z1 = z;
71     z1.imag *= -1;
72     return z1;
73 }
74
75 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
76 {
77     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
78        broj ciji je realan deo zbir realnih delova kompleksnih brojeva
79        z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
80        brojeva z1 i z2 */
81     KompleksanBroj z = z1;
82
83     z.real += z2.real;
84     z.imag += z2.imag;
85
86     return z;
87 }
88
89 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
90 {
91     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
92        broj ciji je realan deo razlika realnih delova kompleksnih
93        brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
94        kompleksnih brojeva z1 i z2 */
95     KompleksanBroj z = z1;
96     z.real -= z2.real;
97     z.imag -= z2.imag;
98     return z;
99 }
```

## 1 Uvodni zadaci

---

```
100 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
102 {
104     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
        broj ciji se realan i imaginaran deo racunaju po formuli za
        mnozenje kompleksnih brojeva z1 i z2 */
106     KompleksanBroj z;

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

110     return z;
112 }

114 float argument(KompleksanBroj z)
116 {
118     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
        iz biblioteke math.h */
        return atan2(z.imag, z.real);
120 }
```

*main.c*

```
1  /*****
2  Ovaj program koristi korektno definisanu biblioteku kompleksnih
3  brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
        kompleksnog broja
        i popis deklaracija podrzanih funkcija, a u kompleksan_broj.c se
        nalaze
5  njihove definicije.

7  Kompilacija programa se najjednostavnije postize naredbom
        gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c
9
        Kompilacija se moze uraditi i na sledeci nacin:
11 gcc -Wall -c -o kompleksan_broj.o kompleksan_broj.c
        gcc -Wall -c -o main.o main.c
13 gcc -lm -o kompleksan_broj kompleksan_broj.o main.o

15 Napomena: Prethodne komande se koriste kada se sva tri navedena
        dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
17 kompleksan_broj.c kompleksan_broj.h) nalazi u direktorijumu sa imenom
        header_dir
        prevodjenje se vrsi dodavanjem opcije opcije -I header_dir
19 gcc -I header_dir -Wall -lm -o kompleksan_broj kompleksan_broj.c main
        .c
        *****/
21

23 #include <stdio.h>
```

```

/* Uključuje se zaglavlje neophodno za rad sa kompleksnim brojevima
*/
25 #include "kompleksan_broj.h"

27 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
   polarni oblik */
29 int main()
{
31     KompleksanBroj z;

33     /* Učitavamo kompleksan broj */
    učitaj_kompleksan_broj(&z);

35     /* Ispisujemo njegov polarni oblik */
37     printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f\n",
           moduo(z), argument(z));

39     return 0;
41 }

```

### Rešenje 1.3

*polinom.h*

```

1  #ifndef _POLINOM_H
   #define _POLINOM_H
3
   #include <stdio.h>
5   #include <stdlib.h>

7   /* Maksimalni stepen polinoma */
   #define MAKS_STEPEN 20
9

11  /* Polinomi se predstavljaju strukturom koja cuva koeficijente
    (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
13  typedef struct {
       double koef[MAKS_STEPEN + 1];
15     int stepen;
   } Polinom;
17

   /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
19    obliku. Polinom se prenosi po adresi da bi se uštedela memorija:
    ne kopira se cela struktura, vec se samo prenosi adresa na kojoj
21    se nalazi polinom koji ispisujemo */
   void ispisi(const Polinom * p);
23

   /* Funkcija koja učitava polinom sa tastature */
25   Polinom učitaj();

```

## 1 Uvodni zadaci

---

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

31 /* Funkcija koja sabira dva polinoma */
    Polinom saberi(const Polinom * p, const Polinom * q);
33
    /* Funkcija koja mnozi dva polinoma p i q */
35 Polinom pomnozi(const Polinom * p, const Polinom * q);

37 /* Funkcija koja racuna izvod polinoma p */
    Polinom izvod(const Polinom * p);
39
    /* Funkcija koja racuna n-ti izvod polinoma p */
41 Polinom n_izvod(const Polinom * p, int n);
    #endif
```

*polinom.c*

```
#include <stdio.h>
2 #include <stdlib.h>
    #include "polinom.h"
4
    void ispisi(const Polinom * p)
6 {
    int nulaPolinom = 1;
    int i;
8    /* Ispisivanje polinoma pocinje od najviseg stepena ka najnizem da
    bi polinom bio ispisan na prirodan nacin. Ispisuju se samo oni
10 koeficijenti koji su razliciti od nule. Ispred pozitivnih
    koeficijenata je potrebno ispisati znak + (osim u slucaju
12 koeficijenta uz najvisi stepen). */
14 for (i = p->stepen; i >= 0; i--) {

16     if (p->koef[i]) {
        /* Polinom nije nula polinom, cim je neki od koeficijenata
18        razlicit od nule */
        nulaPolinom = 0;
20        if (p->koef[i] >= 0 && i != p->stepen)
            putchar('+');
22        if (i > 1)
            printf("%.2fx~%d", p->koef[i], i);
24        else if (i == 1)
            printf("%.2fx", p->koef[i]);
26        else
            printf("%.2f", p->koef[i]);
28    }
}

30 /* U slucaju nula polinoma indikator ce imati vrednost 1 i tada se
    ispisuje nula. */
32 if(nulaPolinom)
```

```

    printf("0");
34   putchar('\n');
}
36
Polinom ucitaj()
38 {
    int i;
40   Polinom p;

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

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

52   /* Unose se koeficijenti polinoma */
       for (i = p.stepen; i >= 0; i--)
54       scanf("%lf", &p.koef[i]);

56   /* Vraca se procitani polinom */
       return p;
58 }

60 double izracunaj(const Polinom * p, double x)
{
62   /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
    svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
64   vrednost odgovarajuceg koeficijenta */

66   /* Primer: Hornerov algoritam za polinom  $x^4+2x^3+3x^2+2x+1$ :
     $x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 2)*x + 1$  */

68   double rezultat = 0;
70   int i = p->stepen;
       for (; i >= 0; i--)
72       rezultat = rezultat * x + p->koef[i];
       return rezultat;
74 }

76 Polinom saberi(const Polinom * p, const Polinom * q)
{
78   Polinom rez;
       int i;

80   /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
    stepenom */
82   rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
84

```

```
86  /* Racunaju se svi koeficijenti rezultujuceg polinoma tako sto se
88  sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
      sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
      od stepena nekog od polaznih polinoma podrazumeva se da je
      koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
      polinoma */
90  for (i = 0; i <= rez.stepen; i++)
92      rez.koef[i] =
          (i > p->stepen ? 0 : p->koef[i]) +
          (i > q->stepen ? 0 : q->koef[i]);

96  /* Vraca se dobijeni polinom */
98  return rez;
100 }

102 Polinom pomnozi(const Polinom * p, const Polinom * q)
104 {
    int i, j;
    Polinom r;

106  /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
      */
    r.stepen = p->stepen + q->stepen;
108  if (r.stepen > MAKS_STEPEN) {
      fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
      exit(EXIT_FAILURE);
110  }

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

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

122  /* Vraca se dobijeni polinom */
124  return r;
126 }

128 Polinom izvod(const Polinom * p)
130 {
    int i;
    Polinom r;

132  /* Izvod polinoma ce imati stepen za jedan stepen manji od stepena
      polaznog polinoma. Ukoliko je stepen polinoma p vec nula, onda
      je rezultujući polinom nula (izvod od konstante je nula). */
134
```



```

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

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

144     /* Vraca se dobijeni polinom */
146     return r;
148 }

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

154     /* Provera da li je n nenegativna vrednost */
156     if (n < 0) {
158         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
160         exit(EXIT_FAILURE);
162     }

160     /* Multi izvod je bas taj polinom */
162     if (n == 0)
164         return *p;

164     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove funkcija
165        za racunanje prvog izvoda polinoma */
166     r = izvod(p);
168     for (i = 1; i < n; i++)
170         r = izvod(&r);

170     /* Vraca se dobijeni polinom */
172     return r;
174 }

```

*main.c*

```

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

4 int main(int argc, char **argv)
5 {
6     Polinom p, q, z, r;
7     double x;
8     int n;

10     /* Unos polinoma p */

```

```
12 printf
    ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
    najveceg stepena do nultog):\n");
p = ucitaj();

14
/* Ispis polinoma p */
16 ispisi(&p);

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

22
/* Polinomi se sabiraju i ispisiuje se izracunati zbir */
24 z = saberi(&p, &q);
printf("Zbir polinoma je polinom z:\n");
26 ispisi(&z);

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

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

38
/* Racuna se n-ti izvoda polinoma i ispisiuje se dobijeni polinoma
    */
40 printf("Unesite izvod polinoma koji zelite:\n");
scanf("%d", &n);
42 r = n_izvod(&r, n);
printf("%d. izvod polinoma r je: ", n);
44 ispisi(&r);

46 exit(EXIT_SUCCESS);
}
```

### Rešenje 1.5

*stampanje\_bitova.h*

```
1 #ifndef _STAMPANJE_BITOVA_H
#define _STAMPANJE_BITOVA_H
3
#include <stdio.h>
```

```

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

11 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
13 celog broja tipa 'short' u memoriji. */
   void stampaj_bitove_short(short x);

15 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
17 karaktera u memoriji. */
   void stampaj_bitove_char(char x);

19 #endif

```

*stampanje\_bitova.c*

```

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

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

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

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

24  putchar('\n');
26 }

28 void stampaj_bitove_short(short x)
{
30  /* Broj bitova celog broja tipa short */
   unsigned velicina = sizeof(short) * 8;

32

```

## 1 Uvodni zadaci

---

```
34  /* Maska koja se koristi za "ocitavanje" bitova broja tipa short */
    unsigned short maska;

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

38
    putchar('\n');
40 }

42 void stampaj_bitove_char(char x)
{
44     /* Broj bitova karaktera */
    unsigned velicina = sizeof(char) * 8;

46
    /* Maska koja se koristi za "ocitavanje" bitova jednog karaktera */
    unsigned char maska;

48
    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
        putchar(x & maska ? '1' : '0');

50
    putchar('\n');
52
54 }
```

*main.c*

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

4 int main()
{
6     int broj_int;
    short broj_short;
8     char broj_char;

10     printf("Unesite broj tipa int: ");
    /* Ucitava se broj sa ulaza */
12     scanf("%x", &broj_int);

14     /* I ispisuje se njegova binarna reprezentacija */
    printf("Binarna reprezentacija: ");
    stampaj_bitove(broj_int);

16

18     printf("Unesite broj tipa short: ");
    /* Ucitava se broj sa ulaza */
20     scanf("%hx", &broj_short);

22     /* I ispisuje se njegova binarna reprezentacija */
    printf("Binarna reprezentacija: ");
    stampaj_bitove_short(broj_short);

24

26     printf("Unesite broj tipa char: ");
```

```

/* Ucitava se broj sa ulaza */
28 scanf("%hhx", &broj_char);

/* I ispisuje se njegova binarna reprezentacija */
30 printf("Binarna reprezentacija: ");
32 stampaj_bitove_char(broj_char);

34 return 0;
}

```

### Rešenje 1.6

```

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

3
/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   kreiranjem odgovarajuće maske i njenim pomeranjem */
5 int prebroj_bitove_1(int x)
7 {
   int br = 0;
   unsigned broj_pomeranja = sizeof(unsigned) * 8 - 1;

11  /* Formiranje se maska čija binarna reprezentacija izgleda
    100000...0000000, koja služi za očitavanje bita najveće težine.
    U svakoj iteraciji maska se pomera u desno za 1 mesto, i
13  očitavamo sledeći bit. Petlja se završava kada više nema
    jedinica tj. kada maska postane nula. */
15  unsigned maska = 1 << broj_pomeranja;
   for (; maska != 0; maska >>= 1)
       x & maska ? br++ : 1;

19  return br;
21 }

23 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   formiranjem odgovarajuće maske i pomeranjem promenljive x */
25 int prebroj_bitove_2(int x)
   {
27     int br = 0;
     unsigned broj_pomeranja = sizeof(int) * 8 - 1;

29     /* Kako je argument funkcije označen ceo broj x naredba x>>=1 bi
    vrsila aritmetičko pomeranje u desno, tj. popunjavanje bita
    najveće težine bitom znaka. U tom slučaju nikad ne bi bio
31  ispunjen uslov x!=0 i program bi bio zarobljen u beskončnoj
    petlji. Zbog toga se koristi pomeranje broja x ulevo i maska
33  koja očitava bit najveće težine. */

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

37
39

```

```
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 izracunat
53        broj jedinica u zapisu broja */
54     printf("Unesite redni broj funkcije:\n");
55     scanf("%d", &i);
56
57     /* Ispisuje se rezultat */
58     if (i == 1){
59         printf("Poziva se funkcija prebroj_bitove_1\n");
60         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_1(x));
61     }else if (i == 2){
62         printf("Poziva se funkcija prebroj_bitove_2\n");
63         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_2(x));
64     }else {
65         fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
66         exit(EXIT_FAILURE);
67     }
68
69     exit(EXIT_SUCCESS);
70 }
```

**Rešenje 1.7**      NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova*

*iz zadatka 1.5.*

```
1  #include <stdio.h>
2  #include "stampanje_bitova.h"
3
4  /* Funkcija vraca najveći neoznačeni broj sastavljen od istih bitova
5     koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
6  unsigned najveći(unsigned x)
7  {
8      unsigned velicina = sizeof(unsigned) * 8;
9
10     /* Formira se maska 100000...0000000 */
11     unsigned maska = 1 << (velicina - 1);
12
13     /* Rezultat se inicijalizuje vrednošću 0 */
14     unsigned rezultat = 0;
15 }
```

```
17  /* Promenljiva x se pomera u levo sve dok postoje jedinice u njenoj
    binarnoj reprezentaciji (tj. sve dok je promenljiva x razlicita
    od nule). */
19  for (; x != 0; x <= 1) {
    /* Za svaku jedinicu koja se koriscenjem maske detektuje na
    21  poziciji najvece tezine u binarnoj reprezentaciji promenjive
    x, potiskuje se jedna nova jedinicu sa leva u rezultat */
    23  if (x & maska) {
        rezultat >>= 1;
    25  rezultat |= maska;
    }
    27  }

    29  /* Vraca se dobijena vrednost */
    return rezultat;
    31  }

    33  /* Funkcija vraca najmanji neoznaceni broj sastavljen od istih bitova
    koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
    35  unsigned najmanji(unsigned x)
    {
    37  /* Rezultat se inicijalizuje vrednoscu 0 */
    unsigned rezultat = 0;
    39

    /* Promenljiva x se pomera u desno sve dok postoje jedinice u
    41  njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
    razlicita od nule). */
    43  for (; x != 0; x >>= 1) {
        /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
        45  detektuje na poziciji najmanje tezine u binarnoj
        reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
        47  sa desna u rezultat */
        if (x & 1) {
            49  rezultat <<= 1;
            rezultat |= 1;
        }
        51  }
    }
    53

    /* Vraca se dobijena vrednost */
    55  return rezultat;
    }

    57  int main()
    59  {
        int broj;

    61

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

    65  /* Ispisuju se, redom, najveci i najmanji broj formirani od bitova
    unetog broja */
    67  printf("Najveci:\n");
```

## 1 Uvodni zadaci

```
    stampaj_bitove(najveci(broj));
69
    printf("Najmanji:\n");
71    stampaj_bitove(najmanji(broj));
73
    return 0;
}
```

**Rešenje 1.8**      NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

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

4 /* Funkcija postavlja na nulu n bitova pocev od pozicije p. */
unsigned postavi_0(unsigned x, unsigned n, unsigned p)
6 {
8     /******
    Formira se maska cija binarna reprezentacija ima n bitova
    postavljenih na 0 pocev od pozicije p, dok su svi ostali
    postavljeni na 1. Na primer, za n=5 i p=10 formira se maska oblika
10    1111 1111 1111 1111 1111 1000 0011 1111
    To se postize na sledeci nacin:
12    ~0                                1111 1111 1111 1111 1111 1111 1111 1111
14    ~(~0 << n)                        1111 1111 1111 1111 1111 1111 1110 0000
    ~(~0 << n)                        0000 0000 0000 0000 0000 0000 0001 1111
16    ~(~0 << n) << (p-n+1)            0000 0000 0000 0000 0000 0111 1100 0000
    ~(~0 << n) << (p-n+1)            1111 1111 1111 1111 1111 1000 0011 1111
18    *****/
    unsigned maska = ~(~0 << n) << (p - n + 1);
20
    return x & maska;
22 }

24 /* Funkcija postavlja na jedinicu n bitova pocev od pozicije p. */
unsigned postavi_1(unsigned x, unsigned n, unsigned p)
26 {
28     /******
    Formira se maska kod koje je samo n bitova pocev od pocev od
    pozicije p jednako 1, a ostali su 0.
    Na primer, za n=5 i p=10 formira se maska oblika
30    0000 0000 0000 0000 0000 0111 1100 0000
    *****/
32    unsigned maska = ~(~0 << n) << (p - n + 1);
34
36    return x | maska;
    }
38

/* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
```



```

40     predstavlja n bitova pocev od pozicije p u binarnoj
      reprezentaciji broja x. */
42 unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
{
44
      /******
46      Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
      Na primer, za n=5
48      0000 0000 0000 0000 0000 0000 0001 1111
      *****/
50     unsigned maska = ~(~0 << n);

52     /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
      polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
54     zeljenih n i funkcija vraca tako dobijenu vrednost */
    return maska & (x >> (p - n + 1));
56 }

58 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
      postavljeni na vrednosti n bitova najmanje tezine binarne
      reprezentacije broja y */
60 unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p,
      unsigned y)
62 {
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 n
      bitova postavljenih na 0 pocevsi od pozicije p, dok su
68     ostali bitovi 1. */
    unsigned srednjih_n_0 = ~(~(~0 << n) << (p - n + 1));

70     /* U promenljivu x_postavi_0 se smesta vrednost dobijena kada se u
      binarnoj reprezentaciji vrednosti promenljive x postavi na 0 n
      bitova na pozicijama pocev od p */
72     unsigned x_postavi_0 = x & srednjih_n_0;

74     /* U promenljivu y_pomeri_srednje se smesta vrednost dobijena od
      binarne reprezentacije vrednosti promenljive y cijih je n bitova
      najnize tezine pomera tako da stoje pocev od pozicije p. Ostali
      bitovi su nule. Sa (y & poslednjih_n_1) postave na 0 svi bitovi
      osim najnizih n */
80     unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);

82     return x_postavi_0 ^ y_pomeri_srednje;
84 }

86 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
      njih n */
88 unsigned invertuj(unsigned x, unsigned n, unsigned p)
90 {

```

```

92  /* Formira se maska sa n jedinica pocev od pozicije p. */
    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
100 int main()
{
    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
    dobije kada se primeni funkcija postavi_0 za x, n i p*/
114  printf("x = %10u %36s = ", x, "");
    stampaj_bitove(x);
116  printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
    stampaj_bitove(postavi_0(x, n, p));
118  printf("\n");

120
    /* Ispisuju se binarne reprezentacije broja x i broja koji se
    dobije kada se primeni funkcija postavi_1 za x, n i p*/
122  printf("x = %10u %36s = ", x, "");
    stampaj_bitove(x);
124  printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
    stampaj_bitove(postavi_1(x, n, p));
126  printf("\n");

128
    /* Ispisuju se binarne reprezentacije broja x i broja koji se
    dobije kada se primeni funkcija vrati_bitove za x, n i p*/
130  printf("x = %10u %36s = ", x, "");
    stampaj_bitove(x);
132  printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
    stampaj_bitove(vrati_bitove(x, n, p));
134  printf("\n");

136
    /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
    dobije kada se primeni funkcija postavi_1_n_bitova za x, n i p*/
138  printf("x = %10u %36s = ", x, "");
    stampaj_bitove(x);
140  printf("y = %10u %36s = ", y, "");
    stampaj_bitove(y);
142

```

```

144 printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
    stampaj_bitove( postavi_1_n_bitova(x, n, p, y));
    printf("\n");
146
    /* Ispisuju se binarne reprezentacije broja x i broja koji se
148       dobije kada se primeni funkcija invertuj za x, n i p*/
    printf("x = %10u %36s = ", x, "");
150    stampaj_bitove(x);
    printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
152    stampaj_bitove( invertuj(x, n, p));
154
    return 0;
}

```

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

```

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

4 /* Funkcija ceo broj x rotira u levo za n mesta. */
unsigned rotiraj_ulevo(int x, unsigned n)
6 {
    unsigned bit_najvece_tezine;

8
    /* Maska koja ima samo bit na poziciji najvece tezine postavljen na
10     1 je neophodna da bi pre pomeranja u levo za 1 bit na poziciji
        najvece tezine bio sacuvan */
12    unsigned bit_najvece_tezine_maska = 1 << (sizeof(unsigned) * 8 - 1)
        ;
    int i;

14
    /* n puta se vrši rotaciju za jedan bit u levo. U svakoj iteraciji
16     se odredi bit na poziciji najvece tezine, a potom se pomera
        binarna reprezentacija trenutne vrednosti promenljive x u levo
18     za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
        na vrednost koju je imao bit na poziciji najvece tezine koji je
20     istisnut pomeranjem */
    for (i = 0; i < n; i++) {
22        bit_najvece_tezine = x & bit_najvece_tezine_maska;
        x = x << 1 | (bit_najvece_tezine ? 1 : 0);
24    }

26    /* Vraca se dobijena vrednost */
    return x;
28 }

30 /* Funkcija neoznaceni broj x rotira u desno za n mesta. */
unsigned rotiraj_udesno(unsigned x, unsigned n)
32 {

```

## 1 Uvodni zadaci

```
    unsigned bit_najmanje_tezine;
34    int i;

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

48    /* Vraca se dobijena vrednost */
50    return x;
}

52    /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
       argument funkcije x oznaceni ceo broj */
54    int rotiraj_udesno_oznaceni(int x, unsigned n)
56    {
        unsigned bit_najmanje_tezine;
58        int i;

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

70            if (x < 0)
72            /*****
               Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
74            na poziciji najveće tezine. Na primer ako je x
               1010 1011 1100 1101 1110 0001 0010 001b
76            (sa b je oznacen ili 1 ili 0 na poziciji najmanje tezine)
               Onda je sadržaj promenljive bit_najmanje_tezine:
78            0000 0000 0000 0000 0000 0000 0000 000b
               Nakon siftovanja sadržaja promenljive x za 1 u desno
80            1101 0101 1110 0110 1111 0000 1001 0001
               Kako bi umesto 1 na poziciji najveće tezine u trenutnoj binarnoj
82            reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
               poziciju najveće tezine jer bi se time dobile 0, a u ovom slucaju
84            su potrebne jedinice zbog bitovskog & zato se prvo vrši
```

```

86     komplementiranje, a zatim pomeranje
      ~bit_najmanje_tezine << (sizeof(int)*8 -1)
      B000 0000 0000 0000 0000 0000 0000 0000
88     gde B oznacava ~b.
      Potom se ponovo vrsi komplementiranje kako bi se b nalazilo na
90     poziciji najveće težine i sve jedinice na ostalim pozicijama
      ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
92     b111 1111 1111 1111 1111 1111 1111 1111
      *****/
94     x = (x >> 1) & ~(~bit_najmanje_tezine << (sizeof(int) * 8 - 1))
      ;
      else
96     x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
      }

98     /* Vraca se dobijena vrednost */
100    return x;
      }

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

106

      /* Ucitavaju se vrednosti sa standardnog ulaza */
108      printf("Unesite neoznaceni broj x:");
      scanf("%x", &x);
110      printf("Unesite neoznaceni broj n:");
      scanf("%x", &n);
112

      /* Ispisuje se binarna reprezentacija broja x */
114      printf("x\t\t\t\t\t= ");
      stampaj_bitove(x);
116

      /* Testira se rad napisanih funkcija */
118      printf("rotiraj_ulevo(%x,%u)\t\t= ", x, n);
      stampaj_bitove(rotiraj_ulevo(x, n));
120

      printf("rotiraj_udesno(%x,%u)\t\t= ", x, n);
122      stampaj_bitove(rotiraj_udesno(x, n));

      printf("rotiraj_udesno_oznaceni(%x,%u)\t\t= ", x, n);
124      stampaj_bitove(rotiraj_udesno_oznaceni(x, n));

      printf("rotiraj_udesno_oznaceni(-%x,%u)\t\t= ", x, n);
126      stampaj_bitove(rotiraj_udesno_oznaceni(-x, n));
128

130      return 0;
      }

```

## Rešenje 1.10

NAPOMENA: Rešenje koristi biblioteku za štampanje bi-

## 1 Uvodni zadaci

---

tova iz zadatka 1.5.

```
1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija vraća vrednost čija je binarna reprezentacija slika u
5    ogledalu binarne reprezentacije broja x. */
6 unsigned ogledalo(unsigned x)
7 {
8     unsigned najnizi_bit;
9     unsigned rezultat = 0;
10
11     int i;
12     /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
13        vrednosti broja x se određuje i pamti u promenljivoj
14        najnizi_bit, nakon čega se na promenljivu x primeni pomeranje u
15        desno */
16     for (i = 0; i < sizeof(x) * 8; i++) {
17         najnizi_bit = x & 1;
18         x >>= 1;
19         /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
20            postavljeni bitovi dobijaju veću poziciju. Novi bit se
21            postavlja na najnizu poziciju */
22         rezultat <<= 1;
23         rezultat |= najnizi_bit;
24     }
25
26     /* Vraća se dobijena vrednost */
27     return rezultat;
28 }
29
30 int main()
31 {
32     int broj;
33
34     /* Učitava se broj sa ulaza */
35     scanf("%x", &broj);
36
37     /* Ispisuje se njegova binarna reprezentacija */
38     stampaj_bitove(broj);
39
40     /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
41        funkcije ogledalo */
42     stampaj_bitove(ogledalo(broj));
43
44     return 0;
45 }
```

### Rešenje 1.11

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

```
2
4 /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
   jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
6 int broj_01(unsigned int n)
8 {
   int broj_nula, broj_jedinica;
   unsigned int maska;

10   broj_nula = 0;
   broj_jedinica = 0;

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

16   /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
      iteraciji pomera u desno pa ce jedini bit koji je postavljen na
      1 biti na svim pozicijama u binarnoj reprezentaciji maske */
20   while (maska != 0) {

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

30       /* Pomera se maska u desnu stranu */
       maska = maska >> 1;
32   }

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

40 int main()
42 {
   unsigned int n;

44   /* Ucitava se broj sa ulaza */
   scanf("%u", &n);

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

48   return 0;
50 }
```

## Rešenje 1.12

## 1 Uvodni zadaci

---

```
1  #include <stdio.h>
2
3  /* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju u
4     binarnom zapisu celog čneoznaenog broja x */
5  int broj_parova(unsigned int x)
6  {
7      int broj_parova;
8      unsigned int maska;
9
10     /* Vrednost promenljive koja predstavlja broj parova se
11        inicijalizuje na 0 */
12     broj_parova = 0;
13
14     /* Postavlja se maska tako da moze da procitamo da li su dva
15        najmanja bita u zapisu broja x 11 */
16     /* Binarna reprezentacija broja 3 je 000...00011 */
17     maska = 3;
18
19     while (x != 0) {
20         /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
21            */
22         if ((x & maska) == maska) {
23             broj_parova++;
24         }
25
26         /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
27            proveravao sledeci par bitova. Pomeranjem u desno bit najvece
28            tezine se popunjava nulom jer je x neoznaceni broj */
29         x = x >> 1;
30     }
31
32     /* Vraca se dobijena vrednost */
33     return broj_parova;
34 }
35
36 int main()
37 {
38     unsigned int x;
39
40     /* Ucitava se broj sa ulaza */
41     scanf("%u", &x);
42
43     /* Ispisuje se rezultat */
44     printf("%d\n", broj_parova(x));
45
46     return 0;
47 }
```

### Rešenje 1.14



```

2  #include <stdio.h>
4  /* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1 jer
   su za svaku heksadekadnu cifru potrebne 4 binarne cifre i jedna
   dodatna pozicija za terminirajucu nulu.
6  Prethodni izraz se moze zapisati kao sizeof(unsigned int)*2+1. */
   #define MAKS_DUZINA sizeof(unsigned int)*2 +1
8
10  /* Funkcija za neoznaceni broj x formira nisku s koja sadrzi njegov
   heksadekadni zapis */
   void prevod(unsigned int x, char s[])
12  {
14     int i;
       unsigned int maska;
       int vrednost;
16
18     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
       maska za citanje 4 uzastopne cifre */
       maska = 15;
20
22     /******
       Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
       tezine. Na primer za broj cija je binarna reprezentacija
24     0000000001101000100001111010101
       u prvom koraku se citaju bitovi izdvojeni sa <...>:
26     000000000110100010000111101<0101>
       u drugom koraku:
28     00000000011010001000011<1101>0101
       u trecem koraku:
30     0000000001101000100<0011>11010101 i tako redom...
32
       Indeks i oznacava poziciju na koju se smesta vrednost.
       *****/
34     for (i = MAKS_DUZINA - 2; i >= 0; i--) {
       /* Vrednost izdvojene cifre */
36         vrednost = x & maska;
38
       /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
       dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
       10 do 15 odgovarajuci karakter se dobija tako sto se prvo
       oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na tako
       dobijenu vrednost doda ASCII kod 'A' (time se dobija
       odgovarajuce slovo 'A', 'B', ... 'F') */
44         if (vrednost < 10) {
           s[i] = vrednost + '0';
46         } else {
           s[i] = vrednost - 10 + 'A';
48         }
50
       /* Primenljiva x se pomera za 4 bita u desnu stranu i time se u
       narednoj iteraciji posmatraju sledeca 4 bita */

```

## 1 Uvodni zadaci

```
52     x = x >> 4;
53 }
54
55 /* Upisuje se terminirajuca nula */
56 s[MAKS_DUZINA - 1] = '\0';
57 }
58
59 int main()
60 {
61     unsigned int x;
62     char s[MAKS_DUZINA];
63
64     /* Ucitava se broj sa ulaza */
65     scanf("%u", &x);
66
67     /* Poziva se funkcija za prevodjenje */
68     prevod(x, s);
69
70     /* I stampa se dobijena niska */
71     printf("%s\n", s);
72
73     return 0;
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     /* Ako je stepen paran */
32     if ((k % 2) == 0)
33         return stepen_2(x * x, k / 2);
34
35     /* Inace (ukoliko je stepen neparan) */
36     return x * stepen_2(x * x, k / 2);
37 }
38
39 int main()
40 {
41     int x, k, ind;
42
43     /* Ucitava se redni broj funkcije koja ce se primeniti */
44     printf("Unesite redni broj funkcije (1/2):\n");
45     scanf("%d", &ind);
46
47     /* Ucitavaju se vrednosti za x i k */
48     printf("Unesite broj x:\n");
49     scanf("%d", &x);
50     printf("Unesite broj k:\n");
51     scanf("%d", &k);
52
53     /* Ispisuje se vrednost koju vraca odgovarajuca funkcija */
54     if (x == 1)
55         printf("%d\n", stepen(x, k));
56     else if (x == 2)
57         printf("%d\n", stepen_2(x, k));
58     else {
59         fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
60         exit(EXIT_FAILURE);
61     }
62
63     exit(EXIT_SUCCESS);
64 }

```

### Rešenje 1.18

```

#include <stdio.h>
2
/* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
4     (posrednu) rekurziju */
6
/* Deklaracija funkcije neparan mora da bude navedena jer se ta
8     funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
    definicije. Funkcija je mogla biti deklarirana i u telu funkcije
    paran. */

```

## 1 Uvodni zadaci

---

```
10 unsigned neparan(unsigned n);

12 /* Funkcija vraca 1 ako broj n ima paran broj cifara, inace vraca 0
   */
   unsigned paran(unsigned n)
14 {
16     if (n <= 9)
18         return 0;
20     else
22         return neparan(n / 10);
24 }

26 /* Funkcija vraca 1 ako broj n ima neparan broj cifara, inace vraca
   0 */
   unsigned neparan(unsigned n)
28 {
30     if (n <= 9)
32         return 1;
34     else
36         return paran(n / 10);
38 }

40 int main()
42 {
44     int n;

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

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

56     return 0;
58 }
```

### Rešenje 1.19

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Pomocna funkcija koja izracunava n! * result. Koristi repnu
5    rekurziju. Result je argument u kome se akumulira do tada
6    izracunatu vrednost faktoriijela. Kada dodje do izlaza iz rekurzije
7    iz rekurzije potrebno je da vratimo result. */
8 int faktoriijel_repna(int n, int result)
9 {
10     if (n == 0)
11         return result;
12
13     return faktoriijel_repna(n - 1, n * result);
14 }
```

```
15 }
16
17 /* U sledecim funkcijama je prikazan postupak oslobadjanja od
18 repne rekurzije koja postoji u funkciji faktorijel_repna. */
19
20 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
21 naredbama kojima se vrednost argumenta funkcije postavlja na
22 vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
23 goto naredbe za vracanje na pocetak tela funkcije. */
24 int faktorijel_repna_v1(int n, int result)
25 {
26     pocetak:
27     if (n == 0)
28         return result;
29
30     result = n * result;
31     n = n - 1;
32     goto pocetak;
33 }
34
35 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
36 praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
37 iterativno resenje bez bezuslovnih skokova */
38 int faktorijel_repna_v2(int n, int result)
39 {
40     while (n != 0) {
41         result = n * result;
42         n = n - 1;
43     }
44
45     return result;
46 }
47
48 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
49 broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
50 ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
51 udobnosti korisnika, jer je sasvim prirodno da za faktorijel
52 zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
53 sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
54 faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
55 funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
56 poziv faktorijel_repna sa pozivom faktorijel_repna_v1, a zatim sa
57 pozivom funkcije faktorijel_repna_v2. */
58 int faktorijel(int n)
59 {
60     return faktorijel_repna(n, 1);
61 }
62
63 int main()
64 {
65     int n;
```

## 1 Uvodni zadaci

---

```
/* Ucitava se ceo broj */
67 printf("Unesite n (<= 12): ");
   scanf("%d", &n);
69 if (n > 12) {
       fprintf(stderr, "Greska: nedozvoljena vrednost za n!\n");
71   exit(EXIT_FAILURE);
   }
73
/* Ispisuje se rezultat */
75 printf("%d! = %d\n", n, faktorijel(n));
77 exit(EXIT_SUCCESS);
}
```

### Rešenje 1.21

```
1  #include <stdio.h>
3  /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
   int f_iterativna(int n, int a, int b)
5  {
       int i;
7   int f_0 = 0;
       int f_1 = 1;
9   int tmp;

11  if (n == 0)
       return 0;

13  for (i = 2; i <= n; i++) {
       tmp = a * f_1 + b * f_0;
15     f_0 = f_1;
       f_1 = tmp;
17  }

19  return f_1;
21 }

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

29     /* Rekurzivni pozivi */
31     return a * f_rekurzivna(n - 1, a, b) +
         b * f_rekurzivna(n - 2, a, b);
33 }

35 /* NAPOMENA: U slucaju da se rekurzijom problem svodi na vise manjih
```

```

37 podproblema koji se mogu preklapati, postoji opasnost da se
38 pojedini podproblemi manjih dimenzija resavaju veci broj puta.
39 Npr.  $F(20) = a \cdot F(19) + b \cdot F(18)$ , a  $F(19) = a \cdot F(18) + b \cdot F(17)$ , tj.
40 problem  $F(18)$  se resava dva puta! Problemi manjih
41 dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
42 problem je kombinacija rekurzije sa dinamickim programiranjem.
43 Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
44 memoriji (obicno u nizovima ili matricama), odakle se koriste ako
45 tokom resavanja ponovo budu potrebni.

46
47 U narednoj funkciji vec izracunati clanovi niza se cuvaju u
48 statickom nizu celih brojeva, jer se staticki niz ne smesta
49 na stek, kao sto je to slucaj sa lokalnim promenljivama, vec na
50 segment podataka, odakle je dostupan svim pozivima
51 rekurzivne funkcije. */

52
53 /* c) Funkcija racuna n-ti broj niza F - napredna rekurzivna
54 verzija */
55 int f_napredna(int n, int a, int b)
56 {
57     /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
58     staticke promenljive na podrazumevane vrednosti. Stoga, elemente
59     celobrojnog niza inicijalizuje na 0 */
60     static int f[20];

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

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

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

72
73
74 int main()
75 {
76     int n, a, b, ind;

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

83     /* Ucitavaju se koeficijenti a i b */
84     printf("Unesite koeficijente:\n");

```

## 1 Uvodni zadaci

---

```
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 poziva
95 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));

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)
6         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
16 int main()
{
18     unsigned int x;

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

22     /* Ispisuje se zbir cifara ucitanog broja */
    printf("%d\n", zbir_cifara(x));

24     return 0;
26 }
```

### Rešenje 1.23



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAKS_DIM 100
4
5  /* 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
7   element niza. */
8  int suma_niza_1(int *a, int n)
9  {
10     if (n <= 0)
11         return 0;
12
13     return suma_niza_1(a, n - 1) + a[n - 1];
14 }
15
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:");
38     scanf("%d", &n);
39
40     /* Ucitava se n elemenata niza. */
41     printf("Unesite elemente niza:");
42     for (i = 0; i < n; i++)
43         scanf("%d", &a[i]);
44
45     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
46 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{
```

## 1 Uvodni zadaci

---

```
52     fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
    exit(EXIT_FAILURE);
54 }

56     exit(EXIT_SUCCESS);
}
```

### Rešenje 1.24

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

4  /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
    dimenzije n */
6  int maksimum_niza(int niz[], int n)
{
8     /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je
        ujedno i jedini element niza */
10    if (n == 1)
        return niz[0];
12
14    /* Resavanje problema manje dimenzije */
    int max = maksimum_niza(niz, n - 1);

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

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

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

38    /* Stampa se maksimum unetog niza brojeva */
    printf("%d\n", maksimum_niza(brojevi, n));
40
42    return 0;
}
```

## Rešenje 1.25

```
1  #include <stdio.h>
2  #define MAKS_DIM 256
3
4  /* Funkcija koja izracunava skalarni proizvod dva data vektora */
5  int skalarno(int a[], int b[], int n)
6  {
7      /* Izlazak iz rekurzije: vektori su duzine 0 */
8      if (n == 0)
9          return 0;
10
11     /* Na osnovu resenja problema dimenzije n-1, resava se problem
12        dimenzije n primenom definicije skalarnog proizvoda
13        a*b = a[0]*b[0] + a[1]*b[1] +...+ a[n-2]*a[n-2] + a[n-1]*a[n-1]
14        Dakle, skalarni proizvod dva vektora duzine n se dobija kada se
15        na skalarni proizvod dva vektora duzine n-1 koji se dobiju od
16        polazna dva vektora otklanjanjem poslednjih elemenata, doda
17        proizvod poslednja dva elementa polaznih vektora. */
18     else
19         return skalarno(a, b, n - 1) + a[n - 1] * b[n - 1];
20 }
21
22 int main()
23 {
24     int i, a[MAKS_DIM], b[MAKS_DIM], n;
25
26     /* Unosi se dimenzija nizova */
27     printf("Unesite dimenziju nizova:");
28     scanf("%d", &n);
29
30     /* Provera da li je dimenzija niza odgovarajuca */
31     if (n < 0 || n > MAKS_DIM) {
32         printf("Dimenzija mora biti prirodan broj <= %d!\n", MAKS_DIM);
33         return 0;
34     }
35
36     /* A zatim i elementi nizova */
37     printf("Unesite elemente prvog niza:");
38     for (i = 0; i < n; i++)
39         scanf("%d", &a[i]);
40
41     printf("Unesite elemente drugog niza:");
42     for (i = 0; i < n; i++)
43         scanf("%d", &b[i]);
44
45     /* Ispisuje se rezultat skalarnog proizvoda dva ucitana niza */
46     printf("Skalarni proizvod je %d\n", skalarno(a, b, n));
47 }
```

```
    return 0;
49 }
```

### Rešenje 1.26

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

4 /* Funkcija koja racuna broj pojavljivanja elementa x u nizu a duzine
   n */
6 int br_pojave(int x, int a[], int n)
{
8     /* Izlazak iz rekurziije: 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 poziciji n-1 u
       nizu a nalazi broj x */
16    int bp = br_pojave(x, a, n - 1);
18    return a[n - 1] == x ? 1 + bp : bp;
}

20
22 int main()
{
24     int x, a[MAKS_DIM];
    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 tekućeg broja. U niz se ne moze
       učitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
       i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
34     printf("Unesite elemente niza:");
    i = 0;
36     while (scanf("%d", &a[i]) != EOF) {
        i++;
38         if (i == MAKS_DIM)
            break;
40     }
    n = i;
42

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

46     return 0;
```

```
}

```

### Rešenje 1.27

```

1  #include <stdio.h>
2  #define MAKS_DIM 256
3
4  /* Funkcija koja proverava da li su tri zadata broja uzastopni
   *   clanovi niza */
5
6  int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
7  {
8      /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i vraca
   *   se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi niza */
9      if (n < 3)
10         return 0;
11
12     /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
   *   dovoljno je da su oni poslednja tri clana niza ili da se oni
   *   rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
13     return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
14         || tri_uzastopna_clana(x, y, z, a, n - 1);
15 }
16
17
18
19
20 int main()
21 {
22     int x, y, z, a[MAKS_DIM];
23     int n;
24
25     /* Ucitavaju se tri cela broja za koje se ispituje da li su
   *   uzastopni clanovi niza */
26     printf("Unesite tri cela broja:");
27     scanf("%d%d%d", &x, &y, &z);
28
29     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
   *   Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
   *   ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
   *   i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
30     printf("Unesite elemente niza:");
31     int i = 0;
32     while (scanf("%d", &a[i]) != EOF) {
33         i++;
34         if (i == MAKS_DIM)
35             break;
36     }
37     n = i;
38
39     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana ispisuje
   *   se odgovarajuca poruka */
40     if (tri_uzastopna_clana(x, y, z, a, n))
41         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
42     else

```

## 1 Uvodni zadaci

---

```
48     printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
50     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 odgovarajuce maske proverava se
12     vrednost bita na poziciji najvece tezine i na osnovu toga se
        razlikuju dva slucaja. Ukoliko je na toj poziciji nula, onda je
14     broj jedinica u zapisu x isti kao broj jedinica u zapisu broja
        x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0. Ako je
16     na poziciji najvece tezine jedinica, rezultat dobijen
        rekurzivnim pozivom funkcije za x<<1 treba uvecati za jedan.
18     Za rekurzivni poziv se salje vrednost koja se dobija kada se x
        pomeri u levo. Napomena: argument funkcije x je oznacen ceo
20     broj, usled cega se ne koristi pomeranje udesno, jer funkciji
        moze biti prosledjen i negativan broj. Iz tog razloga,
22     odlucujemo se da proveramo najvisi, umesto najnizeg bita */
    if (x & (1 << (sizeof(x) * 8 - 1)))
24         return 1 + prebroj(x << 1);
    else
26         return prebroj(x << 1);
    /******
28     Krace zapisano
        return ((x & (1<<(sizeof(x)*8-1))) ? 1 : 0) + prebroj(x<<1);
30     *****/
}

32
int main()
34 {
    int x;

36
    /* Ucitava se ceo broj */
38     scanf("%x", &x);

40
    /* Ispisuje se rezultat */
    printf("%d\n", prebroj(x));
42
    return 0;
44 }
```

## Rešenje 1.30

```
1  #include <stdio.h>
2
3  /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
4     */
5  int maks_oktalna_cifra(unsigned x)
6  {
7      /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8         vrednost najveće oktalne cifre u broju 0 */
9      if (x == 0)
10         return 0;
11
12     /* Odredjivanje poslednje oktalne cifre u broju */
13     int poslednja_cifra = x & 7;
14
15     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
16        izbrise poslednja oktalna cifra */
17     int maks_bez_poslednje_cifre = maks_oktalna_cifra(x >> 3);
18
19     return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
20         : maks_bez_poslednje_cifre;
21 }
22
23 int main()
24 {
25     unsigned x;
26
27     /* Ucitava se neoznaceni ceo broj */
28     scanf("%u", &x);
29
30     /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
31     printf("%d\n", maks_oktalna_cifra(x));
32
33     return 0;
34 }
```

## Rešenje 1.31

```
1  #include <stdio.h>
2
3  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
4     broju */
5  int maks_heksadekadna_cifra(unsigned x)
6  {
7      /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8         vrednost najveće heksadekadne cifre u broju 0 */
```

## 1 Uvodni zadaci

```
10     if (x == 0)
11         return 0;
12
13     /* Odredjivanje poslednje heksadekadne cifre u broju */
14     int poslednja_cifra = x & 15;
15
16     /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
17        njega izbrise poslednja heksadekadna cifra */
18     int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);
19
20     return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
21         : maks_bez_poslednje_cifre;
22 }
23
24 int main()
25 {
26     unsigned x;
27
28     /* Ucitava se neoznaceni ceo broj */
29     scanf("%u", &x);
30
31     /* Ispisuje se vrednost najveće heksadekadne cifre unetog broja */
32     printf("%d\n", maks_heksadekadna_cifra(x));
33
34     return 0;
35 }
```

### Rešenje 1.32

```
1  #include <stdio.h>
2  #include <string.h>
3
4  /* Niska može imati najviše 31 karaktera + 1 za terminalnu nulu */
5  #define MAKS_DIM 32
6
7  /* Funkcija ispituje da li je zadana niska dužine n palindrom */
8  int palindrom(char s[], int n)
9  {
10     /* Izlaz iz rekurzije - trivijalno, niska dužine 0 ili 1 je
11        palindrom */
12     if ((n == 1) || (n == 0))
13         return 1;
14
15     /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
16        poslednji karakter i da je palindrom niska koja nastaje kada se
17        polaznoj nisci otklone prvi i poslednji karakter */
18     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
19 }
20
21 int main()
22 {
```



```

24   char s[MAKS_DIM];
    int n;

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

28   /* Odredjuje se duzina niske */
30   n = strlen(s);

32   /* Ispisuje se poruka 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

```

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

4
/* Funkcija koja ispisuje elemente niza a duzine n */
6  void ispisi_niz(int a[], int n)
{
8      int i;

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

14
/* Funkcija koja proverava da li se broj x nalazi u nizu a duzine n
   */
16  int koriscen(int a[], int n, int x)
{
18     int i;

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

26     /* Zakljucuje se da broj nije pronadjen */
    return 0;
28 }

30 /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n} dobija

```

```
32     kao argument niz a[] u koji se smesta permutacija, broj m oznacava
34     da se u okviru tog poziva funkcije na m-tu poziciju u permutaciji
36     smesta jedan od preostalih celih brojeva, n je velicina skupa koji
38     se permutuje. Funkciju se inicijalno poziva sa argumentom m = 1,
40     jer formiranje permutacije pocinje od pozicije broj 1. Stoga, a[0]
42     se ne koristi. */
44 void permutacija(int a[], int m, int n)
46 {
48     int i;
50
52     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
54     premasila velicinu skupa, onda se svi brojevi vec nalaze u
56     permutaciji i ispisuje se permutacija. */
58     if (m > n) {
60         ispisi_niz(a, n);
62         return;
64     }
66
68     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
70     mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
72     Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
74     ovako postavljenom pocetku permutacije. Kada se to zavrshi, vrshi
76     se provera da li postoji jos neki broj koji moze da se stavi na
78     m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
80     funkcija zavrшава sa radom. Ukoliko takav broj postoji, onda se
82     ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
84     ali sada sa drugacije postavljenim prefiksom. */
86     for (i = 1; i <= n; i++) {
88         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
90         pozicije, onda se on postavlja na poziciju m i poziva se
92         ponovo funkcija da dopuni ostatak permutacije posle upisivanja
94         i na poziciju m. Inace, nastavlja se dalje, trazeci broj koji
96         se nije pojavio do sada u permutaciji. */
98         if (!koriscen(a, m - 1, i)) {
100             a[m] = i;
102             permutacija(a, m + 1, n);
104         }
106     }
108 }
110
112 int main(void)
114 {
116     int n;
118     int a[MAKS_DUZINA_PERMUTACIJE+1];
120
122     /* Ucitava se broja n i provera se da li je u odgovarajucem opsegu
124     */
126     scanf("%d", &n);
128     if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
130         fprintf(stderr,
132             "Duzina permutacije mora biti broj iz intervala [0, %d]!\n",
134             n",
136         );
138     }
140 }
```

```

    MAKS_DUZINA_PERMUTACIJE);
82     exit(EXIT_FAILURE);
    }
84
    /* Ispisuju se permutacije duzine n */
86     permutacija(a, 1, n);
88     exit(EXIT_SUCCESS);
}

```

### Rešenje 1.34

```

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

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

16 /******
    Iterativno izracunavanje datog binomnog koeficijenta
18
    int binomni_koeficijent (int n, int k) {
20         int i, j, b;

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

24         return b;
    }

26
    Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
28     resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
    *****/
30
    /* 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
34     veca od sume elemenata prethodne hipotenuze. */
int suma_elemenata_hipotenuze(int n)
36 {
    return n > 0 ? 2 * suma_elemenata_hipotenuze(n - 1) : 1;
38 }

```

## 1 Uvodni zadaci

---

```
40 int main()
41 {
42     int n, k, i, d, r;
43
44     /* Ucitavaju se brojevi d i r */
45     scanf("%d %d", &d, &r);
46
47     /* Ispisuje se Paskalov trougao */
48     putchar('\n');
49     for (n = 0; n <= d; n++) {
50         for (i = 0; i < d - n; i++)
51             printf(" ");
52         for (k = 0; k <= n; k++)
53             printf("%4d", binomni_koeficijent(n, k));
54         putchar('\n');
55     }
56
57     /* Provera da li je r nenegativan */
58     if (r < 0) {
59         fprintf(stderr,
60             "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
61         );
62         exit(EXIT_FAILURE);
63     }
64
65     /* Ispisuje se suma elemenata hipotenuze */
66     printf("%d\n", suma_elementa_hipotenuze(r));
67
68     exit(EXIT_SUCCESS);
69 }
```

## 2

# Pokazivači

## 2.1 Pokazivačka aritmetika

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

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

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

### *Primer 1*

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

### *Primer 2*

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

[Rešenje 2.1]

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

## 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
Greska: neodgovarajuca dimenzija niza.
```

### Primer 4

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju niza: 101
Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.3]

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

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

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

### Primer 1

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

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

### Primer 2

```
POKRETANJE: ./a.out

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

[Rešenje 2.4]

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

### Primer 1

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

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

### Primer 2

```
POKRETANJE: ./a.out

INTERAKCIJA SA PROGRAMOM:
  Broj argumenata komandne linije koji
  koji su palindromi je 0.
```

[Rešenje 2.5]

**Zadatak 2.6** Napisati program koji kao prvi argument komandne linije prihvata putanju do datoteke za koju treba proveriti koliko reči ima  $n$  karaktera, gde se  $n$  zadaje kao drugi argument komandne linije. Smatrati da reč ne sadrži više od 100 karaktera. U zadatku ne koristiti ugrađene funkcije za rad sa niskama, već implementirati svoje koristeći pokazivačku sintaksu.

### Primer 1

```
POKRETANJE: ./a.out ulaz.txt 1

ULAZ.TXT
Ovo je sadržaj 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

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

INTERAKCIJA SA PROGRAMOM:
Greska: Nedovoljan broj argumenata
komandne linije.
Program se poziva sa
./a.out ime_dat br_karaktera.
```

### Primer 3

```
POKRETANJE: ./a.out ulaz.txt 2

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

[Rešenje 2.6]

**Zadatak 2.7** Napisati program koji kao prvi argument komandne linije prihvata putanju do datoteke za koju treba proveriti koliko reči ima zadati sufiks (ili prefiks), koji se zadaje kao drugi argument komandne linije. Smatrati da reč ne sadrži više od 100 karaktera. Program je neophodno pozvati sa jednom od opcija `-s` ili `-p` u zavisnosti od čega treba proveriti koliko reči ima zadati sufiks (ili prefiks). U zadatku ne koristiti ugrađene funkcije za rad sa niskama, već implementirati svoje koristeći pokazivačku sintaksu.

### Primer 1

```
POKRETANJE: ./a.out ulaz.txt ke -s

ULAZ.TXT
Ovo je sadržaj 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 sadržaj datoteke i u njoj ima reci
koje počinju sa sa

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



*Primer 3*

```

POKRETANJE: ./a.out ulaz.txt sa -p

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
  Greska: Neuspesno otvaranje
  datoteke ulaz.txt.

```

*Primer 4*

```

POKRETANJE: ./a.out ulaz.txt

ULAZ.TXT
  Ovo je sadrzaj ulaza.

INTERAKCIJA SA PROGRAMOM:
  Greska: Nedovoljan broj argumenata
  komandne linije.
  Program se poziva sa
  ./a.out ime_dat suf/pref -s/-p.

```

[Rešenje 2.7]

## 2.2 Višedimenzioni nizovi

**Zadatak 2.8** Data je kvadratna matrica dimenzije  $n \times n$ .

- Napisati funkciju koja izračunava trag matrice (sumu elemenata na glavnoj dijagonali).
- Napisati funkciju koja izračunava euklidsku normu matrice (koren sume kvadrata svih elemenata).
- Napisati funkciju koja izračunava gornju vandijagonalnu normu matrice (sumu apsolutnih vrednosti elemenata iznad glavne dijagonale).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati broj vrsta (ili kolona) kvadratne matrice  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrice. Na standardni izlaz ispisati 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, vrstu po vrstu:
  1 -2 3
  4 -5 6
  7 -8 9
  Trag matrice je 5.
  Euklidska norma matrice je 16.88.
  Vandijagonalna norma matrice je 11.

```

*Primer 2*

```

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

```

[Rešenje 2.8]

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

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

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

*Primer 1*

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

[Rešenje 2.9]

**Zadatak 2.10** Relacija se može predstaviti kvadratnom matricom nula i jedinica na sledeći način: element  $i$  je u relaciji sa elementom  $j$  ukoliko se u preseku  $i$ -te vrste i  $j$ -te kolone matrice nalazi broj 1, a nije u relaciji ukoliko se tu nalazi broj 0.

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

- (d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja 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.

- (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 učitanoj matrici.

### Primer 1

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

### Primer 2

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

[Rešenje 2.12]

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

- (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, vrstu po vrstu:
1 2 3
4 5 6
7 8 9
Spiralno ispisana matrica:
1 2 3 6 9 8 7 4 5
```

*Primer 2*

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

[Rešenje 2.13]

**Zadatak 2.14** Napisati funkciju koja izračunava  $k$ -ti stepen kvadratne matrice dimenzije  $n \times n$  ( $0 < n \leq 32$ ). Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati broj vrsta celobrojne matrice  $n$ , elemente matrice i stepen  $k$  ( $0 < k \leq 10$ ). Na standardni izlaz ispisati rezultat primene napisane funkcije. NAPOMENA: *Voditi računa da se prilikom stepenovanja matrice izvrši što manji broj množenja.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta kvadratne matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 2 3
4 5 6
7 8 9
Unesite stepen koji se racuna: 8
8. stepen matrice je:
510008400 626654232 743300064
1154967822 1419124617 1683281412
1799927244 2211595002 2623262760
```

## 2.3 Dinamička alokacija memorije

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

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.15]

**Zadatak 2.16** Napisati program koji sa standardnog ulaza učitava niz celih brojeva. Brojevi se unose sve dok se ne unese nula. Ne praviti nikakve pretpostavke o dimenziji niza. Na standardni izlaz ispisati ovaj niz brojeva u obrnutom poretku. Zadatak uraditi na dva načina:

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

Od korisnika sa ulaza tražiti da izabere način realokacije memorije.

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.16]

**Zadatak 2.17** Napisati funkciju koja kao rezultat vraća nisku koja se dobija nadovezivanjem dve niske, bez promene njihovog sadržaja. Napisati program koji testira rad napisane funkcije. Sa standardnog ulaza učitati dve niske karaktera (pretpostaviti da niske nisu duže od 50 karaktera i da ne sadrže praznine). Na standardni izlaz ispisati nisku koja se dobija njihovim nadovezivanjem. Za rezultujuću nisku dinamički alocirati memoriju.

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.17]

**Zadatak 2.18** Napisati program koji sa standardnog ulaza učitava matricu realnih brojeva. Prvo se učitavaju broj vrsta  $n$  i broj kolona  $m$  matrice (ne praviti nikakve pretpostavke o njihovoj veličini), a zatim i elementi matrice. Na standardni izlaz ispisati trag matrice.

### Primer 1

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

### Primer 2

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

[Rešenje 2.18]

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

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

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

Napisati programe koji testiraju napisanu biblioteku.

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

### Primer 1

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

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

### Primer 2

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

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



<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>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>	<pre>POKRETANJE: ./a.out ulaz.txt  ULAZ.TXT dimenzija: 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  IZLAZ: Neispravan pocetak fajla</pre>	<pre>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 matrice:
2 3
Unesite elemente matrice, vrstu po vrstu:
1 -2 3
-4 5 -6
Elementi ispod glavne dijagonale matrice:
1
-4 5
```

[Rešenje 2.20]

**Zadatak 2.21** Za zadatu matricu dimenzije  $n \times m$  napisati funkciju koja izračunava redni broj kolone matrice čiji je zbir maksimalan. Napisati program koji testira ovu funkciju. Sa standardnog ulaza učitati dimenzije matrice  $n$  i  $m$  (ne praviti nikakve pretpostavke o njihovoj veličini), a zatim elemente matrice. Na standardni izlaz ispisati redni broj kolone matrice sa maksimalnim zbirom. Ukoliko ima više takvih, ispisati prvu. NAPOMENA: *Koristiti biblioteku za rad sa celobrojnim matricama iz zadatka 2.19.*

### Primer 1

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

### Primer 2

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

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

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

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

### Primer 1

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

### Primer 1

```
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 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
```

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

`redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`

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

### Primer 1

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

```
INTERAKCIJA SA PROGRAMOM:
```

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

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

### Primer 1

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

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

### Primer 2

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

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

## 2.4 Pokazivači na funkcije

**Zadatak 2.27** Napisati program koji tabelarno štampa vrednosti proizvoljne realne funkcije sa jednim realnim argumentom, odnosno izračunava i ispisuje vrednosti date funkcije u  $n$  ekvidistantnih tačaka na intervalu  $[a, b]$ . Realni brojevi  $a$  i  $b$  ( $a < b$ ) kao i ceo broj  $n$  ( $n \geq 2$ ) se učitavaju sa standardnog ulaza. Ime funkcije se zadaje kao argument komandne linije (`sin`, `cos`, `tan`, `atan`, `acos`, `asin`, `exp`, `log`, `log10`, `sqrt`, `floor`, `ceil`, `sqr`).

### Primer 1

```
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 (vrednosti  $n$  i  $a$  uneti sa standardnog ulaza kao i ime funkcije):

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

*Primer 1*

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

*Primer 2*

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

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

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

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

*Primer 1*

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

*Primer 1*

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

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

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

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

### Primer 1

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

### Primer 2

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

## 2.5 Rešenja

### Rešenje 2.1

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

```

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

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

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

52
53 int main()
54 {
55     /* Deklarise se niz od najvise MAX elemenata */
56     int a[MAX];

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

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

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

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

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

77
78     obrni_niz_v1(a, n);

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

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

85
86     obrni_niz_v2(a, n);

```

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

### Rešenje 2.2

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

#define MAX 100

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

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

    return s;
}

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

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

    return p;
}

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

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



```
40     return min;
42 }

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

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

56     return max;
58 }

60 int main()
61 {
62     double a[MAX];
63     int n, i;

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

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

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

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

84     exit(EXIT_SUCCESS);
86 }
```

### Rešenje 2.3

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

```

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

```

## Rešenje 2.4

```

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

```

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

### Rešenje 2.5

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

## Rešenje 2.6

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Implementacija funkcije strlen() iz standardne biblioteke */
7  int duzina(char *s)
8  {
9      int i;
10     for (i = 0; *(s + i); i++);
11     return i;
12 }
13
14 int main(int argc, char **argv)
15 {
16     char rec[MAX_KARAKTERA+1];
17     int br = 0, n;
18     FILE *in;
19
20     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
21      */
22     if (argc < 3) {
23         printf("Greska: ");
24         printf("Nedovoljan broj argumenata komandne linije.\n");
25         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
26             argv[0]);
27         exit(EXIT_FAILURE);
28     }
29
30     /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
31      komandne linije. */
32     in = fopen(*(argv + 1), "r");
33     if (in == NULL) {
34         fprintf(stderr, "Greska: ");
35         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
36         exit(EXIT_FAILURE);
37     }
38
39     n = atoi(*(argv + 2));
40
41     /* Broje se reci cija je duzina jednaka broju zadatom drugim
42      argumentom komandne linije */
43     while (fscanf(in, "%s", rec) != EOF)
44         if (duzina(rec) == n)
45             br++;
46
47     printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);
48
49     /* Zatvara se datoteka */
50     fclose(in);
51 }
```

```
50     exit(EXIT_SUCCESS);
52 }
```

### Rešenje 2.7

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

#define MAX_KARAKTERA 100

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

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

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

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

/* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
```

```

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

```

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

### Rešenje 2.8

```
1  #include <stdio.h>
   #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++)
         for (j = 0; j < n; j++)
23             norma += M[i][j] * M[i][j];

25     return sqrt(norma);
27 }

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

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

39     return norma;
41 }

43 int main()
```



```

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

```

### Rešenje 2.9

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

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

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

### Rešenje 2.10

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

```

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

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

54     return 1;
}

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

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

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

74

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

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

```

```
90         return 0;

92     return 1;
93 }

94
96 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
97    relacije koriscenjem Varsalovog algoritma */
98 void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
99 {
100     int i, j, k;

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

106     /* Odredjuje se reflektivno zatvorenje matrice */
107     for (i = 0; i < n; i++)
108         zatvorenje[i][i] = 1;

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 }

120
122 /* Funkcija ispisuje elemente matrice */
123 void pisi_matricu(int m[][MAX], int n)
124 {
125     int i, j;

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
134 int main(int argc, char *argv[])
135 {
136     FILE *ulaz;
137     int m[MAX][MAX];
138     int pomocna[MAX][MAX];
139     int n, i, j;

140     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
141        */
```

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

```
194     /* Zatvara se datoteka */
    fclose(ulaz);

196     exit(EXIT_SUCCESS);
}
```

### Rešenje 2.11

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

4 #define MAX 32

6 /* Funkcija izracunava najveći element na sporednoj dijagonali. Za
   elemente sporedne dijagonale vazi da je zbir indeksa vrste i
   indeksa kolone jednak n-1 */
8 int max_sporedna_dijagonala(int m[][MAX], int n)
10 {
    int i;
12     int max_na_sporednoj_dijagonali = m[0][n - 1];

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

18     return max_na_sporednoj_dijagonali;
}

20
/* Funkcija izracunava indeks kolone najmanjeg elementa */
22 int indeks_min(int m[][MAX], int n)
{
24     int i, j;
    int min = m[0][0], indeks_kolone = 0;

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

32 }

34     return indeks_kolone;
}

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

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



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

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

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

62     return broj_negativnih;
63 }

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

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

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

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

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

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

```
196         max_sporedna_dijagonala(m, n));
198     printf("Indeks kolone sa najmanjim elementom je %d.\n",
199           indeks_min(m, n));
200
201     printf("Indeks vrste sa najvećim elementom je %d.\n",
202           indeks_max(m, n));
203
204     printf("Broj negativnih elemenata matrice je %d.\n",
205           broj_negativnih(m, n));
206
207     exit(EXIT_SUCCESS);
208 }
```

### Rešenje 2.12

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 32
5
6  /* Funkcija učitava elemente kvadratne matrice sa standardnog ulaza
7   */
8  void učitaj_matricu(int m[][MAX], int n)
9  {
10     int i, j;
11
12     for (i = 0; i < n; i++)
13         for (j = 0; j < n; j++)
14             scanf("%d", &m[i][j]);
15 }
16
17 /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
18 */
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, odnosno,
31    da li je normirana i ortogonalna. Matrica je normirana ako je
32    proizvod svake vrste matrice sa samom sobom jednak jedinici.
33    Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
34    vrste matrice jednak nuli */
35 int ortonormirana(int m[][MAX], int n)
```

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

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

### Rešenje 2.13

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 32
5
   /* Funkcija učitava elemente kvadratne matrice sa standardnog ulaza
      */
7  void učitaj_matricu(int m[][MAX], int n)
   {
9      int i, j;

11     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
13         scanf("%d", &m[i][j]);
   }

15
   /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
      */
17  void ispisi_matricu(int m[][MAX], int n)
   {
19     int i, j;

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

27
   /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
29     da li je normirana i ortogonalna. Matrica je normirana ako je
        proizvod svake vrste matrice sa samom sobom jednak jedinici.
31     Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
        vrste matrice jednak nuli */
33  int ortonormirana(int m[][MAX], int n)
   {
35     int i, j, k;
        int proizvod;

37
        /* Ispituje se uslov normiranosti */
39     for (i = 0; i < n; i++) {
        proizvod = 0;

41
        for (j = 0; j < n; j++)
43         proizvod += m[i][j] * m[i][j];
   }
```

```

45     if (proizvod != 1)
46         return 0;
47 }

48 /* Ispituje se uslov ortogonalnosti */
49 for (i = 0; i < n - 1; i++) {
50     for (j = i + 1; j < n; j++) {
51
52         proizvod = 0;
53
54         for (k = 0; k < n; k++)
55             proizvod += m[i][k] * m[j][k];
56
57         if (proizvod != 0)
58             return 0;
59     }
60 }

61 }

62 /* Ako su oba uslova ispunjena, matrica je ortonormirana */
63 return 1;
64 }

65
66 int main()
67 {
68     int A[MAX][MAX];
69     int n;
70
71     printf("Unesite broj vrsta matrice: ");
72     scanf("%d", &n);
73
74     if (n > MAX || n <= 0) {
75         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
76         fprintf(stderr, "matrice.\n");
77         exit(EXIT_FAILURE);
78     }
79
80     printf("Unesite elemente matrice, vrstu po vrstu:\n");
81     ucitaj_matricu(A, n);
82
83     printf("Matrica %s ortonormirana.\n",
84           ortonormirana(A, n) ? "je" : "nije");
85
86     exit(EXIT_SUCCESS);
87 }

```

### Rešenje 2.15

```

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

```

```
int main()
5 {
    int *p = NULL;
    7 int i, n;

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

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

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

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

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

    31 exit(EXIT_SUCCESS);
}
```

### Rešenje 2.16

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

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

    /* Velicina alocirane memorije */
    10 int alocirano;

    /* Broj elemenata niza */
    12 int n;

    /* Broj koji se učitava sa ulaza */
    14 int x;

    16 int i;
    18 int *b = NULL;
```

```
20 char realokacija;

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

24

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

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

30

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

34

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

38

40           if (b == NULL) {
               fprintf(stderr, "malloc(): ");
               fprintf(stderr, "greska pri alokaciji memorije.\n");
               free(a);
               exit(EXIT_FAILURE);
44           }

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

50           free(a);

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

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

60           a = (int *) realloc(a, alocirano * sizeof(int));
           if (a == NULL) {
               fprintf(stderr, "realloc(): ");
               fprintf(stderr, "greska pri alokaciji memorije.\n");
               exit(EXIT_FAILURE);
66           }

68       }
   }

70   a[n++] = x;
```

```
72     scanf("%d", &x);
74 }

76 printf("Niz u obrnutom poretku je: ");
77 for (n--; n >= 0; n--)
78     printf("%d ", a[n]);
79 printf("\n");
80
81 /* Oslobadja se dinamicki alocirana memorija */
82 free(a);
83
84 exit(EXIT_SUCCESS);
85 }
```

### Rešenje 2.17

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 1000
6
7 /* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
8  nadovezivanja niski. Adresa niza se vraća kao povratna vrednost.
9  */
10 char *nadovezi(char *s, char *t)
11 {
12     char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
13                               * sizeof(char));
14
15     /* Proverava se da li je memorija uspesno alocirana */
16     if (p == NULL) {
17         fprintf(stderr, "malloc(): ");
18         fprintf(stderr, "greska pri alokaciji memorije.\n");
19         exit(EXIT_FAILURE);
20     }
21
22     /* Kopiraju se i nadovezuju niske karaktera */
23     strcpy(p, s);
24     strcat(p, t);
25
26     return p;
27 }
28
29 int main()
30 {
31     char *s = NULL;
32     char s1[MAX], s2[MAX];
33
34     printf("Unesite dve niske karaktera:\n");
```



```

34  scanf("%s", s1);
    scanf("%s", s2);

36

    /* Poziva se funkcija koja nadovezuje niske */
38  s = nadovezi(s1, s2);

40  /* Prikazuje se rezultat */
    printf("Nadovezane niske: %s\n", s);
42

    /* Oslobadja se memorija alocirana u funkciji nadovezi() */
44  free(s);

46  exit(EXIT_SUCCESS);
}

```

### Rešenje 2.18

```

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

5  int main()
    {
7      int i, j;

9      /* Pokazivac na niz vrsta matrice realnih brojeva */
        double **A = NULL;

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

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

15     printf("Unesite broj vrsta i broj kolona matrice:\n ");
17     scanf("%d%d", &n, &m);

19     /* Dinamiki se alocira prostor za niz vrsta matrice */
        A = (double **)malloc(sizeof(double *) * n);

21     /* Provera se da li je doslo do greske pri alokaciji */
        if (A == NULL) {
23         fprintf(stderr, "malloc(): ");
25         fprintf(stderr, "greska pri alokaciji memorije.\n");
            exit(EXIT_FAILURE);
27         }

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

```

## 2 Pokazivači

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

### Rešenje 2.19

*matrica.h*

```
1  #ifndef _MATRICA_H_
2  #define _MATRICA_H_ 1
3
4  /* Funkcija dinamicki alocira memoriju za matricu dimenzija n x m */
5  int **alociraj_matricu(int n, int m);
6
7  /* Funkcija dinamicki alocira memoriju za kvadratnu matricu dimenzije
8     n */
9  int **alociraj_kvadratnu_matricu(int n);
10
11 /* Funkcija dealocira memoriju za matricu sa n vrsta */
```

```

13  int **deallociraj_matricu(int **matrica, int n);
14  /* Funkcija ucitava vec alociranu matricu dimenzija n x m sa
15     standardnog ulaza */
16  void ucitaj_matricu(int **matrica, int n, int m);
17  /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n sa
18     standardnog ulaza */
19  void ucitaj_kvadratnu_matricu(int **matrica, int n);
20  /* Funkcija ispisuje matricu dimenzija n x m na standardnom izlazu */
21  void ispisi_matricu(int **matrica, int n, int m);
22  /* Funkcija ispisuje kvadratnu matricu dimenzije n na standardnom
23     izlazu */
24  void ispisi_kvadratnu_matricu(int **matrica, int n);
25  /* Funkcija ucitava vec alociranu matricu dimenzija n x m iz datoteke
26     f */
27  int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
28      ;
29  /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n iz
30     datoteke f */
31  int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
32                                             FILE * f);
33  /* Funkcija upisuje matricu dimenzija n x m u datoteku f */
34  int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);
35  /* Funkcija upisuje kvadratnu matricu dimenzije n u datoteku f */
36  int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
37                                             FILE * f);
38  #endif

```

*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 ce
13        biti NULL, sto mora biti provereno u main funkciji */

```

```
15     if (matrica == NULL)
16         return NULL;
17
18     /* Alocira se prostor za svaku vrstu matrice */
19     for (i = 0; i < n; i++) {
20         matrica[i] = (int *) malloc(m * sizeof(int));
21         /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
22            prethodno alocirani resursi, i povratna vrednost je NULL */
23         if (matrica[i] == NULL) {
24             for (j = 0; j < i; j++)
25                 free(matrica[j]);
26             free(matrica);
27             return NULL;
28         }
29     }
30     return matrica;
31 }
32
33 int **alociraj_kvadratnu_matricu(int n)
34 {
35     /* Alociranje matrice dimenzije n x n */
36     return alociraj_matricu(n, n);
37 }
38
39 int **dealociraj_matricu(int **matrica, int n)
40 {
41     int i;
42     /* Oslobadja se prostor rezervisan za svaku vrstu */
43     for (i = 0; i < n; i++)
44         free(matrica[i]);
45     /* Oslobadja se memorija za niz pokazivaca na vrste */
46     free(matrica);
47
48     /* Matrica postaje prazna, tj. nealocirana */
49     return NULL;
50 }
51
52 void ucitaj_matricu(int **matrica, int n, int m)
53 {
54     int i, j;
55     /* Elementi matrice se ucitavaju po vrstama */
56     for (i = 0; i < n; i++)
57         for (j = 0; j < m; j++)
58             scanf("%d", &matrica[i][j]);
59 }
60
61 void ucitaj_kvadratnu_matricu(int **matrica, int n)
62 {
63     /* Ucitavanje matrice n x n */
64     ucitaj_matricu(matrica, n, n);
65 }
```

```

void ispisi_matricu(int **matrica, int n, int m)
67 {
    int i, j;
69     /* Ispis po vrstama */
    for (i = 0; i < n; i++) {
71         for (j = 0; j < m; j++)
            printf("%d ", matrica[i][j]);
73         printf("\n");
    }
75 }

void ispisi_kvadratnu_matricu(int **matrica, int n)
77 {
79     /* Ispis matrice n x n */
    ispisi_matricu(matrica, n, n);
81 }

83 int učitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
{
85     int i, j;
    /* Elementi matrice se učitavaju po vrstama */
87     for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
89             /* Ako je nemoguće učitati sledeći element, povratna vrednost
                funkcije je 1, kao indikator neuspešnog učitavanja */
                if (fscanf(f, "%d", &matrica[i][j]) != 1)
                    return 1;
93
    /* Uspešno učitana matrica */
95     return 0;
}

97 int učitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
99                                     FILE * f)
{
101     /* Učitavanje matrice n x n iz datoteke */
    return učitaj_matricu_iz_datoteke(matrica, n, n, f);
103 }

105 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
{
107     int i, j;
    /* Ispis po vrstama */
109     for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
111             /* Ako je nemoguće ispisati sledeći element, povratna vrednost
                funkcije je 1, kao indikator neuspešnog ispisa */
                if (fprintf(f, "%d ", matrica[i][j]) <= 0)
                    return 1;
113         fprintf(f, "\n");
115     }
117 }

```

```
119     /* Uspesno upisana matrica */
120     return 0;
121 }
122
123 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n, FILE * f
124 )
125 {
126     /* Ispis matrice n x n u datoteku */
127     return upisi_matricu_u_datoteku(matrica, n, n, f);
128 }
```

*main\_a.c*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "matrica.h"
4
5  int main()
6  {
7      int **matrica = NULL;
8      int n, m;
9      FILE *f;
10
11     /* Ucitavanje dimenzije matrice */
12     printf("Unesi broj vrsta matrice: ");
13     scanf("%d", &n);
14     printf("Unesi broj kolona matrice: ");
15     scanf("%d", &m);
16
17     /* Provera dimenzija matrice */
18     if (n <= 0 || m <= 0) {
19         fprintf(stderr, "Neodgovarajce dimenzije matrice\n");
20         exit(EXIT_FAILURE);
21     }
22
23     /* Alokacija matrice i provera alokacije */
24     matrica = alociraj_matricu(n, m);
25     if (matrica == NULL) {
26         fprintf(stderr, "Neuspesna alokacija matrice\n");
27         exit(EXIT_FAILURE);
28     }
29
30     /* Ucitavanje matrice sa standardnog ulaza */
31     printf("Unesi elemente matrice po vrstama:\n");
32     ucitaj_matricu(matrica, n, m);
33
34     /* Otvaranje fajla za upis matrice */
35     if ((f = fopen("matrica.txt", "w")) == NULL) {
36         fprintf(stderr, "fopen() error\n");
37         matrica = dealociraj_matricu(matrica, n);
38         exit(EXIT_FAILURE);
39     }
```

```

    }

40
    /* Upis matrice u fajl */
42    if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
        fprintf(stderr, "Neuspesno upisivanje matrice u datoteku\n");
44        matrica = dealociraj_matricu(matrica, n);
        exit(EXIT_FAILURE);
46    }

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

50
    /* Dealokacija matrice */
52    matrica = dealociraj_matricu(matrica, n);

54    exit(EXIT_SUCCESS);
}

```

*main\_b.c*

```

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

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

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

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

23     /* Ucitavanje dimenzije matrice */
       if (fscanf(f, "%d", &n) != 1) {
25         fprintf(stderr, "Neispravan pocetak fajla\n");
         exit(EXIT_FAILURE);
27     }

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

```

```
    exit(EXIT_FAILURE);
33 }

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

41 /* Ucitavanje matrice iz datoteke */
43 if (ucitaj_kvadratnu_matricu_iz_datoteke(matrica, n, f) != 0) {
    fprintf(stderr, "Neuspesno ucitavanje matrice iz datoteke\n");
45     matrica = dealociraj_matricu(matrica, n);
    exit(EXIT_FAILURE);
47 }

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

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

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

57 exit(EXIT_SUCCESS);
59 }
```

### Rešenje 2.20

```
1 #include <stdio.h>
#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 matrice:\n ");
    scanf("%d %d", &n, &m);

25

27  /* Alocira se matrica */
    matrica = alociraj_matricu(n, m);
    /* Provera alokacije */
29  if (matrica == NULL) {
        fprintf(stderr, "Neuspesna alokacija matrice\n");
31      exit(EXIT_FAILURE);
    }

33

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

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

39

41  /* Oslobadjanje memorije */
    matrica = deallociraj_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
    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 učitava elemente kvadratne matrice dimenzije n iz zadate
36   datoteke */
void učitaj_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 }

/* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
46   standardni izlaz */
void ispisi_matricu(float **m, int n)
48 {
    int i, j;
50
    for (i = 0; i < n; i++) {
52        for (j = 0; j < n; j++)
54            printf("%.2f ", m[i][j]);
            printf("\n");
56    }
}
58

/* Funkcija alokira memoriju za kvadratnu matricu dimenzije n */
60 float **alociraj_memoriju(int n)
{
    int i, j;
    float **m;
62
    m = (float **) malloc(n * sizeof(float *));
64
    if (m == NULL) {
66        fprintf(stderr, "malloc(): Neuspela alokacija\n");
68        exit(EXIT_FAILURE);
    }
70
    for (i = 0; i < n; i++) {
72        m[i] = (float *) malloc(n * sizeof(float));

        if (m[i] == NULL) {
74            printf("malloc(): neuspela alokacija memorije!\n");
76            for (j = 0; j < i; j++)
```

```

        free(m[i]);
78     free(m);
        exit(EXIT_FAILURE);
80     }
    }
82     return m;
}

84
/* Funkcija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
86     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     /* Ako 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     /* Otvara se datoteka za citanje */
    ulaz = fopen(argv[1], "r");
112     if (ulaz == NULL) {
        fprintf(stderr, "Greska: ");
114         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
116     }

118     /* Cita se dimenzija matrice */
    fscanf(ulaz, "%d", &n);
120

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

124     /* Ucitavaju se elementi matrice */
    ucitaj_matricu(ulaz, a, n);
126

    float zbir = zbir_ispod_sporedne_dijagonale(a, n);
```

```
128      /* Poziva se funkcija za transformaciju matrice */
130      izmeni(a, n);

132      /* Ispisuje se rezultat */
133      printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
134      printf("je %.2f.\n", zbir);

136      printf("Transformisana matrica je:\n");
137      ispisi_matricu(a, n);

138      /* Oslobadja se memorija */
139      oslobodi_memoriju(a, n);

142      /* Zatvara se datoteka */
143      fclose(ulaz);

144      exit(EXIT_SUCCESS);
146  }
```

### 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))
12  {
13      int i;
14      double x;
15
16      printf("-----\n");
17      for (i = 0; i < n; i++) {
18          x = a + i * (b - a) / (n - 1);
19          printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
20      }
21      printf("-----\n");
22  }
23
24  double sqr(double a)
25  {
26      return a * a;
27  }
28
29
```

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

    char ime_funkcije[6];

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

    /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
       */
    if (argc < 2) {
        printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
               argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Niska ime_funkcije sadrzi ime trazene funkcije koja je navedena
       u
       komandnoj liniji */
    strcpy(ime_funkcije, argv[1]);

    /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
       */
    if (strcmp(ime_funkcije, "sin") == 0)
        fp = &sin;
    else if (strcmp(ime_funkcije, "cos") == 0)
        fp = &cos;
    else if (strcmp(ime_funkcije, "tan") == 0)
        fp = &tan;
    else if (strcmp(ime_funkcije, "atan") == 0)
        fp = &atan;
    else if (strcmp(ime_funkcije, "acos") == 0)
        fp = &acos;
    else if (strcmp(ime_funkcije, "asin") == 0)
        fp = &asin;
    else if (strcmp(ime_funkcije, "exp") == 0)
        fp = &exp;
    else if (strcmp(ime_funkcije, "log") == 0)
        fp = &log;
    else if (strcmp(ime_funkcije, "log10") == 0)
        fp = &log10;
    else if (strcmp(ime_funkcije, "sqrt") == 0)
        fp = &sqrt;
    else if (strcmp(ime_funkcije, "floor") == 0)
        fp = &floor;
    else if (strcmp(ime_funkcije, "ceil") == 0)
        fp = &ceil;
```

```
81     else if (strcmp(ime_funkcije, "sqr") == 0)
        fp = &sqr;
83     else {
        printf("Program jos uvek ne podrzava trazenu funkciju!\n");
        exit(EXIT_SUCCESS);
85     }

87     printf("Unesite krajeve intervala:\n");
    scanf("%lf %lf", &a, &b);
89

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

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

101    /* Ispisuje se ime funkcije */
    printf("      x %10s(x)\n", ime_funkcije);
103

    /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
105       komandne linije */
    tabela(a, b, n, fp);
107

    exit(EXIT_SUCCESS);
109 }
```

## 3

# Algoritmi pretrage i sortiranja

## 3.1 Algoritmi pretrage

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

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

Napisati i program koji generiše rastući niz slučajnih brojeva dimenzije  $n$  i pozivajući napisane funkcije traži broj  $x$ . Programu se kao prvi argument komandne linije 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      VREMENA.TXT
|| 23542
||
|| Izlaz:
||   Linearna pretraga:      Dimenzija niza: 1000000
||   Element nije u nizu     Linearna: 3615091 ns
||   Binarna pretraga:      Binarna: 1536 ns
||   Element nije u nizu     Interpolaciona: 558 ns
||   Interpolaciona pretraga:
||   Element nije u nizu
```

#### Test 2

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

[Rešenje 3.1]

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

#### Primer 1

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

#### Primer 2

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

[Rešenje 3.2]



**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks ili prezime studenta čije informacije se potom prikazuju na ekranu. U slučaju više studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti `-indeks` ili `-prezime`. U slučaju neuspešnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složeno. 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 tačaka. U zavisnosti od prisustva opcija komandne linije (`-x` ili `-y`), pronaći onu koja je najbliža  $x$ , ili  $y$  osi, ili koordinatnom početku, ako

### 3 Algoritmi pretrage i sortiranja

---

nije prisutna nijedna opcija. Pretpostaviti da je broj tačaka u datateci veći od 0 i ne veći od 1024.

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

[Rešenje 3.5]

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

```
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.

### 3 Algoritmi pretrage i sortiranja

---

Test 1	Test 2	Test 3
ULAZ: 151 44 5 -12 -13 -15	ULAZ: 100 55 15 0 -15 -124 -155 -258 -315 -516	ULAZ: 100 15 11 8 7 5 4 3 2
IzLAZ: 3	IzLAZ: 4	IzLAZ: -1

[Rešenje 3.9]

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

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

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

Test 1	Test 2	Test 3
ULAZ: 4	ULAZ: 17	ULAZ: 1031
IzLAZ: 2 2	IzLAZ: 4 4	IzLAZ: 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*

```

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

```

*Primer 2*

```

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

```

*Primer 3*

```

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

```

## 3.2 Algoritmi sortiranja

**Zadatak 3.12** 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*.

*Test 1*

```

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

```

*Test 2*

```

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

```

*Test 3*

```

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

```

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

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

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

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

[Rešenje 3.15]

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

[Rešenje 3.16]

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

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

[Rešenje 3.17]

**Zadatak 3.18** Napisati program koji čita sadržaj dveju datoteka od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima, a u slučaju istog imena po prezimenima, i kreira jedinstven spisak studenata sortiranih takođe po istom kriterijumu. Program dobija nazive datoteka iz komandne linije i jedinstveni spisak upisuje u datoteku `ceo-tok.txt`. Pretpostaviti da je ime studenta nije duže od 10, a prezime od 15 karaktera.

*Test 1*

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

PRVI-DEO.TXT                                CEO-TOK.TXT
Andrija Petrovic                             Aleksandra Cvetic
Anja Ilic                                    Andrija Petrovic
Ivana Markovic                              Anja Ilic
Lazar Micic                                Bojan Golubovic
Nenad Brankovic                             Dragan Markovic
Sofija Filipovic                           Filip Dukic
Uros Milic                                 Ivana Markovic
Vladimir Savic                            Ivana Stankovic
                                           Lazar Micic
                                           Marija Stankovic
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 (najviše njih 1000) iz datoteke `biracki-spisak.txt` i kreira biračke spiskove. Jedan birački spisak je sortiran po imenu građana, a drugi po prezimenu. Program treba da ispisuje koliko građana ima isti redni broj u oba biračka spiska. Pretpostaviti da je za ime, odnosno prezime građana dovoljno 15 karaktera, i da se nijedno ime i prezime ne pojavljuje više od jednom.

*Test 1*

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

```
IZLAZ:
3
```

*Test 2*

```
BIRACKI-SPISAK.TXT
Milan Milicevic
```

```
IZLAZ:
1
```

*Test 3*

```
DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI
```

```
IZLAZ ZA GREŠKU:
Neupesno otvaranje
datoteke
biracki-spisak.txt.
```

[Rešenje 3.20]

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

### 3 Algoritmi pretrage i sortiranja

---

komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

#### Test 1

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

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

#### Test 2

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

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

**Zadatak 3.22** Napisati funkciju koja sortira niz niski po broju suglasnika u niski. Ukoliko reči imaju isti broj suglasnika tada sortirati ih po dužini niske rastuće, a ukoliko su i dužine jednake onda leksikografski rastuće. Napisati program koji testira ovu funkciju za niske koje se zadaju u datoteci `niske.txt`. 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 proizvođača 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 traženim kodom!
Unesite kod artikla [ili 0 za prekid]: 2530
Trazili ste: Napolitanke Bambi 230.00
Unesite kod artikla [ili 0 za prekid]: 0

UKUPNO: 230.00 dinara.

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

Kraj rada kase!

```

[Rešenje 3.23]

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

### 3 Algoritmi pretrage i sortiranja

---

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

#### Test 1

AKTIVNOSTI.TXT

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

DAT1.TXT

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

DAT2.TXT

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

DAT3.TXT

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

[Rešenje 3.24]

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

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

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

- nema opcija, sortiranje se vrši po broju gledanja;

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

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

Test 1	Test 2	Test 3
POKRETANJE: ./a.out	POKRETANJE: ./a.out -i	POKRETANJE: ./a.out -n
PESME.TXT	PESME.TXT	PESME.TXT
5	5	5
Ana - Nebo, 2342	Ana - Nebo, 2342	Ana - Nebo, 2342
Laza - Oblaci, 29	Laza - Oblaci, 29	Laza - Oblaci, 29
Pera - Ptice, 327	Pera - Ptice, 327	Pera - Ptice, 327
Jelena - Sunce, 92321	Jelena - Sunce, 92321	Jelena - Sunce, 92321
Mika - Kisa, 5341	Mika - Kisa, 5341	Mika - Kisa, 5341
IZLAZ:	IZLAZ:	IZLAZ:
Jelena - Sunce, 92321	Ana - Nebo, 2342	Mika - Kisa, 5341
Mika - Kisa, 5341	Jelena - Sunce, 92321	Ana - Nebo, 2342
Ana - Nebo, 2342	Laza - Oblaci, 29	Laza - Oblaci, 29
Pera - Ptice, 327	Mika - Kisa, 5341	Pera - Ptice, 327
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:
Unesi niz operacija: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5
0 2 8 2 6
```

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

### 3 Algoritmi pretrage i sortiranja

najmanja, najveća, itd... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene.

3	5	2	1
4	4	1__	2
5__	3	3	3
1	1	4	4
2	2__	5	5

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

#### Test 1

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

**Zadatak 3.28** 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 standardnom izlazu. 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]

### 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.29** Za zadanu kvadratnu matricu dimenzije  $n$  napisati funkciju koja sortira kolone matrice opadajuće na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. 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 standardnom izlazu 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
```

### 3 Algoritmi pretrage i sortiranja

---

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

#### Primer 1

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

#### Primer 2

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

[Rešenje 3.31]

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

#### Primer 1

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

#### Primer 2

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

#### Primer 3

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

[Rešenje 3.32]

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

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



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

#### Primer 1

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

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

[Rešenje 3.33]

**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 biblioteke `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:
```

```
Unesi sledeci datum: 13.12.2016.
postoji
Unesi sledeci datum: 10.5.2015.
ne postoji
Unesi sledeci datum: 5.2.2009.
postoji
```

## 3.4 Rešenja

### Rešenje 3.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
7    -lrt zbog funkcije clock_gettime() */
8
9 /* Naredne tri funkcije koje vrse pretragu, ukoliko se trazeni
10    element pronadje u nizu, vracaju indeks pozicije na kojoj je
11    element pronadjen. Ovaj indeks je uvek nenegativan. Ako element
12    nije pronadjen u nizu, funkcije vracaju negativnu vrednost -1, kao
13    indikator neuspesne pretrage. */
14
15 /* Linearna pretraga: Funkcija pretrazuje niz a[] celih brojeva
16    duzine n, trazeci u njemu prvo pojavljivanje elementa x. Pretraga
17    se 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     else
48         /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
49            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 on
69            nije u tom delu niza. Ova provera je neophodna, da se ne bi
70            dogodilo da se prilikom izracunavanja indeksa srednji izadje
71            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 x
76            jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
77            desni). Ova provera je neophodna, jer bi se u suprotnom
78            prilikom izracunavanja indeksa srednji pojavilo deljenje
79            nulom. */
80         else if (a[levi] == a[desni])
81             return levi;
82         /* Racunanje srednjeg indeksa */
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 ce
88            verovatno biti blize trazenoj vrednosti nego da je prosto uvek
89            uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
90            porediti sa pretragom recnika: ako neko trazi rec na slovo
91            'B', sigurno nece da otvori recnik na polovini, vec verovatno
92            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 se
98            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 se
103            pretraga završava na poziciji srednji */
104         return srednji;
105 }
106 /* U slucaju neuspesne pretrage vraca 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             "koriscenje programa: %s dim_niza broj\n", argv[0]);
120         exit(EXIT_FAILURE);
121     }
122
123     /* Dimenzija niza */
124     n = atoi(argv[1]);
125     if (n > MAX || n <= 0) {
126         fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
127         exit(EXIT_FAILURE);
128     }
129
130     /* Broj koji se trazi */
131     x = atoi(argv[2]);
132     /* Elementi niza se generisu slucajno, tako da je svaki sledeci
133        veci od prethodnog. Funkcija srandom() inicijalizuje pocetnu
134        vrednost sa kojom se kreće u izracunavanje sekvence
135        pseudo-slucajnih brojeva. Kako generisani niz ne bi uvek bio
136        isti, ova vrednost se postavlja na tekuce vreme u sekundama od
137        Nove godine 1970, tako da je za svako sledece pokretanje
138        programa (u vremenskim intervalima vecim od jedne sekunde) ove
139        vrednost drugacija. random()%100 vraca brojeve izmedju 0 i 99 */
140     srandom(time(NULL));
141     for (i = 0; i < n; i++)
142         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
143     /* Linearna pretraga */
144     printf("Linearna pretraga:\n");
145     /* Vreme proteklo od Nove godine 1970 */
146     clock_gettime(CLOCK_REALTIME, &vreme1);
147     i = linearna_pretraga(a, n, x);
148     /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
149        linearnu pretragu */
149     clock_gettime(CLOCK_REALTIME, &vreme2);
```

```

151     if (i == -1)
152         printf("Element nije u nizu\n");
153     else
154         printf("Element je u nizu na poziciji %d\n", i);
155     /* Binarna pretraga */
156     printf("Binarna pretraga:\n");
157     clock_gettime(CLOCK_REALTIME, &vreme3);
158     i = binarna_pretraga(a, n, x);
159     clock_gettime(CLOCK_REALTIME, &vreme4);
160     if (i == -1)
161         printf("Element nije u nizu\n");
162     else
163         printf("Element je u nizu na poziciji %d\n", i);
164     /* Interpolaciona pretraga */
165     printf("Interpolaciona pretraga:\n");
166     clock_gettime(CLOCK_REALTIME, &vreme5);
167     i = interpolaciona_pretraga(a, n, x);
168     clock_gettime(CLOCK_REALTIME, &vreme6);
169     if (i == -1)
170         printf("Element nije u nizu\n");
171     else
172         printf("Element je u nizu na poziciji %d\n", i);
173     /* Podaci o izvršavanju programa bivaju upisani u log fajl */
174     if ((f = fopen("vremena.txt", "a")) == NULL) {
175         fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
176         exit(EXIT_FAILURE);
177     }

179     fprintf(f, "Dimenzija niza: %d\n", n);
180     fprintf(f, "\tLinearna: %10ld ns\n",
181             (vreme2.tv_sec - vreme1.tv_sec) * 1000000000 +
182             vreme2.tv_nsec - vreme1.tv_nsec);
183     fprintf(f, "\tBinarna: %19ld ns\n",
184             (vreme4.tv_sec - vreme3.tv_sec) * 1000000000 +
185             vreme4.tv_nsec - vreme3.tv_nsec);
186     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
187             (vreme6.tv_sec - vreme5.tv_sec) * 1000000000 +
188             vreme6.tv_nsec - vreme5.tv_nsec);
189     /* Zatvaranje datoteke */
190     fclose(f);

191     exit(EXIT_SUCCESS);
192 }

```

## Rešenje 3.2

```

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

4 #define MAX 1024

```

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

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

32 int binarna_pretraga_r(int a[], int l, int d, int x)
{
34     int srednji;
    if (l > d)
36         return -1;
    /* Sredisnja pozicija na kojoj se trazi vrednost x */
38    srednji = (l + d) / 2;
    /* Ako je element na sredisnjoj poziciji trazeni */
40    if (a[srednji] == x)
        return srednji;
42    /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
        pretrazuje se desna polovina niza */
44    if (a[srednji] < x)
        return binarna_pretraga_r(a, srednji + 1, d, x);
46    /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
        pretrazuje se leva polovina niza */
48    else
        return binarna_pretraga_r(a, l, srednji - 1, x);
50 }

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



```

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

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

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

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

92     /* Linearna pretraga */
    printf("Linearna pretraga\n");
94     indeks = linearna_pretraga_r1(a, i, x);
    if (indeks == -1)
96         printf("Element se ne nalazi u nizu.\n");
    else
98         printf("Pozicija elementa je %d.\n", indeks);

100    /* Binarna pretraga */
    printf("Binarna pretraga\n");
102    indeks = binarna_pretraga_r(a, 0, i - 1, x);
    if (indeks == -1)
104        printf("Element se ne nalazi u nizu.\n");
    else
106        printf("Pozicija elementa je %d.\n", indeks);

108    /* Interpolaciona pretraga */

```

### 3 Algoritmi pretrage i sortiranja

---

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

#### 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 za
12     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
/* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
24 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;
    int desni = n - 1;
30     int srednji;
    /* Dokle god je indeks levi levo od indeksa desni */
32     while (levi <= desni) {
        /* Racuna se srednja pozicija */
34         srednji = (levi + desni) / 2;
        /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
36         se trazeni indeks mora nalaziti u levoj polovini niza */
        if (x < a[srednji].indeks)
38             desni = srednji - 1;
        /* Ako je pak manji od trazenog, tada se on mora nalaziti u
```

```

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

52 /* Linearnom pretragom niza studenata traži 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         /* Poređenje prezimena i-tog studenta i poslatog x */
58         if (strcmp(a[i].prezime, x) == 0)
59             return i;
60     return -1;
61 }

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

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

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

```

```
92     }

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

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

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

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

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

```

144     }
146     exit(EXIT_SUCCESS);
}

```

### Rešenje 3.4

```

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

5  #define MAX_STUDENATA 128
   #define MAX_DUZINA 16

7
   typedef struct {
9       long indeks;
       char ime[MAX_DUZINA];
11      char prezime[MAX_DUZINA];
   } Student;

13
   int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
15                                   long x)
   {
17       /* Ako je pozicija elementa na levom kraju veca od pozicije
           elementa na desnom kraju dela niza koji se pretrazuje, onda se
19       zapravo pretrazuje prazan deo niza. U praznom delu niza nema
           trazenog elementa pa se vraca -1 */
21       if (levi > desni)
           return -1;
23       /* Racunanje pozicije srednjeg elementa */
       int srednji = (levi + desni) / 2;
25       /* Da li je srednji bas onaj trazeni */
       if (a[srednji].indeks == x) {
27           return srednji;
       }
29       /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
           poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
31       je poznato da je niz sortiran po indeksu u rastucem poretku. */
       if (x < a[srednji].indeks)
33           return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
       /* Inace ga treba traziti u desnoj polovini */
35       else
           return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37   }

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

```

```
45     prvim studentom u nizu. */
46     if (strcmp(a[0].prezime, x) == 0)
47         return 0;
48     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
49     return i >= 0 ? 1 + i : -1;
50 }
51
52 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
53 {
54     /* Ako je niz prazan, vraca se -1 */
55     if (n == 0)
56         return -1;
57     /* Ako se trazi poslednji student sa trazanim prezimenom, pocinje
58        se sa poslednjim studentom u nizu. */
59     if (strcmp(a[n - 1].prezime, x) == 0)
60         return n - 1;
61     return linearna_pretraga_rekurzivna(a, n - 1, x);
62 }
63
64 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
65    prosledjuju kao argumenti komandne linije */
66 int main(int argc, char *argv[])
67 {
68     Student dosije[MAX_STUDENATA];
69     FILE *fin = NULL;
70     int i;
71     int br_studenata = 0;
72     long trazen_indeks = 0;
73     char trazeno_prezime[MAX_DUZINA];
74     int bin_pretraga;
75
76     /* Provera da li je korisnik prilikom poziva programa prosledio ime
77        datoteke sa informacijama o studentima i opciju pretrage */
78     if (argc != 3) {
79         fprintf(stderr,
80             "Greska: Program se poziva sa %s ime_datoteke opcija\n",
81             argv[0]);
82         exit(EXIT_FAILURE);
83     }
84
85     /* Provera prosledjene opcije */
86     if (strcmp(argv[2], "-indeks") == 0)
87         bin_pretraga = 1;
88     else if (strcmp(argv[2], "-prezime") == 0)
89         bin_pretraga = 0;
90     else {
91         fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
92         exit(EXIT_FAILURE);
93     }
94
95     /* Otvaranje datoteke */
96     fin = fopen(argv[1], "r");
```

```
97  if (fin == NULL) {
98      fprintf(stderr,
99          "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
100      exit(EXIT_FAILURE);
101  }

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

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

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

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

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

137      /* Rezultat linearne pretrage */
138      if (i == -1)
139          printf("Ne postoji student sa prezimenom %s\n",
140                 trazeno_prezime);
141      else
142          printf("Indeks: %ld, Ime i prezime: %s %s\n",
143                 dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
144  }
145  }
```

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

#### Rešenje 3.5

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



```

47     return najbliza;
48 }
49
50 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
51    t dimenzije n */
52 Tacka najbliza_y_osi(Tacka t[], int n)
53 {
54     Tacka najbliza;
55     int i;
56     najbliza = t[0];
57     for (i = 1; i < n; i++) {
58         if (fabs(t[i].y) < fabs(najbliza.y)) {
59             najbliza = t[i];
60         }
61     }
62     return najbliza;
63 }
64
65 #define MAX 1024
66
67 int main(int argc, char *argv[])
68 {
69     FILE *ulaz;
70     Tacka tacke[MAX];
71     Tacka najbliza;
72     int i, n;
73
74     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75        ime datoteke sa tackama */
76     if (argc < 2) {
77         fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
79         exit(EXIT_FAILURE);
80     }
81
82     /* Otvaranje datoteke za citanje */
83     ulaz = fopen(argv[1], "r");
84     if (ulaz == NULL) {
85         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
87         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. Ako nema

```

### 3 Algoritmi pretrage i sortiranja

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

119     /* Stampanje koordinata trazene tacke */
120     printf("%g %g\n", najbliza.x, najbliza.y);

121     /* Zatvaranje datoteke */
122     fclose(ulaz);

123     exit(EXIT_SUCCESS);
124 }
125 }
```

#### Rešenje 3.6

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

4 /* Tacnost */
5 #define EPS 0.001

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

10
11     /* Kod intervala [0, 2] leva granica je 0, a desna 2 */
12     l = 0;
13     d = 2;

14
15     /* Sve dok se ne pronadje trazena vrednost argumenta */
16     while (1) {
17         /* Polovi se interval */
18         s = (l + d) / 2;
19         /* Ako je apsolutna vrednost kosinusa u ovoj tacki manja od
20            zadate tacnosti, prekida se pretraga */
21     }
```

```

    if (fabs(cos(s)) < EPS) {
22         break;
    }
24     /* Ako je nula u levom delu intervala, nastavlja se pretraga na
        [l, s] */
26     if (cos(l) * cos(s) < 0)
        d = s;
28     else
        /* Inace, na intervalu [s, d] */
30         l = s;
    }

32     /* Stampanje vrednosti trazene tacke */
34     printf("%g\n", s);

36     return 0;
}

```

### Rešenje 3.7

```

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

6 int main(int argc, char **argv)
{
8     double l, d, s, epsilon;

10     char ime_funkcije[6];

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

16     /* Ako korisnik nije uneo argument, prijavljuje se greska */
    if (argc != 2) {
18         fprintf(stderr, "Greska: ");
        fprintf(stderr, "Nedovoljan broj argumenata komandne linije.\n");
20         fprintf(stderr,
            "Program se poziva sa %s ime_funkcije iz math.h.\n",
22             argv[0]);
        exit(EXIT_FAILURE);
24     }

26     /* Niska ime_funkcije sadrzi ime trazene funkcije koja je navedena
        u komandnoj liniji */
28     strcpy(ime_funkcije, argv[1]);

30     /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
    if (strcmp(ime_funkcije, "sin") == 0)

```

```
32     fp = &sin;
33     else if (strcmp(ime_funkcije, "cos") == 0)
34         fp = &cos;
35     else if (strcmp(ime_funkcije, "tan") == 0)
36         fp = &tan;
37     else if (strcmp(ime_funkcije, "atan") == 0)
38         fp = &atan;
39     else if (strcmp(ime_funkcije, "asin") == 0)
40         fp = &asin;
41     else if (strcmp(ime_funkcije, "log") == 0)
42         fp = &log;
43     else if (strcmp(ime_funkcije, "log10") == 0)
44         fp = &log10;
45     else {
46         fprintf(stderr, "Program ne podrzava trazenu funkciju!\n");
47         exit(EXIT_SUCCESS);
48     }
49
50     printf("Unesite krajeve intervala: ");
51     scanf("%lf %lf", &l, &d);
52
53     if ((*fp) (l) * (*fp) (d) >= 0) {
54         fprintf(stderr,
55             "%s na intervalu [%g, %g] ne zadovoljava uslove\n",
56             ime_funkcije, l, d);
57         exit(EXIT_FAILURE);
58     }
59
60     printf("Unesite preciznost: ");
61     scanf("%lf", &epsilon);
62
63     /* Sve dok se ne pronadje trazena vrednost argumenta */
64     while (1) {
65         /* Polovi se interval */
66         s = (l + d) / 2;
67         /* Ako je apsolutna vrednost trazene funkcije u ovoj tacki manja
68            od zadate tacnosti, prekida se pretraga */
69         if (fabs((*fp) (s)) < epsilon) {
70             break;
71         }
72         /* Ako je nula u levom delu intervala, nastavlja se pretraga na
73            [l, s] */
74         if ((*fp) (l) * (*fp) (s) < 0)
75             d = s;
76         else
77             /* Inace, na intervalu [s, d] */
78             l = s;
79     }
80
81     /* Stampanje vrednosti trazene tacke */
82     printf("%g\n", s);
```

```
84     return 0;
    }
```

### Rešenje 3.8

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

#define MAX 256

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

int main()
{
    int niz[MAX];
    int n = 0;

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

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

    return 0;
}
```

#### Rešenje 3.9

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

#### Rešenje 3.10

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

```
3 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 }

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 }

37 int main()
38 {
39     unsigned int x;

41     /* Unos podatka */
42     scanf("%u", &x);

43     /* Provera da li je uneti broj pozitivan */
44     if (x == 0) {
45         fprintf(stderr, "Logaritam od nule nije definisan\n");
46         exit(EXIT_FAILURE);
47     }

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

52     exit(EXIT_SUCCESS);
53 }
```

#### Rešenje 3.12

*sort.h*

```
1  #ifndef _SORT_H_
2  #define _SORT_H_ 1

4  /* Selection sort: Funkcija sortira niz celih brojeva metodom
   sortiranja izborom. Ideja algoritma je sledeca: U svakoj
6  iteraciji pronalazi se najmanji element i premesta se na pocetak
   niza. Dakle, u prvoj iteraciji, pronalazi se najmanji element, i
8  dovodi na nulto mesto u nizu. U i-toj iteraciji najmanjih i-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 element
18 a[i] umetnuti na pravu poziciju medju prvih i elemenata tako da se
   dobije niz duzine i+1 koji je sortiran. Ovo se radi tako sto se
20 i-ti element najpre uporedi sa njegovim prvim levim susedom
   (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i niz
22 a[0],a[1],...,a[i] je sortiran, pa se moze preci na sledecu
   iteraciju. Ako je a[i-1] vece, tada se zamenjuju a[i] i a[i-1], a
24 zatim se proverava da li je potrebno dalje potiskivanje elementa u
   levo, poredeci ga sa njegovim novim levim susedom. Ovim uzastopnim
26 premestanjem se a[i] umece na pravo mesto u nizu. */
void insertion_sort(int a[], int n);

28

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

40

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
44 vise puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
   koji je manji od n (sto omogucuje razmenu udaljenih elemenata) i
46 tako se dobija h-sortiran niz, tj. niz u kome su elementi na
   rastojanju h sortirani, mada susedni elementi to ne moraju biti. U
```



```

48     drugom prolazu kroz isti algoritam sprovodi se isti postupak ali
49     za manji korak h. Sa prolazima se nastavlja sve do koraka h = 1, u
50     kome se dobija potpuno sortirani niz. Izbor pocetne vrednosti za
51     h, i nacina njegovog smanjivanja menja u nekim slucajevima brzinu
52     algoritma, ali bilo koja vrednost ce rezultovati ispravnim
53     sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
54     vrednost 1. */
55     void shell_sort(int a[], int n);
56
57     /* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
58     Sortiranje se vrši od elementa na poziciji l do onog na poziciji
59     d. Na pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a
60     d je jednako poslednjem validnom indeksu u nizu. Funkcija niz
61     podeli na dve polovine, levu i desnu, koje zatim rekurzivno
62     sortira. Od ova dva sortirana podniza, sortiran niz se dobija
63     ucesljavanjem, tj. istovremenim prolaskom kroz oba niza i izborom
64     trenutnog manjeg elementa koji se smesta u pomocni niz. Na kraju
65     algoritma, sortirani elementi su u pomocnom nizu, koji se kopira u
66     originalni niz. */
67     void merge_sort(int a[], int l, int d);
68
69     /* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
70     i d. Njena ideja sortiranja je izbor jednog elementa niza, koji se
71     naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
72     svi elementi levo od njega bice manji, a svi desno bice veci od
73     njega. Kako je pivot doveden na svoje mesto, da bi niz bio
74     kompletno sortiran, potrebno je sortirati elemente levo (manje) od
75     njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
76     manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
77     deo moze se uraditi rekurzivno. */
78     void quick_sort(int a[], int l, int d);
79
80 #endif

```

sort.c

```

1 #include "sort.h"
2
3 #define MAX 1000000
4
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++) {

```

### 3 Algoritmi pretrage i sortiranja

---

```
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 ako
24         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 a[0],...,a[i-1]
38         sortiran */
39      for (i = 1; i < n; i++) {
40          /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
41             potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...,a[i]
42             bude sortiran. Indeks j je trenutna pozicija na kojoj se
43             element koji se umeće nalazi. Petlja se završava ili kada
44             element dodje do levog kraja (j==0) ili kada se naidje na
45             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;
56      int ind;
57
58      for (i = n, ind = 1; i > 1 && ind; i--)
59          /* Poput "mehurica" potiskuje se najveći element medju elementima
60             od a[0] do a[i-1] na poziciju i-1 upoređujući susedne
61             elemente niza i potiskujući veci u desno */
62          for (j = 0, ind = 0; j < i - 1; j++)
63              if (a[j] > a[j + 1]) {
64                  int temp = a[j];
65                  a[j] = a[j + 1];
66                  a[j + 1] = temp;
67                  /* Promenljiva ind registruje da je bilo premestanja. Samo u
```

```

68         tom slucaju ima smisla ici na sledecu iteraciju, jer ako
        nije bilo premestanja, znaci da su svi elementi vec u
70     dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
        niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
72     ispravno, ali manje efikasano, jer bi se cesto nepotrebno
        vrsila mnoga uporedjivanja, kada je vec jasno da je
74     sortiranje zavrшено. */
        ind = 1;
76     }
    }
78
    void shell_sort(int a[], int n)
80 {
        int h = n / 2, i, j;
82     while (h > 0) {
        /* Insertion sort sa korakom h */
84         for (i = h; i < n; i++) {
            int temp = a[i];
86             j = i;
            while (j >= h && a[j - h] > temp) {
88                 a[j] = a[j - h];
                j -= h;
89             }
            a[j] = temp;
92         }
        h = h / 2;
94     }
    }
96
    void merge_sort(int a[], int l, int d)
98 {
        int s;
100     static int b[MAX];          /* Pomocni niz */
        int i, j, k;

102     /* Izlaz iz rekurzije */
104     if (l >= d)
        return;

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

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

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
120  /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
122  while (i <= s && j <= d) {
124      if (a[i] < a[j])
126          b[k++] = a[i++];
128      else
129          b[k++] = a[j++];
130  }
132  /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive j
133   iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
134   polovine niza */
135  while (i <= s)
136      b[k++] = a[i++];
137
138  /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive i
139   iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
140   polovine niza */
141  while (j <= d)
142      b[k++] = a[j++];
143
144  /* Prepisuje se "ucesljani" niz u originalni niz */
145  for (k = 0, i = 1; i <= d; i++, k++)
146      a[i] = b[k];
147
148  /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
149  void swap(int a[], int i, int j)
150  {
151      int tmp = a[i];
152      a[i] = a[j];
153      a[j] = tmp;
154  }
155
156  void quick_sort(int a[], int l, int d)
157  {
158      int i, pivot_pozicija;
159
160      /* Izlaz iz rekurzije -- prazan niz */
161      if (l >= d)
162          return;
163
164      /* Particionisanje niza. Svi elementi na pozicijama levo od
165       pivot_pozicija (izuzev same pozicije l) su strogo manji od
166       pivota. Kada se pronadje neki element manji od pivota, uvecava
167       se promenljiva pivot_pozicija i na tu poziciju se premesta
168       nadjeni element. Na kraju ce pivot_pozicija zaista biti pozicija
169       na koju treba smestiti pivot, jer ce svi elementi levo od te
170       pozicije biti manji a desno biti veci ili jednaki od pivota. */
171      pivot_pozicija = l;
172      for (i = l + 1; i <= d; i++)
173          if (a[i] < a[l])
```

```

172     swap(a, ++pivot_pozicija, i);

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

176

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

```

*main.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <time.h>
   #include "sort.h"
5
   /* Maksimalna duzina niza */
7  #define MAX 1000000

9  int main(int argc, char *argv[])
   {
11     /******
        tip_sortiranja == 0 => selectionsort, (podrazumevano)
13     tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
        tip_sortiranja == 2 => bubblesort,    -b opcija komandne linije
15     tip_sortiranja == 3 => shellsort,      -s opcija komandne linije
        tip_sortiranja == 4 => mergesort,     -m opcija komandne linije
17     tip_sortiranja == 5 => quicksort,      -q opcija komandne linije
        *****/
19     int tip_sortiranja = 0;
   /******
21     tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
        tip_niza == 1 => rastuce sortirani nizovi, -r opcija
23     tip_niza == 2 => opadajuće soritrani nizovi, -o opcija
        *****/
25     int tip_niza = 0;

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

31

   /* Provera argumenata komandne linije */
33     if (argc < 2) {
        fprintf(stderr,
35         "Program zahteva bar 2 argumenta komandne linije!\n");
        exit(EXIT_FAILURE);
37     }
}

```

```
39  /* Ocitavanje opcija i argumenata prilikom poziva programa */
    for (i = 1; i < argc; i++) {
41      /* Ako je u pitanju opcija... */
      if (argv[i][0] == '-') {
43          switch (argv[i][1]) {
              case 'i':
45                  tip_sortiranja = 1;
                      break;
47                  case 'b':
                      tip_sortiranja = 2;
49                      break;
                      case 's':
51                          tip_sortiranja = 3;
                              break;
53                          case 'm':
                              tip_sortiranja = 4;
55                              break;
                              case 'q':
57                                  tip_sortiranja = 5;
                                      break;
                                      case 'r':
59                                          tip_niza = 1;
                                              break;
                                              case 'o':
61                                                  tip_niza = 2;
63                                                  break;
                                                  default:
65                                                      printf("Pogresna opcija -%c\n", argv[i][1]);
67                                                      return 1;
69                                                      break;
69          }
      }
71      /* Ako je u pitanju argument, onda je to duzina niza koji treba
         da se sortira */
73      else {
          dimenzija = atoi(argv[i]);
75          if (dimenzija <= 0 || dimenzija > MAX) {
              fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
77              exit(EXIT_FAILURE);
          }
79      }
    }

81
83      /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
         niza dobijenom iz komandne linije. srand() funkcija obezbedjuje
         novi seed za pozivanje rand funkcije, i kako generisani niz ne
85         bi uvek bio isti seed je postavljen na tekuce vreme u sekundama
         od Nove godine 1970. rand()%100 daje brojeve izmedju 0 i 99 */
87      srand(time(NULL));
      if (tip_niza == 0)
89          for (i = 0; i < dimenzija; i++)
              niz[i] = rand();
```

```

91  else if (tip_niza == 1)
    for (i = 0; i < dimenzija; i++)
93      niz[i] = i == 0 ? rand() % 100 : niz[i - 1] + rand() % 100;
    else
95      for (i = 0; i < dimenzija; i++)
        niz[i] = i == 0 ? rand() % 100 : niz[i - 1] - rand() % 100;
97
    /* Ispisivanje elemenata niza */
99  /******
    Ovaj deo je iskomentaran jer sledeci ispis ne treba da se nadje
101  na standardnom izlazu. Njegova svrha je samo bila provera da li
    je niz generisan u skladu sa opcijama komandne linije.

    printf("Niz koji sortiramo je:\n");
105  for (i = 0; i < dimenzija; i++)
    printf("%d\n", niz[i]);
107  *****/

109
    /* Sortiranje niza na odgovarajuci nacin */
111  if (tip_sortiranja == 0)
    selection_sort(niz, dimenzija);
113  else if (tip_sortiranja == 1)
    insertion_sort(niz, dimenzija);
115  else if (tip_sortiranja == 2)
    bubble_sort(niz, dimenzija);
117  else if (tip_sortiranja == 3)
    shell_sort(niz, dimenzija);
119  else if (tip_sortiranja == 4)
    merge_sort(niz, 0, dimenzija - 1);
121  else
    quick_sort(niz, 0, dimenzija - 1);
123
    /* Ispis elemenata niza */
125  /******
    Ovaj deo je iskomentaran jer vreme potrebno za njegovo
127  izvršavanje ne bi trebalo da bude ukljuceno u vreme izmereno
    programom time. Takodje, kako je svrha ovog programa da prikaze
129  vremena razlicitih algoritama sortiranja, dimenzije nizova ce
    biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
131  od toliko elemenata. Ovaj deo je koriscen u razvoju programa
    zarad testiranja korektnosti.

    printf("Sortiran niz je:\n");
135  for (i = 0; i < dimenzija; i++)
    printf("%d\n", niz[i]);
137  *****/

139  exit(EXIT_SUCCESS);
}

```

#### Rešenje 3.13

```
1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAX_DIM 128
5
6  /* Funkcija za sortiranje niza karaktera */
7  void selectionSort(char s[])
8  {
9      int i, j, min;
10     char pom;
11     for (i = 0; s[i] != '\0'; i++) {
12         min = i;
13         for (j = i + 1; s[j] != '\0'; j++)
14             if (s[j] < s[min])
15                 min = j;
16         if (min != i) {
17             pom = s[i];
18             s[i] = s[min];
19             s[min] = pom;
20         }
21     }
22 }
23
24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
25 int anagrami(char s[], char t[])
26 {
27     int i;
28
29     /* Ako dve niske imaju razlicit broj karaktera onda one nisu
30        anagrami */
31     if (strlen(s) != strlen(t))
32         return 0;
33
34     /* Sortiramo niske */
35     selectionSort(s);
36     selectionSort(t);
37
38     /* Dve sortirane niske su anagrami ako i samo ako su jednake */
39     for (i = 0; s[i] != '\0'; i++)
40         if (s[i] != t[i])
41             return 0;
42     return 1;
43 }
44
45 int main()
46 {
47     char s[MAX_DIM], t[MAX_DIM];
48
49     /* Ucitavanje niski sa ulaza */
50     printf("Unesite prvu nisku: ");
```



```

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      min = a[1] - a[0];
       for (i = 2; i < n; i++)
13          if (a[i] - a[i - 1] < min)
              min = a[i] - a[i - 1];
15      return min;
   }
17
   int main()
19   {
       int i, a[MAX];

21
       /* Ucitavaju se elementi niza sve do kraja ulaza */
23      i = 0;
       while (scanf("%d", &a[i]) != EOF)
25          i++;

27      /* Za sortiranje niza moze se koristiti bilo koja od funkcija
       sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
29      se selection sort. */
       selection_sort(a, i);
31
       /* Ispis rezultata */
33      printf("%d\n", najmanje_rastojanje(a, i));

```

### 3 Algoritmi pretrage i sortiranja

---

```
35 | return 0;
    | }
```

#### Rešenje 3.15

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

```
1  #include <stdio.h>
   #include "sort.h"
3
   #define MAX_DIM 256
5
   /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
7     najvise puta pojavio u tom nizu */
   int najvise_puta(int a[], int n)
9   {
       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 */
       for (i = 0; i < n; i = j) {
13         br_pojava = 1;
           for (j = i + 1; j < n && a[i] == a[j]; j++)
15             br_pojava++;
           /* Ispitivanje da li se do tog trenutka i-ti element pojavio
17             najvise puta u nizu */
           if (br_pojava > max_br_pojava) {
19             max_br_pojava = br_pojava;
               i_max_pojava = i;
21         }
       }
23      /* Vraca se element koji se najvise puta pojavio u nizu */
       return a[i_max_pojava];
25   }

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

31       /* Ucitavanje elemenata niza sve do kraja ulaza */
       i = 0;
33       while (scanf("%d", &a[i]) != EOF)
           i++;
35
       /* Za sortiranje niza moze se koristiti bilo koja od funkcija
37         sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
           se merge sort. */
39       merge_sort(a, 0, i - 1);

41       /* Odredjuje se broj koji se najvise puta pojavio u nizu */
       printf("%d\n", najvise_puta(a, i));
43
       return 0;
```

---

45 }

### Rešenje 3.16

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

```

1  #include <stdio.h>
   #include "sort.h"
3
   #define MAX_DIM 256
5
   /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
   7     u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
       poretku */
9  int binarna_pretraga(int a[], int n, int x)
   {
11     int levi = 0, desni = n - 1, srednji;
13
       while (levi <= desni) {
           srednji = (levi + desni) / 2;
15             if (a[srednji] == x)
                   return 1;
17             else if (a[srednji] > x)
                   desni = srednji - 1;
19             else if (a[srednji] < x)
                   levi = srednji + 1;
21         }
       return 0;
23     }
25
   int main()
   {
27     int a[MAX_DIM], n = 0, zbir, i;
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
       /* Za sortiranje niza moze se koristiti bilo koja od funkcija
41         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++)

```

### 3 Algoritmi pretrage i sortiranja

```
47      /* Za i-ti element niza binarno se pretražuje da li se u 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 dim3 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 prepisujemo u treci */
24  while (i < dim1)
      niz3[k++] = niz1[i++];
26
   /* Ostatak drugog niza prepisujemo 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
64    return 0;
}

```

### Rešenje 3.18

```

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

4
int main(int argc, char *argv[])
6  {
    FILE *fin1 = NULL, *fin2 = NULL;
    FILE *fout = NULL;
    char ime1[11], ime2[11];
10   char prezime1[16], prezime2[16];
    int kraj1 = 0, kraj2 = 0;

12
    /* Ako nema dovoljno argumenata komandne linije */
14   if (argc < 3) {
        fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0])
        ;
16        exit(EXIT_FAILURE);
    }

18
    /* Otvaranje datoteke zadate prvim argumentom komandne linije */
20   fin1 = fopen(argv[1], "r");

```

### 3 Algoritmi pretrage i sortiranja

---

```
22     if (fin1 == NULL) {
23         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
24         exit(EXIT_FAILURE);
25     }
26
27     /* Otvaranje datoteke zadate drugim argumentom komandne linije */
28     fin2 = fopen(argv[2], "r");
29     if (fin2 == NULL) {
30         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
31         exit(EXIT_FAILURE);
32     }
33
34     /* Otvaranje datoteke za upis rezultata */
35     fout = fopen("ceo-tok.txt", "w");
36     if (fout == NULL) {
37         fprintf(stderr,
38             "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
39         exit(EXIT_FAILURE);
40     }
41
42     /* Citanje narednog studenta iz prve datoteke */
43     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
44         kraj1 = 1;
45
46     /* Citanje narednog studenta iz druge datoteke */
47     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
48         kraj2 = 1;
49
50     /* Sve dok nije dostignut kraj neke datoteke */
51     while (!kraj1 && !kraj2) {
52         int tmp = strcmp(ime1, ime2);
53         if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 0)) {
54             /* Ime i prezime iz prve datoteke je leksikografski ranije, i
55              biva upisano u izlaznu datoteku */
56             fprintf(fout, "%s %s\n", ime1, prezime1);
57             /* Citanje narednog studenta iz prve datoteke */
58             if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
59                 kraj1 = 1;
60         } else {
61             /* Ime i prezime iz druge datoteke je leksikografski ranije, i
62              biva upisano u izlaznu datoteku */
63             fprintf(fout, "%s %s\n", ime2, prezime2);
64             /* Citanje narednog studenta iz druge datoteke */
65             if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
66                 kraj2 = 1;
67         }
68     }
69
70     /* Ako se iz prethodne petlje izašlo zato što je dostignut kraj
71     druge datoteke, onda ima još studenata u prvoj datoteci, koje
72     treba prepisati u izlaznu, redom, jer su već sortirani po imenu.
73     */
```

```

74 while (!kraj1) {
    fprintf(fout, "%s %s\n", ime1, prezime1);
    if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
76     kraj1 = 1;
    }

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

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

92     exit(EXIT_SUCCESS);
94 }

```

### Rešenje 3.19

```

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

5

7  #define MAX_BR_TACAKA 128

   /* 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;

```

### 3 Algoritmi pretrage i sortiranja

---

```
27     for (i = 0; i < n - 1; i++) {
28         min = i;
29         for (j = i + 1; j < n; j++) {
30             if (rastojanje(t[j]) < rastojanje(t[min])) {
31                 min = j;
32             }
33         }
34         if (min != i) {
35             tmp = t[i];
36             t[i] = t[min];
37             t[min] = tmp;
38         }
39     }
40 }
41
42 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
43 void sortiraj_po_x(Tacka t[], int n)
44 {
45     int min, i, j;
46     Tacka tmp;
47
48     for (i = 0; i < n - 1; i++) {
49         min = i;
50         for (j = i + 1; j < n; j++) {
51             if (abs(t[j].x) < abs(t[min].x)) {
52                 min = j;
53             }
54         }
55         if (min != i) {
56             tmp = t[i];
57             t[i] = t[min];
58             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     /* Proveravanje broja argumenata komandne linije: ocekuje se ime
92        izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
93        datoteke, tj. 4 argumenta */
94     if (argc != 4) {
95         fprintf(stderr,
96                 "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
97         return 0;
98     }
99
100    /* Otvaranje datoteke u kojoj su zadate tacke */
101    ulaz = fopen(argv[2], "r");
102    if (ulaz == NULL) {
103        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
104                argv[2]);
105        return 0;
106    }
107
108    /* Otvaranje datoteke u koju treba upisati rezultat */
109    izlaz = fopen(argv[3], "w");
110    if (izlaz == NULL) {
111        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
112                argv[3]);
113        return 0;
114    }
115
116    /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
117       koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
118       brojacem i. */
119    i = 0;
120    while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
121        i++;
122    }
123
124    /* Ukupan broj procitanih tacaka */
125    n = i;
126
127    /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
128       "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
129       (karakter -), a karakter argv[1][1] odredjuje kriterijum
130       sortiranja */
131 }
```

### 3 Algoritmi pretrage i sortiranja

```
131  switch (argv[1][1]) {
132      case 'x':
133          /* Sortiranje po vrednosti x koordinate */
134          sortiraj_po_x(tacke, n);
135          break;
136      case 'y':
137          /* Sortiranje po vrednosti y koordinate */
138          sortiraj_po_y(tacke, n);
139          break;
140      case 'o':
141          /* Sortiranje po udaljenosti od koordinatnog pocetka */
142          sortiraj_po_rastojanju(tacke, n);
143          break;
144      }
145
146      /* Upisivanje dobijenog niza u izlaznu datoteku */
147      for (i = 0; i < n; i++) {
148          fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
149      }
150
151      /* Zatvaranje otvorenih datoteka */
152      fclose(ulaz);
153      fclose(izlaz);
154
155      return 0;
156  }
```

#### Rešenje 3.20

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  #define MAX 1000
6  #define MAX_DUZINA 16
7
8  /* Struktura koja reprezentuje jednog gradjanina */
9  typedef struct gr {
10      char ime[MAX_DUZINA];
11      char prezime[MAX_DUZINA];
12  } Gradjanin;
13
14  /* Funkcija sortira niz gradjana rastuce po imenima */
15  void sort_ime(Gradjanin a[], int n)
16  {
17      int i, j;
18      int min;
19      Gradjanin pom;
20
21      for (i = 0; i < n - 1; i++) {
22          /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
```

```

23     najmanji od elemenata a[i].ime,...,a[n-1].ime. */
    min = i;
25     for (j = i + 1; j < n; j++)
        if (strcmp(a[j].ime, a[min].ime) < 0)
27             min = j;
    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
29     su (i) i min razliciti, inace je nepotrebno. */
    if (min != i) {
31         pom = a[i];
        a[i] = a[min];
33         a[min] = pom;
    }
35 }
}

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

45     for (i = 0; i < n - 1; i++) {
        /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
47         najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
        min = i;
49         for (j = i + 1; j < n; j++)
            if (strcmp(a[j].prezime, a[min].prezime) < 0)
51                 min = j;
        /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
53         su (i) i min razliciti, inace je nepotrebno. */
        if (min != i) {
55             pom = a[i];
            a[i] = a[min];
57             a[min] = pom;
        }
59     }
}

61 /* Pretraga niza Gradjana */
63 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
{
65     int i;
    for (i = 0; i < n; i++)
67         if (strcmp(a[i].ime, x->ime) == 0
            && strcmp(a[i].prezime, x->prezime) == 0)
69             return i;
    return -1;
71 }

73 int main()

```

### 3 Algoritmi pretrage i sortiranja

```
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;
78     int i, n;
79     FILE *fp = NULL;

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

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

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

95     sort_ime(spisak1, n);

96     /******
101     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
102     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
103
104     printf("Biracki spisak [uredjen prema imenima]:\n");
105     for(i=0; i<n; i++)
106         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107     *****/

108     sort_prezime(spisak2, n);

109     /******
111     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
112     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
113
114     printf("Biracki spisak [uredjen prema prezimenima]:\n");
115     for(i=0; i<n; i++)
116         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
117     *****/

118     /* Linearno pretrazivanje nizova */
119     for (i = 0; i < n; i++)
120         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
121             isti_rbr++;

122     /* Alternativno (efikasnije) resenje */
123     /******
```

```

127     for(i=0; i<n ;i++)
        if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
129           strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
            isti_rbr++;
131     *****/

133     /* Ispis rezultata */
    printf("%d\n", isti_rbr);
135
    exit(EXIT_SUCCESS);
137 }

```

### Rešenje 3.22

```

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

5  #define MAX_BR_RECII 128
   #define MAX_DUZINA_RECII 32

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

29 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
    duzini reci se mora proslediti zbog pravilnog upravljanja
31     memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)
33 {
    int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35     duzina_j, duzina_min;
    char tmp[MAX_DUZINA_RECII];
37     for (i = 0; i < n - 1; i++) {

```

```

    min = i;
39  for (j = i; j < n; j++) {
    /* Prvo se upoređuje broj suglasnika */
41  broj_suglasnika_j = broj_suglasnika(reci[j]);
    broj_suglasnika_min = broj_suglasnika(reci[min]);
43  if (broj_suglasnika_j < broj_suglasnika_min)
        min = j;
45  else if (broj_suglasnika_j == broj_suglasnika_min) {
    /* Zatim, recima koje imaju isti broj suglasnika upoređuju
47  se duzine */
        duzina_j = strlen(reci[j]);
49  duzina_min = strlen(reci[min]);

        if (duzina_j < duzina_min)
            min = j;
51  else {
53  /* Ako reci imaju i isti broj suglasnika i iste duzine,
55  upoređuju se leksikografski */
        if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) <
0)
57  min = j;
    }
59  }
    }
61  if (min != i) {
        strcpy(tmp, reci[min]);
63  strcpy(reci[min], reci[i]);
        strcpy(reci[i], tmp);
65  }
    }
67 }

69 int main()
{
71  FILE *ulaz;
    int i = 0, n;

73
    /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
75  a drugi maksimalnu duzinu pojedinačne reci */
    char reci[MAX_BR_RECII][MAX_DUZINA_RECII];

77
    /* Otvaranje datoteke niske.txt za citanje */
79  ulaz = fopen("niske.txt", "r");
    if (ulaz == NULL) {
81  fprintf(stderr,
        "Greska prilikom otvaranja datoteke niske.txt!\n");
83  return 0;
    }

85
    /* Sve dok se moze procitati sledeca rec */
87  while (fscanf(ulaz, "%s", reci[i]) != EOF) {
        /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
```

```

89     prekida se učitavanje */
    if (i == MAX_BR_RECI)
91         break;
    /* Priprema brojava za narednu iteraciju */
93     i++;
}

95
/* n je dužina niza reci i predstavlja poslednju vrednost
97     koriscenog brojava */
n = i;
99     /* Poziv funkcije za sortiranje reci */
    sortiraj_reci(reci, n);
101
/* Ispis sortiranog niza reci */
103     for (i = 0; i < n; i++) {
        printf("%s ", reci[i]);
105     }
    printf("\n");
107
/* Zatvaranje datoteke */
109     fclose(ulaz);
111
return 0;
}

```

### Rešenje 3.23

```

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

4
#define MAX_ARTIKALA 100000

6
/* Struktura koja predstavlja jedan artikal */
8  typedef struct art {
    long kod;
10     char naziv[20];
    char proizvođač[20];
12     float cena;
} Artikal;

14
/* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
16     traženim bar kodom */
int binarna_pretraga(Artikal a[], int n, long x)
18 {
    int levi = 0;
20     int desni = n - 1;

22     /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
24         /* Racuna se sredisnji indeks */

```

```

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

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

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

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

70     FILE *fp = NULL;

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



```

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

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

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

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

```

### 3 Algoritmi pretrage i sortiranja

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

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

#### Rešenje 3.24

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

5 #define MAX 500

7 /* Struktura sa svim informacijama o pojedinacnom studentu */
typedef struct {
9     char ime[21];
    char prezime[26];
11     int prisustvo;
    int zadaci;
13 } Student;

15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
    rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
{
19     int i, j;
    int min;
21     Student pom;

23     for (i = 0; i < n - 1; i++) {
        min = i;
25         for (j = i + 1; j < n; j++)
```

```

27     if (strcmp(niz[j].ime, niz[min].ime) < 0)
        min = j;

29     if (min != i) {
        pom = niz[min];
31     niz[min] = niz[i];
        niz[i] = pom;
33     }
    }
35 }

37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
    zadataka opadajuće, a ukoliko neki studenti imaju isti broj
39 uradjenih zadataka sortiraju se po dužini imena rastuće. */
void sort_zadatke_pa_imena(Student niz[], int n)
41 {
    int i, j;
43     int max;
    Student pom;
45     for (i = 0; i < n - 1; i++) {
        max = i;
47         for (j = i + 1; j < n; j++)
            if (niz[j].zadaci > niz[max].zadaci)
49                 max = j;
            else if (niz[j].zadaci == niz[max].zadaci
51                     && strlen(niz[j].ime) < strlen(niz[max].ime))
                max = j;
53         if (max != i) {
            pom = niz[max];
55         niz[max] = niz[i];
            niz[i] = pom;
57         }
    }
59 }

61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
    su bili opadajuće. Ukoliko neki studenti imaju isti broj casova,
63 sortiraju se opadajuće po broju uradjenih zadataka, a ukoliko se
    i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65 prezimenu opadajuće. */
void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
67 {
    int i, j;
69     int max;
    Student pom;
71     for (i = 0; i < n - 1; i++) {
        max = i;
73         for (j = i + 1; j < n; j++)
            if (niz[j].prisustvo > niz[max].prisustvo)
75                 max = j;
            else if (niz[j].prisustvo == niz[max].prisustvo
77                     && niz[j].zadaci > niz[max].zadaci)

```

```

    max = j;
79     else if (niz[j].prisustvo == niz[max].prisustvo
               && niz[j].zadaci == niz[max].zadaci
81               && strcmp(niz[j].prezime, niz[max].prezime) > 0)
        max = j;
83     if (max != i) {
        pom = niz[max];
85         niz[max] = niz[i];
        niz[i] = pom;
87     }
    }
89 }

91 int main(int argc, char *argv[])
{
93     Student praktikum[MAX];
    int i, br_studenata = 0;
95
    FILE *fp = NULL;
97
    /* Otvaranje datoteke za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke aktivnost.txt.\n");
101        exit(EXIT_FAILURE);
    }
103
    /* Ucitavanje sadrzaja */
105     for (i = 0;
          fscanf(fp, "%s%s%d", praktikum[i].ime,
107                praktikum[i].prezime, &praktikum[i].prisustvo,
                &praktikum[i].zadaci) != EOF; i++);
109     /* Zatvaranje datoteke */
    fclose(fp);
111     br_studenata = i;

113     /* Kreiranje prvog spiska studenata po prvom kriterijumu */
    sort_ime_leksikografski(praktikum, br_studenata);
115     /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat1.txt", "w")) == NULL) {
117         fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
        exit(EXIT_FAILURE);
119     }
    /* Upis niza u datoteku */
121     fprintf
        (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
123     for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
125                praktikum[i].prezime, praktikum[i].prisustvo,
                praktikum[i].zadaci);
127     /* Zatvaranje datoteke */
    fclose(fp);
129 }
```

```

131  /* Kreiranje drugog spiska studenata po drugom kriterijumu */
    sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
133  if ((fp = fopen("dat2.txt", "w")) == NULL) {
        fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
135      exit(EXIT_FAILURE);
    }
137  /* Upis niza u datoteku */
    fprintf(fp, "Studenti sortirani po broju zadataka opadajuće,\n");
139    fprintf(fp, "pa po dužini imena rastuće:\n");
    for (i = 0; i < br_studenata; i++)
141        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
                praktikum[i].prezime, praktikum[i].prisustvo,
143                praktikum[i].zadaci);
    /* Zatvaranje datoteke */
145    fclose(fp);

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

165    exit(EXIT_SUCCESS);
}

```

### Rešenje 3.25

```

#include <stdio.h>
2  #include <stdlib.h>
#include <string.h>
4
#define KORAK 10
6
/* Struktura koja opisuje jednu pesmu */
8  typedef struct {
    char *izvodjac;
10    char *naslov;
    int broj_gledanja;

```

```
12 } Pesma;

14 /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
    rad qsort funkcije) */
16 int uporedi_gledanost(const void *pp1, const void *pp2)
17 {
18     Pesma *p1 = (Pesma *) pp1;
19     Pesma *p2 = (Pesma *) pp2;
20
21     return p2->broj_gledanja - p1->broj_gledanja;
22 }

24 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
    funkcije) */
26 int uporedi_naslove(const void *pp1, const void *pp2)
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 izvodjacu (potrebna za rad
    qsort funkcije) */
36 int uporedi_izvodjace(const void *pp1, const void *pp2)
37 {
38     Pesma *p1 = (Pesma *) pp1;
39     Pesma *p2 = (Pesma *) pp2;
40
41     return strcmp(p1->izvodjac, p2->izvodjac);
42 }

44 int main(int argc, char *argv[])
45 {
46     FILE *ulaz;
47     Pesma *pesme;                                /* Pokazivac na deo memorije za
48                                                    cuvanje pesama */
49
50     int alocirano_za_pesme;                        /* Broj mesta alociranih za pesme */
51     int i;                                         /* Redni broj pesme cije se
52                                                    informacije citaju */
53
54     int n;                                         /* Ukupan broj pesama */
55     int j, k;
56     char c;
57     int alocirano;                                /* Broj mesta alociranih za propratne
58                                                    informacije o pesmama */
59
60     int broj_gledanja;

61     /* Priprema datoteke za citanje */
62     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
63     if (ulaz == NULL) {
64         fprintf(stderr, "Greska pri otvaranju ulazne datoteke!\n");
65         return 0;
66     }
```

```
64 }
65
66 /* Citanje informacija o pesmama */
67 pesme = NULL;
68 alocirano_za_pesme = 0;
69 i = 0;
70
71 while (1) {
72
73     /* Proverava da li je dostignut kraj datoteke */
74     c = fgetc(ulaz);
75     if (c == EOF) {
76         /* Nema vise sadrzaja za citanje */
77         break;
78     } else {
79         /* Inace, vracamo procitani karakter nazad */
80         ungetc(c, ulaz);
81     }
82
83     /* Provera da li postoji dovoljno vec alocirane memorije za
84     citanje nove pesme */
85     if (alocirano_za_pesme == i) {
86         /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
87         novih KORAK mesta */
88         alocirano_za_pesme += KORAK;
89         pesme =
90             (Pesma *) realloc(pesme,
91                               alocirano_za_pesme * sizeof(Pesma));
92
93         /* Proverava da li je nova memorija uspesno realocirana */
94         if (pesme == NULL) {
95             /* Ako nije ispisuje se obavestenje */
96             fprintf(stderr, "Problem sa alokacijom memorije!\n");
97             /* I oslobadja sva memorija zauzeta do ovog koraka */
98             for (k = 0; k < i; k++) {
99                 free(pesme[k].izvodjac);
100                 free(pesme[k].naslov);
101             }
102             free(pesme);
103             return 0;
104         }
105     }
106
107     /* Ako jeste, nastavlja se sa citanjem pesama ... */
108     /* Cita se ime izvodjaca */
109     j = 0;
110     /* Pozicija na koju treba smestiti
111     procitani karakter */
112     alocirano = 0;
113     /* Broj alociranih mesta */
114     pesme[i].izvodjac = NULL;
115     /* Memorija za smestanje procitanih
116     karaktera */
117
118     /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
```

### 3 Algoritmi pretrage i sortiranja

---

```
116     izvodjaca) citaju se karakteri iz datoteke */
117 while ((c = fgetc(ulaz)) != ' ') {
118     /* Provera da li postoji dovoljno memorije za smestanje
119        procitanog karaktera */
120     if (j == alocirano) {
121
122         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
123            se novih KORAK mesta */
124         alocirano += KORAK;
125         pesme[i].izvodjac =
126             (char *) realloc(pesme[i].izvodjac,
127                             alocirano * sizeof(char));
128
129         /* Provera da li je nova alokacija uspesna */
130         if (pesme[i].izvodjac == NULL) {
131             /* Ako nije oslobadja se sva memorija zauzeta do ovog
132                koraka */
133             for (k = 0; k < i; k++) {
134                 free(pesme[k].izvodjac);
135                 free(pesme[k].naslov);
136             }
137             free(pesme);
138             /* I prekida sa izvrsavanjem programa */
139             return 0;
140         }
141     }
142     /* Ako postoji dovoljno alocirane memorije, smesta se vec
143        procitani karakter */
144     pesme[i].izvodjac[j] = c;
145     j++;
146     /* I nastavlja se sa citanjem */
147 }
148
149 /* Upis terminirajuće nule na kraj reci */
150 pesme[i].izvodjac[j] = '\0';
151
152 /* Preskace se karakter - */
153 fgetc(ulaz);
154
155 /* Preskace se razmak */
156 fgetc(ulaz);
157
158 /* Cita se naslov pesme */
159 j = 0; /* Pozicija na koju treba smestiti
160        procitani karakter */
161 alocirano = 0; /* Broj alociranih mesta */
162 pesme[i].naslov = NULL; /* Memorija za smestanje procitanih
163                          karaktera */
164
165 /* Sve do zarez (koji se nalazi nakon naslova pesme) citaju se
166    karakteri iz datoteke */
```



```
168 while ((c = fgetc(ulaz)) != ',') {
170     /* Provera da li postoji dovoljno memorije za smestanje
        procitanog karaktera */
172     if (j == alocirano) {
174         /* Ako ne, ako je potrošena sva alocirana memorija, alocira
            se novih KORAK mesta */
176         alocirano += KORAK;
            pesme[i].naslov =
                (char *) realloc(pesme[i].naslov,
                                alocirano * sizeof(char));

178
180         /* Provera da li je nova alokacija uspesna */
            if (pesme[i].naslov == NULL) {
182                 /* Ako nije, oslobadja se sva memorija zauzeta do ovog
                    koraka */
184                 for (k = 0; k < i; k++) {
186                     free(pesme[k].izvodjac);
188                     free(pesme[k].naslov);
190                 }
                    free(pesme[i].izvodjac);
                    free(pesme);

192                 /* I prekida izvršavanje programa */
                    return 0;
194             }
            }
196     /* Ako postoji dovoljno memorije, smesta se procitani karakter
        */
        pesme[i].naslov[j] = c;
        j++;
198     /* I nastavlja dalje sa citanjem */
    }
    /* Upisuje se terminirajuca nula na kraj reci */
    pesme[i].naslov[j] = '\0';

200
202    /* Preskace se razmak */
    fgetc(ulaz);

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

208    /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
        datoteke */
210    while ((c = fgetc(ulaz)) != '\n') {
        broj_gledanja = broj_gledanja * 10 + (c - '0');
212    }
    pesme[i].broj_gledanja = broj_gledanja;

214
216    /* Prelazi se na citanje sledece pesme */
    i++;
218 }
```

### 3 Algoritmi pretrage i sortiranja

```
220  /* Informacija o broju procitanih pesama */
    n = i;
222  /* Zatvaranje nepotrebne datoteke */
    fclose(ulaz);

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

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

250  /* Oslobadjanje memorije */
    for (i = 0; i < n; i++) {
252        free(pesme[i].izvodjac);
        free(pesme[i].naslov);
254    }
    free(pesme);
256
    return 0;
258 }
```

#### 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 koja odredjuje zbir v-te vrste matrice a koja ima m
```

```

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

```

```
58      /* Poziv funkcije koja sortira vrste matrice prema zbiru */
60      sortiraj_vrste(a, n, m);

62      /* Ispis rezultujuće matrice */
64      printf("Sortirana matrica je:\n");
        ispisi_matricu(a, n, m);

66      /* Oslobađanje memorije */
        a = dealociraj_matricu(a, n);

68
70      return 0;
    }
```

#### 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)
   {
11     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
        se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
13     int b1 = *((int *) a);
        int b2 = *((int *) b);

15
        /* Zbog moguceg prekoracenja opsega celih brojeva, oduzimanje b1-b2
17         treba izbegavati */
        if (b1 > b2)
19             return 1;
        else if (b1 < b2)
21             return -1;
        else
23             return 0;
   }

25
   /* Funkcija poredjenja dva cela broja (nerastuci poredak) */
27  int poredi_int_nerastuce(const void *a, const void *b)
   {
29     /* Za obrnuti poredak treba samo promeniti znak vrednosti koju koju
        vraca prethodna funkcija */
31     return -poredi_int(a, b);
   }

33
35  int main()
   {
```

```
size_t n;
37 int i, x;
int a[MAX], *p = NULL;
39
/* Unos dimenzije */
41 printf("Uneti dimenziju niza: ");
scanf("%ld", &n);
43 if (n > MAX)
    n = MAX;
45
/* Unos elementa niza */
47 printf("Uneti elemente niza:\n");
for (i = 0; i < n; i++)
49     scanf("%d", &a[i]);
51
/* Sortiranje niza celih brojeva */
qsort(a, n, sizeof(int), &poredi_int);
53
/* Prikaz sortiranog niz */
55 printf("Sortirani niz u rastucem poretku:\n");
for (i = 0; i < n; i++)
57     printf("%d ", a[i]);
putchar('\n');
59
/* Pretrazivanje niza */
/* Vrednost koja ce biti trazena u nizu */
61 printf("Uneti element koji se trazi u nizu: ");
scanf("%d", &x);
63
/* Binarna pretraga */
65 printf("Binarna pretraga: \n");
p = bsearch(&x, a, n, sizeof(int), &poredi_int);
67 if (p == NULL)
69     printf("Elementa nema u nizu!\n");
else
71     printf("Element je nadjen na poziciji %ld\n", p - a);
73
/* Linearna pretraga */
printf("Linearna pretraga (lfind): \n");
75 p = lfind(&x, a, &n, sizeof(int), &poredi_int);
if (p == NULL)
77     printf("Elementa nema u nizu!\n");
else
79     printf("Element je nadjen na poziciji %ld\n", p - a);
81 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 izašlo kada je
28        promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29        jos jednog delioca */
30     if (i * i == x)
31         br++;
32
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];
```

```

53  /* Unos dimenzije */
    printf("Uneti dimenziju niza: ");
55  scanf("%ld", &n);
    if (n > MAX)
57      n = MAX;

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

63

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

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

73  return 0;
}

```

### Rešenje 3.33

```

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

5

7  #define MAX_NISKI 1000
   #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 veka 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 upoređeni, (npr. pri porecenju prve i poslednje
   reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
   nad njima.

```

```
27  *****/
28  int poredi_leksikografski(const void *a, const void *b)
29  {
30      return strcmp((char *) a, (char *) b);
31  }
32
33  /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
34     leksikografski, vec po duzini */
35  int poredi_duzine(const void *a, const void *b)
36  {
37      return strlen((char *) a) - strlen((char *) b);
38  }
39
40  int main()
41  {
42      int i;
43      size_t n;
44      FILE *fp = NULL;
45      char niske[MAX_NISKI][MAX_DUZINA];
46      char *p = NULL;
47      char x[MAX_DUZINA];
48
49      /* Otvaranje datoteke */
50      if ((fp = fopen("niske.txt", "r")) == NULL) {
51          fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
52          exit(EXIT_FAILURE);
53      }
54
55      /* Citanje sadrzaja datoteke */
56      for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
57
58      /* Zatvaranje datoteke */
59      fclose(fp);
60      n = i;
61
62      /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
63         prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
64         niske po duzini */
65      qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);
66
67      printf("Leksikografski sortirane niske:\n");
68      for (i = 0; i < n; i++)
69          printf("%s ", niske[i]);
70      printf("\n");
71
72      /* Unos trazene niske */
73      printf("Uneti trazenu nisku: ");
74      scanf("%s", x);
75
76      /* Binarna pretraga */
77      /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
78         je niz vec sortiran leksikografski. */
```



```

79  p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
              &poredi_leksikografski);
81
82  if (p != NULL)
83      printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
              p, (p - (char *) niske) / MAX_DUZINA);
85  else
86      printf("Niska nije pronadjena u nizu\n");
87
88  /* Sortiranje po duzini */
89  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
91
92  printf("Niske sortirane po duzini:\n");
93  for (i = 0; i < n; i++)
94      printf("%s ", niske[i]);
95  printf("\n");
96
97  /* Linearna pretraga */
98  p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
            &poredi_leksikografski);
99
100 if (p != NULL)
101     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
            p, (p - (char *) niske) / MAX_DUZINA);
103 else
104     printf("Niska nije pronadjena u nizu\n");
105
106 exit(EXIT_SUCCESS);
107 }

```

### Rešenje 3.34

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <search.h>
5
6  #define MAX_NISKI 1000
7  #define MAX_DUZINA 31
8
9  /*****
10   Niz pokazivaca na karaktere ovog potpisa
11   char *niske[3];
12   posle alokacije u main-u se moze graficki predstaviti ovako:
13   -----
14   | X | -----> | a | b | c | \0|
15   -----
16   | Y | -----> | d | e | \0|
17   -----
18   | Z | -----> | f | g | h | \0|
19   -----

```

### 3 Algoritmi pretrage i sortiranja

```
21 Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
    njegov element pokazivac koji pokazuje na alocirane karaktere i-te
    reci. Njegov tip je char*.
23
25 Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
    koji trebaju biti uporedjeni (recimo adresu od X i adresu od Z), i
    kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
27 moraju i kastovati. Da bi se leksikografski uporedili elementi X i
    Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
29 u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
    kastovani na odgovarajuci tip.
31 *****/
    int poredi_leksikografski(const void *a, const void *b)
33 {
        return strcmp(*(char **) a, *(char **) b);
35 }
37 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
    leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
    {
41     return strlen(*(char **) a) - strlen(*(char **) b);
    }
43
45 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
    element u nizu sa kojim se poredi, pa njega treba kastovati na
    char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
47 ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
    main funkciji je to x, koji je tipa char*, tako da pokazivac a
49 ovde samo treba kastovati i ne dereferencirati. */
    int poredi_leksikografski_b(const void *a, const void *b)
51 {
        return strcmp((char *) a, *(char **) b);
53 }
55 int main()
    {
57     int i;
        size_t n;
59     FILE *fp = NULL;
        char *niske[MAX_NISKI];
61     char **p = NULL;
        char x[MAX_DUZINA];
63
        /* Otvaranje datoteke */
65     if ((fp = fopen("niske.txt", "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
67         exit(EXIT_FAILURE);
    }
69
        /* Citanje sadrzaja datoteke */
71     i = 0;
```

```

while (fscanf(fp, "%s", x) != EOF) {
73  /* Alociranje dovoljne memorije za i-tu nisku */
    if ((niske[i] = malloc((strlen(x) + 1) * sizeof(char))) == NULL)
    {
75      fprintf(stderr, "Greska pri alociranju niske\n");
      exit(EXIT_FAILURE);
77    }
    /* Kopiranje procitane niske na svoje mesto */
79    strcpy(niske[i], x);
    i++;
81  }

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

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

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

97  /* Unos trazene niske */
  printf("Uneti trazenu nisku: ");
99  scanf("%s", x);

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

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

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

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
123     printf("\n");
125     /* Oslobađanje zauzete memorije */
127     for (i = 0; i < n; i++)
128         free(niske[i]);
129     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 */
typedef struct {
10     char ime[21];
    char prezime[21];
12     int bodovi;
} Student;

14
/* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
16 istim brojem bodova se dodatno sortiraju leksikografski po
    prezimenu */
18 int poredi1(const void *a, const void *b)
{
20     Student *prvi = (Student *) a;
    Student *drugi = (Student *) b;

22
    if (prvi->bodovi > drugi->bodovi)
24         return -1;
    else if (prvi->bodovi < drugi->bodovi)
26         return 1;
    else
28         /* Ako su jednaki po broju bodova, treba ih uporediti po
            prezimenu */
30         return strcmp(prvi->prezime, drugi->prezime);
}

32
/* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
34 Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
    parametar je element niza ciji se bodovi porede. */
36 int poredi2(const void *a, const void *b)
{
38     int bodovi = *(int *) a;
    Student *s = (Student *) b;
40     return s->bodovi - bodovi;
}
```

```
42 }
43
44 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
45    Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
46    parametar je element niza cije se prezime poredi. */
47 int poredi3(const void *a, const void *b)
48 {
49     char *prezime = (char *) a;
50     Student *s = (Student *) b;
51     return strcmp(prezime, s->prezime);
52 }
53
54 int main(int argc, char *argv[])
55 {
56     Student kolokvijum[MAX];
57     int i;
58     size_t br_studenata = 0;
59     Student *nadjen = NULL;
60     FILE *fp = NULL;
61     int bodovi;
62     char prezime[21];
63
64     /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
65        informacija korisniku kako se program koristi i prekida se
66        izvorsavanje. */
67     if (argc < 2) {
68         fprintf(stderr,
69             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
70             argv[0]);
71         exit(EXIT_FAILURE);
72     }
73
74     /* Otvaranje datoteke */
75     if ((fp = fopen(argv[1], "r")) == NULL) {
76         fprintf(stderr, "Neupesno otvaranje datoteke %s\n", argv[1]);
77         exit(EXIT_FAILURE);
78     }
79
80     /* Ucitavanje sadrzaja */
81     for (i = 0;
82         fscanf(fp, "%s%s%d", kolokvijum[i].ime,
83             kolokvijum[i].prezime,
84             &kolokvijum[i].bodovi) != EOF; i++);
85
86     /* Zatvaranje datoteke */
87     fclose(fp);
88     br_studenata = i;
89
90     /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
91        studenata sa istim brojem bodova sortiranje vrsi po prezimenu */
92     qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
```

### 3 Algoritmi pretrage i sortiranja

```
printf("Studenti sortirani po broju poena opadajuće, ");
94 printf("pa po prezimenu rastuće:\n");
for (i = 0; i < br_studenata; i++)
96     printf("%s %s %d\n", kolokvijum[i].ime,
            kolokvijum[i].prezime, kolokvijum[i].bodovi);
98
/* Pretrazivanje studenata po broju bodova se vrši binarnom
100 pretragom jer je niz sortiran po broju bodova. */
printf("Unesite broj bodova: ");
102 scanf("%d", &bodovi);

nadjen =
104     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 vršiti linearno jer je niz
116 sortiran po bodovima. */
printf("Unesite prezime: ");
118 scanf("%s", prezime);

nadjen =
120     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;
11 }
12
13 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14 int anagrami(char s[], char t[])
15 {
16     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
17     if (strlen(s) != strlen(t))
18         return 0;
19
20     /* Sortiranje niski */
21     qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);
23
24     /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
25        suprotnom, nisu */
26     return !strcmp(s, t);
27 }
28
29 int main()
30 {
31     char s[MAX], t[MAX];
32
33     /* Unos niski */
34     printf("Unesite prvu nisku: ");
35     scanf("%s", s);
36     printf("Unesite drugu nisku: ");
37     scanf("%s", t);
38
39     /* Ispituje se da li su niske anagrami */
40     if (anagrami(s, t))
41         printf("jesu\n");
42     else
43         printf("nisu\n");
44
45     return 0;
46 }

```

### Rešenje 3.37

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  #define MAX 10
6  #define MAX_DUZINA 32
7
8  /* Funkcija porenjenja */
9  int uporedi_niske(const void *pa, const void *pb)
10 {

```

### 3 Algoritmi pretrage i sortiranja

```
11  return strcmp((char *) pa, (char *) pb);
12  }
13
14  int main()
15  {
16      int i, n;
17      char S[MAX][MAX_DUZINA];
18
19      /* Unos broja niski */
20      printf("Unesite broj niski:");
21      scanf("%d", &n);
22
23      /* Unos niza niski */
24      printf("Unesite niske:\n");
25      for (i = 0; i < n; i++)
26          scanf("%s", S[i]);
27
28      /* Sortiranje niza niski */
29      qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
30
31      /******
32       Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
33       sortiranih niski. Koriscen je samo u fazi testiranja programa.
34
35       printf("Sortirane niske su:\n");
36       for(i = 0; i < n; i++)
37           printf("%s ", S[i]);
38       *****/
39
40      /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
41       niza biti jedna do druge */
42      for (i = 0; i < n - 1; i++)
43          if (strcmp(S[i], S[i + 1]) == 0) {
44              printf("ima\n");
45              return 0;
46          }
47      printf("nema\n");
48
49      return 0;
50  }
```

#### Rešenje 3.38

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX 21
6
7  /* Struktura koja predstavlja jednog studenta */
8  typedef struct student {
```



```
9   char nalog[8];
10  char ime[MAX];
11  char prezime[MAX];
12  int poeni;
13 } Student;

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

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

49 /* Funkcija poredjenja po nalogu za upotrebu u biblioteckoj funkciji
50    bsearch */
51 int uporedi_bsearch(const void *a, const void *b)
52 {
53     /* Nalog studenta koji se trazi */
54     char *nalog = (char *) a;
55     /* Kljuc pretrage */
56     Student s = *(Student *) b;
57
58     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
59     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
60     char smer1 = nalog[1];
```

```
61     char smer2 = s.nalog[1];
62     int indeks1 =
63         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
64     ;
65     int indeks2 =
66         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
67         s.nalog[6] - '0';
68     if (godina1 != godina2)
69         return godina1 - godina2;
70     else if (smer1 != smer2)
71         return smer1 - smer2;
72     else
73         return indeks1 - indeks2;
74 }
75
76 int main(int argc, char **argv)
77 {
78     Student *nadjen = NULL;
79     char nalog_trazeni[8];
80     Student niz_studenata[100];
81     int i = 0, br_studenata = 0;
82     FILE *in = NULL, *out = NULL;
83
84     /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
85        korisnik nije ispravno pokrenuo program. */
86     if (argc != 2 && argc != 3) {
87         fprintf(stderr,
88             "Program se poziva sa: ./a.out -opcija [nalog]\n");
89         exit(EXIT_FAILURE);
90     }
91
92     /* Otvaranje datoteke za citanje */
93     in = fopen("studenti.txt", "r");
94     if (in == NULL) {
95         fprintf(stderr,
96             "Greska prilikom otvaranja datoteke studenti.txt!\n");
97         exit(EXIT_FAILURE);
98     }
99
100    /* Otvaranje datoteke za pisanje */
101    out = fopen("izlaz.txt", "w");
102    if (out == NULL) {
103        fprintf(stderr,
104            "Greska prilikom otvaranja datoteke izlaz.txt!\n");
105        exit(EXIT_FAILURE);
106    }
107
108    /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
109    while (fscanf
110        (in, "%s %s %s %d", niz_studenata[i].nalog,
111         niz_studenata[i].ime, niz_studenata[i].prezime,
112         &niz_studenata[i].poeni) != EOF)
```

```
113     i++;
114
115     br_studenata = i;
116
117     /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
118     if (strcmp(argv[1], "-p") == 0)
119         qsort(niz_studenata, br_studenata, sizeof(Student),
120             &uporedi_poeni);
121     /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
122     else if (strcmp(argv[1], "-n") == 0)
123         qsort(niz_studenata, br_studenata, sizeof(Student),
124             &uporedi_nalog);
125
126     /* Sortirani studenti se ispisuju u izlaznu datoteku */
127     for (i = 0; i < br_studenata; i++)
128         fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
129             niz_studenata[i].ime, niz_studenata[i].prezime,
130             niz_studenata[i].poeni);
131
132     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
133        studenta... */
134     if (argc == 3 && (strcmp(argv[1], "-n") == 0)){
135         strcpy(nalog_trazeni, argv[2]);
136
137         /* ... pronalazi se student sa tim nalogom. */
138         nadjen =
139             (Student *) bsearch(nalog_trazeni, niz_studenata,
140                 br_studenata, sizeof(Student),
141                 &uporedi_bsearch);
142
143         if (nadjen == NULL)
144             printf("Nije nadjen!\n");
145         else
146             printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
147                 nadjen->prezime, nadjen->poeni);
148     }
149
150     /* Zatvaranje datoteka */
151     fclose(in);
152     fclose(out);
153
154     exit(EXIT_SUCCESS);
155 }
```



## 4

# Dinamičke strukture podataka

## 4.1 Liste

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

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

- (g) Napisati funkciju `int dodaj_iza(Cvor * tekuci, int broj)` koja iza čvora `tekuci` dodaje novi čvor sa vrednošću `broj`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene zagradama `[, ]` i međusobno razdvojene zapetama.
- (j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.
- (k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.
- (l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.
- (n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. NAPOMENA: *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [3, 2]
14
Lista: [14, 3, 2]
5
Lista: [5, 14, 3, 2]
3
Lista: [3, 5, 14, 3, 2]
17
Lista: [17, 3, 5, 14, 3, 2]

Unosite broj koji se trazi: 5
Trazeni broj 5 je u listi!

```

*Primer 2*

```

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

Unosite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!

```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 14, 3]
3
Lista: [2, 3, 14, 3, 3]
17
Lista: [2, 3, 14, 3, 3, 17]
3
Lista: [2, 3, 14, 3, 3, 17, 3]

Unosite broj koji se brise: 3
Lista nakon brisanja: [2, 14, 17]

```

*Primer 2*

```

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

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

```

- (3) U programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 3, 14]
3
Lista: [2, 3, 3, 3, 14]
5
Lista: [2, 3, 3, 3, 5, 14]

Unesite broj koji se trazi: 14
Trazeni broj 14 je u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 5, 14]
```

### Primer 2

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

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

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

[Rešenje 4.1]

**Zadatak 4.2** Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekursivno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etiketne smestati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
{
    unsigned broj_pojavljivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;
```



*Test 1*

```

POKRETANJE: ./a.out datoteka.html

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

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

```

*Test 2*

```

POKRETANJE: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ ZA GREŠKU:
Greska prilikom otvaranja
datoteke datoteka.html.

```

[Rešenje 4.3]

**Zadatak 4.4** U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan po jedan, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku *da* ili *ne*, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

### Primer 1

```
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.

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

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

### Test 1

```
BROJEVI.TXT
43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
12 15 16
```

### Test 2

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

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

### Test 3

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ:
REZULTAT.TXT
Rezultat.txt ce biti prazna.
```

\* **Zadatak 4.6** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

*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, 4, 5, 6, 6, 10, 11, 12, 14, 15, 16]

```

*Test 2*

```

POKRETANJE: ./a.out dat1.txt dat2.txt

DAT1.TXT
  2 4 6 10 15

DATOTEKA DAT2.TXT NE POSTOJI.

IZLAZ ZA GREŠKU:
  Greska prilikom otvaranja datoteke
  dat2.txt.

```

*Test 3*

```

POKRETANJE: ./a.out dat1.txt dat2.txt

DATOTEKA DAT1.TXT JE PRAZNA

DAT2.TXT
  5 6 11 12 14 16

IZLAZ:
[5, 6, 11, 12, 14, 16]

```

*Test 4*

```

POKRETANJE: ./a.out dat1.txt

IZLAZ ZA GREŠKU:
  Program se poziva sa:
  ./a.out dat1.txt dat2.txt!

```

[Rešenje 4.6]

\* **Zadatak 4.7** Date su dve jednostruko povezane liste  $L1$  i  $L2$ . Napisati funkciju koja od ovih listi formira novu listu  $L$  koja sadrži naizmenično raspoređene čvorove listi  $L1$  i  $L2$ : prvi čvor iz  $L1$ , prvi čvor iz  $L2$ , drugi čvor  $L1$ , drugi čvor  $L2$ , itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *Iskoristiti testove 2 - 6 za zadatak 4.6.*

*Test 1*

```

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

## 4 Dinamičke strukture podataka

---

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

<p><i>Test 1</i></p> <pre>   IZRAZ.TXT    {[23 + 5344] * (24 - 234)} - 23       IZLAZ:      Zagrade su ispravno uparene.</pre>	<p><i>Test 2</i></p> <pre>   IZRAZ.TXT    {[23 + 5] * (9 * 2)} - {23}       IZLAZ:      Zagrade su ispravno uparene.</pre>
<p><i>Test 3</i></p> <pre>   IZRAZ.TXT    {[2 + 54] / (24 * 87)} + (234 + 23)       IZLAZ:      Zagrade nisu ispravno uparene.</pre>	<p><i>Test 4</i></p> <pre>   IZRAZ.TXT    {(2 - 14) / (23 + 11)} * (2 + 13)       IZLAZ:      Zagrade nisu ispravno uparene.</pre>
<p><i>Test 5</i></p> <pre>   DATOTEKA IZRAZ.TXT JE PRAZNA       IZLAZ:      Zagrade su ispravno uparene.</pre>	<p><i>Test 6</i></p> <pre>   DATOTEKA IZRAZ.TXT NE POSTOJI.       IZLAZ ZA GREŠKU:      Greska prilikom otvaranja      datoteke izraz.txt!</pre>

[Rešenje 4.8]

**Zadatak 4.9** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže HTML etikete.*

<p><i>Test 1</i></p> <pre>   POKRETANJE: ./a.out datoteka.html       DATOTEKA.HTML    &lt;html&gt;      &lt;head&gt;        &lt;title&gt;Primer&lt;/title&gt;      &lt;/head&gt;      &lt;body&gt;      &lt;/body&gt;       IZLAZ:      Etikete nisu pravilno uparene      (etiketa &lt;html&gt; nije zatvorena)</pre>	<p><i>Test 2</i></p> <pre>   POKRETANJE: ./a.out datoteka.html       DATOTEKA.HTML      &lt;head&gt;        &lt;title&gt;Primer&lt;/title&gt;      &lt;/head&gt;      &lt;body&gt;      &lt;/body&gt;    &lt;/html&gt;       IZLAZ:      Etikete nisu pravilno uparene      (nadjena je etiketa &lt;/html&gt;      koja nije otvorena)</pre>
--	--

## Test 3

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

DATOTEKA.HTML
<html>
<head>
  <title>Primer</title>
</head>
<body>
  <h1>Naslov</h1>
  Danas je lep i suncan dan. <br>
  Sutra ce biti jos lepsi.
  <a link='http://www.math.rs'>Link</a>
</body>
</html>

IZLAZ:
  Etikete su pravilno uparene!
```

## Test 4

```
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 GREŠKU:
  Greska prilikom otvaranja
  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 (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (EOF). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije. Program mu postavlja pitanje *Da li korisnika vracate na kraj reda?* i ukoliko on da odgovor *Da*, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije *Da*, službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke *izvestaj.txt*. Ova datoteka, za svaki obrađen zahtev, sadrži *JMBG* i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Sluzbenik evidentira korisnicke zahteve:
Novi zahtev [CTRL+D za kraj]
JMBG: 1234567890123
Opis problema: Otvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 2345678901234
Opis problema: Podizanje novca

Novi zahtev [CTRL+D za kraj]
JMBG: 3456789012345
Opis problema: Reklamacija

Novi zahtev [CTRL+D za kraj]
JMBG:

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 2345678901234
i zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

IZVESTAJ.TXT
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 3456789012345 Zahtev: Reklamacija
JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

[Rešenje 4.10]

**Zadatak 4.11** Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

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

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

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

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

[Rešenje 4.11]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 5 3  IZLAZ: 3 1 5 2 4 </pre>	<pre> ULAZ: 8 4  IZLAZ: 4 8 5 2 1 3 7 6 </pre>	<pre> ULAZ: 3 8  IZLAZ ZA GREŠKU: n mora biti uvek vece od k, a 3 &lt; 8! </pre>

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

a levu na rame plesača sa brojem 1 i tako zatvara krug. Plesači izvode svoju plesnu tačku tako što iz formiranog kruga najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smeru. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojivati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

Test 1	Test 2	Test 3
ULAZ: 5 3	ULAZ: 8 4	ULAZ: 5 8
IZLAZ: 3 5 4 2 1	IZLAZ: 4 8 5 7 6 3 2 1	IZLAZ ZA GREŠKU: 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.
- 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
Rezultuje stablo: 1 2 9 18 32
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultuje stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

**Zadatak 4.15** Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova.

## 4 Dinamičke strukture podataka

---

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

### Test 1

```
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

INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

### Test 4

```
POKRETANJE: ./a.out

IZLAZ:
Nedostaje ime ulazne datoteke!
```

[Rešenje 4.15]

**Zadatak 4.16** U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. **Pera Peric 064/123-4567**. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči **KRAJ**, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

*Primer 1*

```
IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858

INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik.txt
Unesite ime i prezime: Pera Peric
Broj je: 011/3240-987
Unesite ime i prezime: Marko Markovic
Broj nije u imeniku!
Unesite ime i prezime: KRAJ
```

*Primer 2*

```
DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik1.txt
Greska: Neuspesno otvaranje datoteke
imenik1.txt.
```

[Rešenje 4.16]

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

*Test 1*

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

IZLAZ:
1. Maja Agic 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9
```

*Test 2*

```
DATOTEKA PRIJEMNI.TXT NE POSTOJI

IZLAZ:
Greska: Neuspesno otvaranje datoteke
prijemni.txt.
```

[Rešenje 4.17]

\* **Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije. Svaka linija datoteke je formata *Ime Prezime DD.MM.* i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan 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 u formatu DD.MM.. 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
```

```
INTERAKCIJA SA PROGRAMOM:
```

```
Greska: Neuspesno otvaranje datoteke
rodjendani.txt.
```

[Rešenje 4.18]

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

*Primer 1*

```

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

```

*Primer 2*

```

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

```

[Rešenje 4.19]

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

*Primer 1*

```

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

```

*Primer 2*

```

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

```

[Rešenje 4.20]

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

*Primer 1*

```

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

```

*Primer 2*

```

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

```

[Rešenje 4.21]

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

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

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

### Test 1

```
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 glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

### Test 1

```

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

### Test 2

```

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

[Rešenje 4.25]

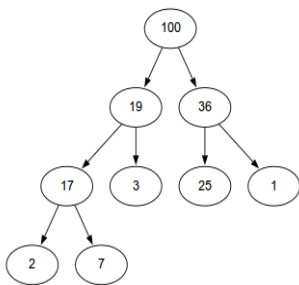
**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int heap(Cvor * koren)` koja proverava da li je dato binarno stablo celih brojeva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

### Test 1

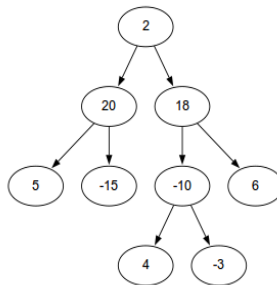
```

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

[Rešenje 4.26]



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.



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

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

*Test 1*

```

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

```

## 4.3 Rešenja

### Rešenje 4.1

*lista.h*

```

1  #ifndef _LISTA_H_
2  #define _LISTA_H_

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

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

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);

20 /* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo

```

```

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

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

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

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

36 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
38 int dodaj_iza(Cvor * tekuci, int broj);

40 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
42 inace vraca 0. */
    int dodaj_sortirano(Cvor ** adresa_glave, int broj);
44

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

50 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na činjenicu da je lista koja se pretražuje
52 neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
    */
54 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 slučaju da se
58 obrise stara glava. */
    void obrisi_cvor(Cvor ** adresa_glave, int broj);
60

62 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na činjenicu da je prosledjena lista sortirana
    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
64 promenjen ukoliko se obrise stara glava liste. */
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
66

68 /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
    liste, razdvojene zarezima i uokvirene zagradama. */
    void ispisi_listu(Cvor * glava);
70

#endif

```

*lista.c*

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

5  Cvor *napravi_cvor(int broj)
6  {
7      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8         alokacije */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;

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

15     /* Vraca se adresa novog cvora */
16     return novi;
17 }

18 void oslobodi_listu(Cvor ** adresa_glave)
19 {
20     Cvor *pomocni = NULL;

21     /* Ako lista nije prazna, onda treba osloboditi memoriju */
22     while (*adresa_glave != NULL) {
23         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
24            osloboditi cvor koji predstavlja glavu liste */
25         pomocni = (*adresa_glave)->sledeci;
26         free(*adresa_glave);

27         /* Sledeci cvor je nova glava liste */
28         *adresa_glave = pomocni;
29     }
30 }

31 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
32 {
33     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
34     Cvor *novi = napravi_cvor(broj);
35     if (novi == NULL)
36         return 1;

37     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
38     novi->sledeci = *adresa_glave;
39     *adresa_glave = novi;

40     /* Vraca se indikator uspesnog dodavanja */
41     return 0;
42 }
```

```

}
51
Cvor *pronadji_poslednji(Cvor * glava)
53 {
    /* U praznoj listi nema cvorova pa se vraca NULL */
55     if (glava == NULL)
        return NULL;
57
    /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
59     glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
        ce pokazivati na cvor liste koji nema sledbenika, tj. na
61     poslednji cvor liste pa se vraca vrednost pokazivaca glava.
        Pokazivac glava je argument funkcije i njegove promene nece se
63     odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
    while (glava->sledeci != NULL)
65         glava = glava->sledeci;
67
    return glava;
}
69
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
71 {
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
73     Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
75         return 1;
77
    /* Ako je lista prazna */
    if (*adresa_glave == NULL) {
79        /* Glava nove liste je upravo novi cvor */
        *adresa_glave = novi;
81    } else {
        /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
83        se dodaje na kraj liste kao sledbenik poslednjeg */
        Cvor *poslednji = pronadji_poslednji(*adresa_glave);
85        poslednji->sledeci = novi;
    }
87
    /* Vraca se indikator uspesnog dodavanja */
89    return 0;
}
91
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
93 {
    /* U praznoj listi nema takvog mesta i vraca se NULL */
95     if (glava == NULL)
        return NULL;
97
    /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
99     pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
        vecu ili jednaku vrednosti novog cvora.
101     Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti

```

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

```

155     return dodaj_iza(pomocni, broj);
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, pretraga
162            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 provera 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 veca 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 je
183        NULL ili tako sto je glava->vrednost == broj. U oba poslednja
184        nacina treba vratiti pokazivac glava bilo da je NULL ili
185        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;

```

```
205 }

207 /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
    funkcije */
209 if (*adresa_glave == NULL)
    return;
211
213 /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
    pokazuje na cvor cija je vrednost razlicita od trazenog broja.
    Isto vazi i za sve cvorove levo od tekućeg. Poredi se vrednost
215 sledećeg cvora (ako postoji) sa trazenim brojem. Cvor se brise
    ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
    Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
217 */
    tekuci = *adresa_glave;
219 while (tekuci->sledeci != NULL)
    if (tekuci->sledeci->vrednost == broj) {
221         /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
            prvo cuva u promenljivoj pomocni. */
223         pomocni = tekuci->sledeci;
            /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
225 njegovog trenutnog sledećeg. Njegov novi sledeci ce biti
            sledeci od cvora koji se brise. */
227         tekuci->sledeci = pomocni->sledeci;
            /* Sada treba osloboditi cvor sa vrednoscu broj. */
229         free(pomocni);
    } else {
231         /* Inace, ne treba brisati sledećeg od tekućeg i pokazivac se
            pomera na sledeci. */
233         tekuci = tekuci->sledeci;
    }
235 return;
}

237
239 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
    Cvor *tekuci = NULL;
241     Cvor *pomocni = NULL;

243     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
        broju i azurira se pokazivac na glavu liste. */
245     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
        /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
247         adresi adresa_glave. */
        pomocni = (*adresa_glave)->sledeci;
249         free(*adresa_glave);
        *adresa_glave = pomocni;
251     }

253     /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
        prekida. Isto se radi i ukoliko glava liste sadrzi vrednost koja
```

## 4 Dinamičke strukture podataka

```
255     je veca od broja, jer kako je lista sortirana rastuce nema
      potrebe broj traziti u repu liste. */
257     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;
259
      /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
261         na cvor cija vrednost je manja od trazenog broja, kao i svim
        cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
263         je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
        ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
265         cija vrednost je veca od trazenog broja. */
      tekuci = *adresa_glave;
267     while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj)
        {
          if (tekuci->sledeci->vrednost == broj) {
269             pomocni = tekuci->sledeci;
            tekuci->sledeci = tekuci->sledeci->sledeci;
271             free(pomocni);
          } else {
273             /* Ne treba brisati sledeceg od tekuceg jer je manji od
              trazenog i tekuci se pomera na sledeci cvor. */
275             tekuci = tekuci->sledeci;
          }
277     return;
  }
279
  void ispisi_listu(Cvor * glava)
281  {
      /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
283         jer se lista nece menjati */
      putchar('[');
285     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
        pocetka prema kraju liste. */
287     for (; glava != NULL; glava = glava->sledeci) {
        printf("%d", glava->vrednost);
289         if (glava->sledeci != NULL)
            printf(", ");
291     }
      printf("]\n");
293  }
```

*main\_a.c*

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

5  /* 1) Glavni program */
   int main()
7  {
      /* Lista je prazna na pocetku */
```



```

9   Cvor *glava = NULL;
   Cvor *trazeni = NULL;
11  int broj;

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

27  /* Testiranje funkcije za pretragu liste */
   printf("\nUnesite broj koji se trazi: ");
29  scanf("%d", &broj);

31  trazeni = pretrazi_listu(glava, broj);
33  if (trazeni == NULL)
   printf("Broj %d se ne nalazi u listi!\n", broj);
35  else
   printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
37

   /* Oslobadja se memorija zauzeta listom */
39  oslobodi_listu(&glava);

41  exit(EXIT_SUCCESS);
   }

```

main\_b.c

```

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

4

   /* 2) Glavni program */
6  int main()
   {
8     /* Lista je prazna na pocetku */
   Cvor *glava = NULL;
10    int broj;

12    /* Testira se funkcija za dodavanja novog broja na kraj liste */
   printf("Unesite brojeve: (za kraj CTRL+D)\n");
14    while (scanf("%d", &broj) > 0) {

```

## 4 Dinamičke strukture podataka

```
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_na_kraj_liste(&glava, broj) == 1) {
20         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
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 iz liste cije je polje vrednost jednako broju
33    procitanom sa ulaza */
34 obrisi_cvor(&glava, broj);
35
36 printf("Lista nakon brisanja: ");
37 ispisi_listu(glava);
38
39 /* Oslobadja se memorija zauzeta listom */
40 oslobodi_listu(&glava);
41
42 exit(EXIT_SUCCESS);
43 }
```

main.c.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  /* 3) Glavni program */
6  int main()
7  {
8      /* Lista je prazna na pocetku */
9      Cvor *glava = NULL;
10     Cvor *trazeni = NULL;
11     int broj;
12
13     /* Testira se funkcija za dodavanje vrednosti u listu tako da bude
14        uredjena neopadajuće */
15     printf("Unosite brojeve (za kraj CTRL+D)\n");
16     while (scanf("%d", &broj) > 0) {
17         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
18            memorije za nov cvor. Memoriju alociranu za cvorove liste
19            treba osloboditi pre napustanja programa. */
20         if (dodaj_sortirano(&glava, broj) == 1) {
```

```

    fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
22     oslobodi_listu(&glava);
    exit(EXIT_FAILURE);
24 }
    printf("\tLista: ");
26     ispisi_listu(glava);
}

28
/* Testira se funkcija za pretragu liste */
30 printf("\nUnesite broj koji se trazi: ");
scanf("%d", &broj);

32
trazeni = pretrazi_listu(glava, broj);
34 if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
36 else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
38

/* Testira se funkcija kojom se brise cvor liste */
40 printf("\nUnesite broj koji se brise: ");
scanf("%d", &broj);

42
/* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
44     procitanom sa ulaza */
obrisi_cvor_sortirane_liste(&glava, broj);

46
printf("Lista nakon brisanja: ");
48     ispisi_listu(glava);

50
/* Oslobadja se memorija zauzeta listom */
oslobodi_listu(&glava);

52
exit(EXIT_SUCCESS);
54 }

```

## Rešenje 4.2

*lista.h*

```

1  #ifndef _LISTA_H_
2  #define _LISTA_H_

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

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,

```

```
12     dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
14 Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
    ciji se pokazivac glava nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);

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

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

28 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
    ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
    memorije, inace vraca 0. */
30 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
32
34 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
    NULL u slučaju da takav cvor ne postoji u listi. */
36 Cvor *pretrazi_listu(Cvor * glava, int broj);

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

42 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivac na glavu liste, koji može biti promenjen u slučaju da se
    obrise stara glava liste. */
44 void obrisi_cvor(Cvor ** adresa_glave, int broj);
46
48 /* 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. */
50 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
52
54 /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
    zaptetama. */
56 void ispisi_vrednosti(Cvor * glava);
58
60 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
    liste, razdvojene zaptetama i uokvirene zagradama. */
62 void ispisi_listu(Cvor * glava);
```

```
#endif
```

lista.c

```

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

5  Cvor *napravi_cvor(int broj)
6  {
7      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8         alokacije */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;

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

15     /* Vraca se adresa novog cvora */
16     return novi;
17 }

19 void oslobodi_listu(Cvor ** adresa_glave)
20 {
21     /* Ako je lista vec prazna */
22     if (*adresa_glave == NULL)
23         return;

24     /* Ako lista nije prazna, treba osloboditi memoriju. Treba
25        osloboditi rep liste, pre oslobadjanja memorije za glavu liste
26        */
27     oslobodi_listu(&(*adresa_glave)->sledeci);
28     /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
29        glava u pozivajucoj funkciji tako da odgovara praznoj listi */
30     free(*adresa_glave);
31     *adresa_glave = NULL;
32 }

34 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
35 {
36     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
37     Cvor *novi = napravi_cvor(broj);
38     if (novi == NULL)
39         return 1;

40     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
41     novi->sledeci = *adresa_glave;
42     *adresa_glave = novi;

```

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

51  int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
    {
53      /* Ako je lista prazna */
      if (*adresa_glave == NULL) {
55
57          /* Novi cvor postaje glava liste */
          Cvor *novi = napravi_cvor(broj);
          /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
          */
59          if (novi == NULL)
              return 1;
61
63          /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
          azurira i pokazivacka promenljiva u pozivajucoj funkciji */
          *adresa_glave = novi;
65
67          /* Vraca se indikator uspesnog dodavanja */
          return 0;
        }

69
71      /* Ako lista nije prazna, broj se dodaje u rep liste. */
      /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
      novi broj se dodaje u rep liste u rekurzivnom pozivu.
73      Informaciju o uspesnosti alokacije u rekurzivnom pozivu funkcija
      prosledjuje visem rekurzivnom pozivu koji tu informaciju vraca u
75      rekurzivni poziv iznad, sve dok se ne vrati u main. Tek je iz
      main funkcije moguće pristupiti pravom pocetku liste i
77      osloboditi je celu, ako ima potrebe. Ako je funkcija vratila 0,
      onda nije bilo greske. */
79      return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
    }

81
83  int dodaj_sortirano(Cvor ** adresa_glave, int broj)
    {
85      /* Ako je lista prazna */
      if (*adresa_glave == NULL) {
87
89          /* Novi cvor postaje glava liste */
          Cvor *novi = napravi_cvor(broj);

          /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
          */
91          if (novi == NULL)
              return 1;
93
95          /* Azurira se glava liste */
          *adresa_glave = novi;
```

```
97      /* Vraca se indikator uspesnog dodavanja */
98      return 0;
99  }

101  /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
102     glavi liste, onda se dodaje na pocetak liste */
103  if ((*adresa_glave)->vrednost >= broj)
104      return dodaj_na_pocetak_liste(adresa_glave, broj);
105
106  /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
107     bude sortirana lista. */
108  return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
109  }

111  Cvor *pretrazi_listu(Cvor * glava, int broj)
112  {
113      /* U praznoj listi nema vrednosti */
114      if (glava == NULL)
115          return NULL;
116
117      /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
118      if (glava->vrednost == broj)
119          return glava;
120
121      /* Inace, pretraga se nastavlja u repu liste */
122      return pretrazi_listu(glava->sledeci, broj);
123  }

125  Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
126  {
127      /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
128         vrednosti u glavi liste, jer je lista neopadajuće sortirana */
129      if (glava == NULL || glava->vrednost > broj)
130          return NULL;
131
132      /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
133      if (glava->vrednost == broj)
134          return glava;
135
136      /* Inace, pretraga se nastavlja u repu liste */
137      return pretrazi_listu(glava->sledeci, broj);
138  }

139  void obrisi_cvor(Cvor ** adresa_glave, int broj)
140  {
141      /* U praznoj listi nema cvorova za brisanje. */
142      if (*adresa_glave == NULL)
143          return;
144
145      /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
146      obrisi_cvor(&(*adresa_glave)->sledeci, broj);
```

```
149  /* Preostaje provera da li glavu liste treba obrisati */
    if ((*adresa_glave)->vrednost == broj) {
151      /* Pomocni pokazuje na cvor koji treba da se obriše */
      Cvor *pomocni = *adresa_glave;
153      /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
         briše se cvor koji je bio glava liste. */
155      *adresa_glave = (*adresa_glave)->sledeci;
      free(pomocni);
157    }
  }

159  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
161  {
    /* Ako je lista prazna ili glava liste sadrži vrednost koja je veća
163     od broja, kako je lista sortirana rastuće nema potrebe broj
     traziti u repu liste i zato se funkcija prekida */
165     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;

167     /* Brisu se cvorovi iz repa koji imaju vrednost broj */
169     obrisi_cvor(&(*adresa_glave)->sledeci, broj);

171     /* Preostaje provera da li glavu liste treba obrisati */
    if ((*adresa_glave)->vrednost == broj) {
173      /* Pomocni pokazuje na cvor koji treba da se obriše */
      Cvor *pomocni = *adresa_glave;
175      /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
         briše se cvor koji je bio glava liste */
177      *adresa_glave = (*adresa_glave)->sledeci;
      free(pomocni);
179    }
  }

181  void ispisi_vrednosti(Cvor * glava)
183  {
    /* Prazna lista */
185     if (glava == NULL)
        return;

187     /* Ispisuje se vrednost u glavi liste */
189     printf("%d", glava->vrednost);

191     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
        se poziva ista funkcija za ispis ostalih. */
193     if (glava->sledeci != NULL) {
        printf(", ");
195         ispisi_vrednosti(glava->sledeci);
    }
197  }

199  void ispisi_listu(Cvor * glava)
    {
```



```

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

```

### Rešenje 4.3

```

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

5  #define MAX_DUZINA 20

7  /* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
   broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
9  typedef struct _Cvor {
       char etiketa[20];
11     unsigned broj_pojavljivanja;
       struct _Cvor *sledeci;
13 } Cvor;

15 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
   ili NULL ako alokacija nije uspesno izvršena */
17 Cvor *napravi_cvor(unsigned br, char *etiketa)
   {
19     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
       alokacije */
21     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
23         return NULL;

25     /* Inicijalizacija polja strukture */
       novi->broj_pojavljivanja = br;
27     strcpy(novi->etiketa, etiketa);
       novi->sledeci = NULL;

29     /* Vraca se adresa novog cvora */
31     return novi;
   }

33

35 /* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
   void oslobodi_listu(Cvor ** adresa_glave)
   {
37     Cvor *pomocni = NULL;

```

```
39  /* Sve dok lista ni bude prazna, brise se tekuca glava liste i
    azurira se vrednost glave liste */
41  while (*adresa_glave != NULL) {
    pomocni = (*adresa_glave)->sledeci;
43  free(*adresa_glave);
    *adresa_glave = pomocni;
45  }
    /* Pokazivac glava u main funkciji, na adresi adresa_glave, bice
    postavljen na NULL vrednost po izlasku iz petlje. */
47  }
49
    /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
    do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
51  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
    char *etiketa)
53  {
55  /* Kreira se novi cvor i proverava se uspesnost alokacije */
    Cvor *novi = napravi_cvor(br, etiketa);
57  if (novi == NULL)
    return 1;
59
    /* Dodaje se novi cvor na pocetak liste */
61  novi->sledeci = *adresa_glave;
    *adresa_glave = novi;
63
    /* Vraca se indikator uspesnog dodavanja */
65  return 0;
    }
67
    /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
    NULL ako takav cvor ne postoji u listi. */
69  Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
71  {
    Cvor *tekuci;
73
    /* Obilazi se cvor po cvor liste */
75  for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
    /* Ako tekuci cvor sadrzi trazenu etiketu, vraca se njegova
    vrednost */
77  if (strcmp(tekuci->etiketa, etiketa) == 0)
79  return tekuci;
81
    /* Cvor nije pronadjen */
    return NULL;
83  }
85
    /* Funkcija ispisuje sadrzaj liste */
    void ispisi_listu(Cvor * glava)
87  {
    /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
    broj njihovog pojavljivanja u HTML datoteci. */
89  for (; glava != NULL; glava = glava->sledeci)
```

```
91     printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
92 }
93
94 /* Glavni program */
95 int main(int argc, char **argv)
96 {
97     /* Provera se da li je program pozvan sa ispravnim brojem
98        argumenata. */
99     if (argc != 2) {
100         fprintf(stderr,
101             "Greska! Program se poziva sa: ./a.out datoteka.html!\n");
102         ;
103         exit(EXIT_FAILURE);
104     }
105
106     /* Priprema datoteke za citanje */
107     FILE *in = NULL;
108     in = fopen(argv[1], "r");
109     if (in == NULL) {
110         fprintf(stderr,
111             "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
112         exit(EXIT_FAILURE);
113     }
114
115     char c;
116     int i = 0;
117     char procitana[MAX_DUZINA];
118     Cvor *glava = NULL;
119     Cvor *trazeni = NULL;
120
121     /* Cita se karakter po karakter datoteke sve dok se ne procita cela
122        datoteka */
123     while ((c = fgetc(in)) != EOF) {
124
125         /* Proverava se da li se pocinje sa citanjem nove etikete */
126         if (c == '<') {
127             /* Proverava se da li se cita zatvarajuca etiketa */
128             if ((c = fgetc(in)) == '/') {
129                 i = 0;
130                 while ((c = fgetc(in)) != '>')
131                     procitana[i++] = c;
132             }
133             /* Cita se otvarajuca etiketa */
134             else {
135                 i = 0;
136                 procitana[i++] = c;
137                 while ((c = fgetc(in)) != ' ' && c != '>')
138                     procitana[i++] = c;
139             }
140             procitana[i] = '\0';
141
142             /* Trazi se procitana etiketa medju postojećim cvorovima liste.
```

```
        Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
sa
143     brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
        etikete */
145     trazeni = pretrazi_listu(glava, procitana);
        if (trazeni == NULL) {
147         if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
            fprintf(stderr, "Neuspela alokacija za nov cvor\n");
149             oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
151         }
        } else
153         trazeni->broj_pojavljivanja++;
    }
155 }

157 /* Zatvaranje datoteke */
    fclose(in);

159 /* Ispisuje se sadrzaj cvorova liste */
    ispisi_listu(glava);

163 /* Oslobadja se memorija zauzeta listom */
    oslobodi_listu(&glava);

165     exit(EXIT_SUCCESS);
167 }
```

### Rešenje 4.4

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

4
    #define MAX_INDEKS 11
6    #define MAX_IME_PREZIME 21

8    /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
        studentu */
10    typedef struct _Cvor {
        char broj_indeksa[MAX_INDEKS];
12        char ime[MAX_IME_PREZIME];
        char prezime[MAX_IME_PREZIME];
14        struct _Cvor *sledeci;
    } Cvor;

16
    /* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
18     novi cvor ili NULL ukoliko je doslo do greske */
    Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20    {
        /* Alocira se memorija za novi cvor liste i proverava se uspesnost
```

```
22     alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
        return NULL;

26     /* Inicijalizacija polja strukture */
    strcpy(novi->broj_indeksa, broj_indeksa);
    strcpy(novi->ime, ime);
30     strcpy(novi->prezime, prezime);
    novi->sledeci = NULL;

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

36 /* Funkcija oslobadja memoriju zauzetu cvorovima liste */
38 void oslobodi_listu(Cvor ** adresa_glave)
{
40     /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
    if (*adresa_glave == NULL)
42         return;

44     /* Rekurzivnim pozivom se oslobadja rep liste */
    oslobodi_listu(&(*adresa_glave)->sledeci);

46     /* Potom se oslobadja i glava liste */
48     free(*adresa_glave);

50     /* Proglasava se lista praznom */
    *adresa_glave = NULL;
52 }

54 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
    do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
56 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
                             char *ime, char *prezime)
58 {
    /* Kreira se novi cvor i proverava se uspesnost alokacije */
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
    if (novi == NULL)
62         return 1;

64     /* Dodaje se novi cvor na pocetak liste */
    novi->sledeci = *adresa_glave;
66     *adresa_glave = novi;

68     /* Vraca se indikator uspesnog dodavanja */
    return 0;
70 }

72 /* Funkcija ispisuje sadrzaj cvorova liste. */
void ispisi_listu(Cvor * glava)
```

```

74 {
75     /* Pocevsi od glave liste */
76     for (; glava != NULL; glava = glava->sledeci)
77         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
78             glava->prezime);
79 }
80
81 /* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj indeksa.
82    U suprotnom funkcija vraca NULL */
83 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
84 {
85     /* Ako je lista prazna, ne postoji trazeni cvor */
86     if (glava == NULL)
87         return NULL;
88
89     /* Poredi se trazeni broj indeksa sa brojem indeksa u glavi liste
90        */
91     if (!strcmp(glava->broj_indeksa, broj_indeksa))
92         return glava;
93
94     /* Ukoliko u glavi liste nije trazeni indeks, pretraga se nastavlja
95        u repu liste */
96     return pretrazi_listu(glava->sledeci, broj_indeksa);
97 }
98
99 /* Glavni program */
100 int main(int argc, char **argv)
101 {
102     /* Argumenti komandne linije su neophodni jer se iz komandne linije
103        dobija ime datoteke sa informacijama o studentima */
104     if (argc != 2) {
105         fprintf(stderr,
106             "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
107         exit(EXIT_FAILURE);
108     }
109
110     /* Priprema datoteke za citanje */
111     FILE *in = NULL;
112     in = fopen(argv[1], "r");
113     if (in == NULL) {
114         fprintf(stderr,
115             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
116         exit(EXIT_FAILURE);
117     }
118
119     /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
120        listu */
121     char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
122     char broj_indeksa[MAX_INDEKS];
123     Cvor *glava = NULL;
124     Cvor *trazeni = NULL;

```

```

126  /* Ucitavanje vrednosti u listu */
127  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
128      if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
129          fprintf(stderr, "Neuspela alokacija za nov cvor\n");
130          oslobodi_listu(&glava);
131          exit(EXIT_FAILURE);
132      }
133
134  /* Datoteka vise nije potrebna i zatvara se. */
135  fclose(in);
136
137  /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
138  while (scanf("%s", broj_indeksa) != EOF) {
139      trazeni = pretrazi_listu(glava, broj_indeksa);
140      if (trazeni == NULL)
141          printf("ne\n");
142      else
143          printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
144  }
145
146  /* Oslobadja se memorija zauzeta za cvorove liste. */
147  oslobodi_listu(&glava);
148
149  exit(EXIT_SUCCESS);
150 }

```

**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"
4
5  /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
6     na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7     pokazivaca postojećih cvorova listi */
8  Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9  {
10     /* Pokazivaci na pocetne cvorove listi koje se prevezuju */
11     Cvor *lista1 = *adresa_glave_1;
12     Cvor *lista2 = *adresa_glave_2;
13
14     /* Pokazivac na pocetni cvor rezultujuće liste */
15     Cvor *rezultujuca = NULL;
16     Cvor *tekuci = NULL;
17
18     /* Ako su obe liste prazne i rezultat je prazna lista */
19     if (lista1 == NULL && lista2 == NULL)
20         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 rezultujuce liste - to je ili prvi cvor
    liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
    koji sadrzi manju vrednost */
33  if (lista1->vrednost < lista2->vrednost) {
        rezultujuca = lista1;
35        lista1 = lista1->sledeci;
    } else {
37        rezultujuca = lista2;
        lista2 = lista2->sledeci;
39    }
    /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste, ne
    sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
    koji sadrzi adresu trenutnog cvora rezultujuce liste */
43    tekuci = rezultujuca;

45    /* U svakoj iteraciji petlje rezultujucoj listi se dodaje novi cvor
    tako da bude uredjena neopadajuce. Pokazivac na listu iz koje se
    uzima cvor se azurira tako da pokazuje na sledeci cvor */
47    while (lista1 != NULL && lista2 != NULL) {
49        if (lista1->vrednost < lista2->vrednost) {
            tekuci->sledeci = lista1;
51            lista1 = lista1->sledeci;
        } else {
53            tekuci->sledeci = lista2;
            lista2 = lista2->sledeci;
55        }
        tekuci = tekuci->sledeci;
57    }

59    /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
    rezultujucu listu treba nadovezati ostatak druge liste */
61    if (lista1 == NULL)
        tekuci->sledeci = lista2;
63    else
        /* U suprotnom treba nadovezati ostatak prve liste */
65        tekuci->sledeci = lista1;

67    /* Preko adresa glava polaznih listi vrednosti pokazivaca u
    pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
69    prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
    se moze doci prateci pokazivace iz glave rezultujuce liste, tako
    da stare pokazivace treba postaviti na NULL. */
71    *adresa_glave_1 = NULL;
73    *adresa_glave_2 = NULL;

```



```
75     return rezultujuca;
76 }
77
78 int main(int argc, char **argv)
79 {
80     /* Argumenti komandne linije su neophodni */
81     if (argc != 3) {
82         fprintf(stderr,
83             "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
84         exit(EXIT_FAILURE);
85     }
86
87     /* Otvaranje datoteka u kojima se nalaze elementi listi */
88     FILE *in1 = NULL;
89     in1 = fopen(argv[1], "r");
90     if (in1 == NULL) {
91         fprintf(stderr, "Greska pri
92     }likom otvaranja datoteke %s.\n", argv[1]);
93         exit(EXIT_FAILURE);
94
95     FILE *in2 = NULL;
96     in2 = fopen(argv[2], "r");
97     if (in2 == NULL) {
98         fprintf(stderr,
99             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
100         exit(EXIT_FAILURE);
101     }
102
103     /* Liste su na pocetku prazne */
104     int broj;
105     Cvor *lista1 = NULL;
106     Cvor *lista2 = NULL;
107
108     /* Ucitavanje listi */
109     while (fscanf(in1, "%d", &broj) != EOF)
110         dodaj_na_kraj_liste(&lista1, broj);
111
112     while (fscanf(in2, "%d", &broj) != EOF)
113         dodaj_na_kraj_liste(&lista2, broj);
114
115     /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
116     fclose(in1);
117     fclose(in2);
118
119     /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
120         objedinjavanjem listi */
121     Cvor *rezultat = objedini(&lista1, &lista2);
122
123     /* Ispis rezultujuce liste. */
124     ispisi_listu(rezultat);
125 }
```

```
127     /* Lista rezultat dobijena je prevezivanjem cvorova polaznih
        listi. Njenim oslobadjanjem bice oslobodjena sva zauzeta
        memorija. */
129     oslobodi_listu(&rezultat);
131     exit(EXIT_SUCCESS);
}
```

### Rešenje 4.8

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
   5     predstavlja vidjenu zagradu i pokazivac na sledeci cvor liste */
   typedef struct cvor {
7       char zagrada;
       struct cvor *sledeci;
9   } Cvor;

11  /* Funkcija koja oslobadja memoriju zauzetu stekom */
   void oslobodi_stek(Cvor ** stek)
13  {
       Cvor *tekuci;
15     Cvor *pomocni;

17     /* Oslobadja se cvor po cvor steka */
       tekuci = *stek;
19     while (tekuci != NULL) {
         pomocni = tekuci->sledeci;
21         free(tekuci);
         tekuci = pomocni;
23     }

25     /* Stek se proglašava praznim */
       *stek = NULL;
27 }

29 /* Glavni program */
   int main()
31 {
       /* Stek je na pocetku prazan */
33     Cvor *stek = NULL;
       FILE *ulaz = NULL;
35     char c;
       Cvor *pomocni = NULL;
37

       /* Otvaranje datotoke za citanje izraza */
39     ulaz = fopen("izraz.txt", "r");
       if (ulaz == NULL) {
41         fprintf(stderr,
```

```

43         "Greska prilikom otvaranja datoteke izraz.txt!\n");
    exit(EXIT_FAILURE);
}

45
/* Cita se karakter po karakter iz datoteke dok se ne dodje do
47 kraja */
while ((c = fgetc(ulaz)) != EOF) {
49     /* Ako je učitana otvorena zagrada, stavlja se na stek */
    if (c == '(' || c == '{' || c == '[') {
51         /* Alocira se memorija za novi cvor liste i proverava se
           uspesnost alokacije */
53         pomocni = (Cvor *) malloc(sizeof(Cvor));
        if (pomocni == NULL) {
55             fprintf(stderr, "Greska prilikom alokacije memorije!\n");
            /* Oslobadja se memorija zauzeta stekom */
57             oslobodi_stek(&stek);
            /* I prekida se sa izvršavanjem programa */
59             exit(EXIT_FAILURE);
        }

61
        /* Inicijalizacija polja strukture */
63         pomocni->zagrada = c;

65
        /* Promena vrha steka */
        pomocni->sledeci = stek;
67         stek = pomocni;
    }

69     /* Ako je učitana zatvorena zagrada, proverava se da li je stek
       prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
71 otvorena zagrada */
    else {
73         if (c == ')' || c == '}' || c == ']') {
            if (stek != NULL && ((stek->zagrada == '(' && c == ')')
75                || (stek->zagrada == '{' && c == '}')
                || (stek->zagrada == '[' && c == ']')))
            {
77                 /* Sa vrha steka se uklanja otvorena zagrada */
                pomocni = stek->sledeci;
79                 free(stek);
                stek = pomocni;
            } else {
81                 /* Dakle zagrade u izrazu nisu ispravno uparene */
                break;
83             }
        }
85     }
}

87

89 /* Pročitana je cela datoteka i treba je zatvoriti */
fclose(ulaz);

91
/* Ako je stek prazan i pročitana je cela datoteka, zagrade su

```

```
93     ispravno uparene */
94     if (stek == NULL && c == EOF)
95         printf("Zagrade su ispravno uparene.\n");
96     else {
97         /* U suprotnom se zaključuje da zagrade nisu ispravno uparene */
98         printf("Zagrade nisu ispravno uparene.\n");
99         /* Oslobadja se memorija koja je ostala zauzeta stekom */
100        oslobodi_stek(&stek);
101    }
102
103    exit(EXIT_SUCCESS);
104 }
```

### 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 vraća njegovu
26    adresu ili NULL ako alokacija nije bila uspesna */
27 Cvor *napravi_cvor(char *etiketa);
28
29 /* Funkcija oslobadja memoriju zauzetu stekom */
30 void oslobodi_stek(Cvor ** adresa_vrha);
31
32 /* Funkcija postavlja na vrh steka novu etiketu. U slučaju greske pri
33    alokaciji memorije za novi cvor funkcija vraća 1, inace vraća 0 */
```

```

34 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);

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

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
/* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
52   etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
   Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
54   procita etiketa. Vraca OTVORENA, ako je procitana otvorena
   etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
56 int uzmi_etiketu(FILE * f, char *etiketa);

58 #endif

```

stek.c

```

#include "stek.h"

2
Cvor *napravi_cvor(char *etiketa)
4 {
   /* Alocira se memorija za novi cvor liste i proverava se uspesnost
   alokacije */
6   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   if (novi == NULL)
8       return NULL;

   /* Inicijalizacija polja u novom cvoru */
12  if (strlen(etiketa) >= MAX) {
      fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
14      etiketa[MAX - 1] = '\0';
  }

16  strcpy(novi->etiketa, etiketa);
  novi->sledeci = NULL;

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

22
void oslobodi_stek(Cvor ** adresa_vrha)

```

```
24 {
25     Cvor *pomocni;
26
27     /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
28     while (*adresa_vrha != NULL) {
29         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
30            osloboditi cvor koji predstavlja vrh steka */
31         pomocni = *adresa_vrha;
32         /* Sledeci cvor je novi vrh steka */
33         *adresa_vrha = (*adresa_vrha)->sledeci;
34         free(pomocni);
35     }
36
37     /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
38        adresa_vrha ce pokazivati na NULL. */
39 }
40
41 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
42 {
43     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
44     Cvor *novi = napravi_cvor(etiketa);
45     if (novi == NULL)
46         return 1;
47
48     /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
49     novi->sledeci = *adresa_vrha;
50     *adresa_vrha = novi;
51     return 0;
52 }
53
54 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
55 {
56     Cvor *pomocni;
57
58     /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom i
59        vraca se 0 */
60     if (*adresa_vrha == NULL)
61         return 0;
62
63     /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
64        adresu kopira etiketa sa vrha steka */
65     if (etiketa != NULL)
66         strcpy(etiketa, (*adresa_vrha)->etiketa);
67
68     /* Element sa vrha steka se uklanja */
69     pomocni = *adresa_vrha;
70     *adresa_vrha = (*adresa_vrha)->sledeci;
71     free(pomocni);
72
73     /* Vraca se indikator uspesno izvorsene radnje */
74     return 1;
75 }
```

```

76 char *vrh_steka(Cvor * vrh)
77 {
78     /* Prazan stek nema cvor koji je vrh i vraca se NULL */
80     if (vrh == NULL)
81         return NULL;
82
83     /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
84        koji je na vrhu steka. */
85     return vrh->etiketa;
86 }
87
88 void prikazi_stek(Cvor * vrh)
89 {
90     /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
91     for (; vrh != NULL; vrh = vrh->sledeci)
92         printf("<#s>\n", vrh->etiketa);
93 }
94
95 int uzmi_etiketu(FILE * f, char *etiketa)
96 {
97     int c;
98     int i = 0;
99     /* Stanje predstavlja informaciju dokle se stalo sa citanjem
100        etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
101        nije zapoceto citanje. */
102     /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
103        OTVORENA ili ZATVORENA. */
104     int stanje = VAN_ETIKETE;
105     int tip;
106
107     /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
108        to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
109        samo malim slovima. */
110     while ((c = fgetc(f)) != EOF) {
111         switch (stanje) {
112             case VAN_ETIKETE:
113                 if (c == '<')
114                     stanje = PROCITANO_MANJE;
115                 break;
116             case PROCITANO_MANJE:
117                 if (c == '/') {
118                     /* Cita se zatvorena etiketa */
119                     tip = ZATVORENA;
120                 } else {
121                     if (isalpha(c)) {
122                         /* Cita se otvorena etiketa */
123                         tip = OTVORENA;
124                         etiketa[i++] = tolower(c);
125                     }
126                 }
127             /* Od sada se cita etiketa i zato se menja stanje */

```

## 4 Dinamičke strukture podataka

```
128     stanje = U_ETIKETI;
129     break;
130 case U_ETIKETI:
131     if (isalpha(c) && i < MAX - 1) {
132         /* Ako je procitani karakter slovo i nije prekoracena
133            dozvoljena duzina etikete, procitani karakter se smanjuje
134            i smesta u etiketu */
135         etiketa[i++] = tolower(c);
136     } else {
137         /* Inace, staje se sa citanjem etikete. Korektno se zavrшава
138            niska koja sadrzi procitanu etiketu i vraca se njen tip */
139         etiketa[i] = '\0';
140         return tip;
141     }
142     break;
143 }
144 }
145 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
146    etiketa i vraca se EOF. */
147 return EOF;
148 }
```

*main.c*

```
#include "stek.h"

2 /* Glavni program */
3 int main(int argc, char **argv)
4 {
5     /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
6        nije procitana. */
7     Cvor *vrh = NULL;
8     char etiketa[MAX];
9     int tip;
10    int uparene = 1;
11    FILE *f = NULL;
12
13    /* Ime datoteke se preuzima iz komandne linije. */
14    if (argc < 2) {
15        fprintf(stderr, "Korisenje: %s ime_html_datoteke\n", argv[0]);
16        exit(EXIT_FAILURE);
17    }
18
19    /* Datoteka se otvara za citanje */
20    if ((f = fopen(argv[1], "r")) == NULL) {
21        fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
22                argv[1]);
23        exit(EXIT_FAILURE);
24    }
25
26    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
```



```

28 while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
    /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
30   etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
    potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
32   koje nemaju sadrzaj (npr link). Zbog jednostavnosti
    pretpostavlja se da njih nema u HTML dokumentu. */
34   if (tip == OTVORENA) {
       if (strcmp(etiketa, "br") != 0
36         && strcmp(etiketa, "hr") != 0
         && strcmp(etiketa, "meta") != 0)
38       if (potisni_na_stek(&vrh, etiketa) == 1) {
           fprintf(stderr, "Neuspela alokacija za nov cvor\n");
40           oslobodi_stek(&vrh);
           exit(EXIT_FAILURE);
42       }
   }
44   /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
    u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
46   uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
    je taj uslov ispunjen, skida se sa steka, jer je upravo
48   zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
    nisu pravilno uparene. */
50   else if (tip == ZATVORENA) {
       if (vrh_steka(vrh) != NULL
52         && strcmp(vrh_steka(vrh), etiketa) == 0)
           skini_sa_steka(&vrh, NULL);
54       else {
           printf("Etikete nisu pravilno uparene\n");
56           printf("(nadjena je etiketa </%s>", etiketa);
           if (vrh_steka(vrh) != NULL)
58               printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh))
           ;
           else
60               printf(" koja nije otvorena)\n");
           uparene = 0;
62           break;
       }
54   }
64 }
66 /* Zavrseno je citanje i datoteka se zatvara */
fclose(f);

68
70 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
    da bude prazan. Ukoliko nije, tada postoje etikete koje su
    ostale otvorene */
72 if (uparene) {
    if (vrh_steka(vrh) == NULL)
74         printf("Etikete su pravilno uparene!\n");
    else {
76         printf("Etikete nisu pravilno uparene\n");
         printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
78         /* Oslobadja se memorija zauzeta stekom */
    }
}

```

```
        oslobodi_stek(&vrh);
80    }
    }
82    exit(EXIT_SUCCESS);
84 }
```

### 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 i
11    opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
18    korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
20     Zahtev nalog;
21     struct cvor *sledeci;
22 } Cvor;
23
24 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
25    poslate adrese i vraća adresu novog cvora ili NULL ako je doslo do
26    greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
27    treba smestiti u novi cvor zbog smestanja manjeg podatka na
28    sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
29    bajtovima(B) u odnosu na strukturu Zahtev. */
30 Cvor *napravi_cvor(Zahtev * zahtev);
31
32 /* Funkcija prazni red oslobadjajuci memoriju koji je red zauzimao */
33 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
34
35 /* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
36    greske pri alokaciji memorije za novi cvor, inace vraća 0. */
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
38                Zahtev * zahtev);
39
```

```

41  /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
    pokazivac razlicit od NULL, tada se u strukturu na koju on
43  pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
    suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
    suprotnom. */
45  int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
    Zahtev * zahtev);
47
    /* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev korisnika
49    na pocetku reda. Ukoliko je red prazan funkcija vraca NULL. */
    Zahtev *pocetak_reda(Cvor * pocetak);
51
    /* Funkcija prikazuje sadrzaj reda. */
53  void prikazi_red(Cvor * pocetak);
55  #endif

```

red.c

```

1  #include "red.h"
3  Cvor *napravi_cvor(Zahtev * zahtev)
    {
5      /* Alocira se memorija za novi cvor liste i proverava uspesnost
        alokacije */
7      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
        if (novi == NULL)
9          return NULL;

11     /* Inicijalizacija polja strukture */
        novi->nalog = *zahtev;
13     novi->sledeci = NULL;

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

19 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
    {
21     Cvor *pomocni = NULL;

23     /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
        while (*pocetak != NULL) {
25         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
            osloboditi cvor sa pocetka reda */
27         pomocni = *pocetak;
            *pocetak = (*pocetak)->sledeci;
29         free(pomocni);
        }
31     /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
        treba postaviti na NULL. */

```

```
33     *kraj = NULL;
34 }
35
36 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
37               Zahtev * zahtev)
38 {
39     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
40     Cvor *novi = napravi_cvor(zahtev);
41     if (novi == NULL)
42         return 1;
43
44     /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
45        kraj, to je podjednako efikasno kao dodavanje na pocetak liste
46        */
47     if (*adresa_kraja != NULL) {
48         (*adresa_kraja)->sledeci = novi;
49         *adresa_kraja = novi;
50     } else {
51         /* Ako je red bio ranije prazan */
52         *adresa_pocetka = novi;
53         *adresa_kraja = novi;
54     }
55
56     /* Vraca se indikator uspesnog dodavanja */
57     return 0;
58 }
59
60 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
61                 Zahtev * zahtev)
62 {
63     Cvor *pomocni = NULL;
64
65     /* Ako je red prazan */
66     if (*adresa_pocetka == NULL)
67         return 0;
68
69     /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
70        zahtev koji je na pocetku reda. */
71     if (zahtev != NULL)
72         *zahtev = (*adresa_pocetka)->nalog;
73
74     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac
75        na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor
76        u redu. */
77     pomocni = *adresa_pocetka;
78     *adresa_pocetka = (*adresa_pocetka)->sledeci;
79     free(pomocni);
80
81     /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,
82        potrebno je azurirati i vrednost pokazivaca na adresi
83        adresa_kraja na NULL */
84     if (*adresa_pocetka == NULL)
```

```

    *adresa_kraja = NULL;
85
    return 1;
87 }

89 Zahtev *pocetak_reda(Cvor * pocetak)
{
91     /* U praznom redu nema zahteva */
    if (pocetak == NULL)
93         return NULL;

95     /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
    return &(pocetak->nalog);
97 }

99 void prikazi_red(Cvor * pocetak)
{
101     /* Prikazuje se sadrzaj reda od pocetka prema kraju */
    for (; pocetak != NULL; pocetak = pocetak->sledeci)
103         printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);

105     printf("\n");
}

```

*main.c*

```

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

6 #define VREME_ZA_PAUZU 5

8 /* Glavni program */
int main(int argc, char **argv)
10 {
    /* Red je prazan. */
12     Cvor *pocetak = NULL, *kraj = NULL;
    Zahtev nov_zahtev;
14     Zahtev *sledeci = NULL;
    char odgovor[3];
16     int broj_usluzenih = 0;

18     /* Sluzbenik evidentira korisnicke zahteve unosjenjem njihovog JMBG
        broja i opisa potrebne usluge. */
    printf("Sluzbenik evidentira korisnicke zahteve:\n");
20     while (1) {

22         /* Ucitava se JMBG */
        printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
24         if (scanf("%s", nov_zahtev.jmbg) == EOF)

```

```
26         break;

28     /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
        JMBG broja procitao i da bi fgets nakon toga procitala
30     ispravan red sa opisom zahteva */
    getchar();

32

33     /* Ucitava se opis problema */
34     printf("\tOpis problema: ");
    fgets(nov_zahtev.opis, MAX - 1, stdin);
36     /* Ako je poslednji karakter nov red, eliminiše se */
    if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
38         nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';

40     /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
    if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
42         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_red(&pocetak, &kraj);
44         exit(EXIT_FAILURE);
    }
46 }

48 /* Otvaranje datoteke za dopisivanje izvestaja */
FILE *izlaz = fopen("izvestaj.txt", "a");
50 if (izlaz == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
52     exit(EXIT_FAILURE);
}

54

55 /* Dokle god ima korisnika u redu, treba ih usluziti */
56 while (1) {
    sledeci = pocetak_reda(pocetak);
58     /* Ako nema nikog vise u redu, prekida se petlja */
    if (sledeci == NULL)
60         break;

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

64

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

66     broj_usluzenih++;

68

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

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

```

78     if (broj_usluzenih == VREME_ZA_PAUZU) {
80         printf("\nDa li je kraj smene? [Da/Ne] ");
81         scanf("%s", odgovor);
82
83         if (strcmp(odgovor, "Da") == 0)
84             break;
85         else
86             broj_usluzenih = 0;
87     }
88 }
89
90 /*****
91  Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:
92
93  while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
94      printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
95          nov_zahtev.jmbg);
96      printf("sa zahtevom: %s\n", nov_zahtev.opis);
97      broj_usluzenih++;
98
99      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
100     scanf("%s", odgovor);
101     if (strcmp(odgovor, "Da") == 0)
102         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
103     else
104         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
105             nov_zahtev.jmbg, nov_zahtev.opis);
106
107     if (broj_usluzenih == VREME_ZA_PAUZU) {
108         printf("\nDa li je kraj smene? [Da/Ne] ");
109         scanf("%s", odgovor);
110         if (strcmp(odgovor, "Da") == 0)
111             break;
112         else
113             broj_usluzenih = 0;
114     }
115 }
116
117 *****/
118 /* Datoteka vise nije potrebna i treba je zatvoriti. */
119 fclose(izlaz);
120
121 /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
122    neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
123    koju zauzima red sa neobradjenim zahtevima korisnika. */
124 oslobodi_red(&pocetak, &kraj);
125
126 exit(EXIT_SUCCESS);
127 }

```

## Rešenje 4.11

## 4 Dinamičke strukture podataka

---

*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
5     vrednost i pokazivace na sledeci i prethodni cvor liste. */
6  typedef struct cvor {
7      int vrednost;
8      struct cvor *sledeci;
9      struct cvor *prethodni;
10 } Cvor;
11
12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
13    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
15 Cvor *napravi_cvor(int broj);
16
17 /* Funkcija oslobadja dinamiciku memoriju zauzetu za cvorove liste
18    ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na
19    adresi adresa_kraja. */
20 void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);
21
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
27 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
28    pri alokaciji memorije, inace vraca 0. */
29 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
30                          int broj);
31
32 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
33    novi cvor sa vrednoscu broj. */
34 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
35
36 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
37    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
38 int dodaj_iza(Cvor * tekuci, int broj);
39
40 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
41    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
42    inace vraca 0. */
43 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
44                     broj);
45
46 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
47    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
48    NULL u slucaju da takav cvor ne postoji u listi. */
49 Cvor *pretrazi_listu(Cvor * glava, int broj);
```



```

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

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

61 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
62    pokazivac na glavu liste, koji može biti promenjen u slucaju da se
63    obrise stara glava. */
64 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
65     broj);

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

73 /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
74    liste, razdvojene zapetama i uokvirene zagradama. */
75 void ispisi_listu(Cvor * glava);

77 /* Funkcija prikazuje vrednosti cvorova liste počevši od kraja ka
78    glavi liste, razdvojene zapetama i uokvirene zagradama. */
79 void ispisi_listu_unazad(Cvor * kraj);

81 #endif
83

```

#### *dvostruko\_povezana\_lista.c*

```

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

4
5  Cvor *napravi_cvor(int broj)
6  {
7      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8         alokacije */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;

12     /* Inicijalizacija polja strukture */
13     novi->vrednost = broj;

```

```
15     novi->sledeci = NULL;

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

21 void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
{
23     Cvor *pomocni = NULL;

25     /* Ako lista nije prazna, onda treba osloboditi memoriju */
    while (*adresa_glave != NULL) {
27         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
            osloboditi memoriju cvora koji predstavlja glavu liste */
29         pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
31         /* Sledeci cvor je nova glava liste */
        *adresa_glave = pomocni;
33     }
    /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
35     treba postaviti na NULL */
    *adresa_kraja = NULL;
37 }

39 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
                           adresa_kraja, int broj)
41 {
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43     Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
45         return 1;

47     /* Sledbenik novog cvora je glava stare liste */
    novi->sledeci = *adresa_glave;
49

51     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
        treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
        krajnji */
53     if (*adresa_glave != NULL)
        (*adresa_glave)->prethodni = novi;
55     else
        *adresa_kraja = novi;
57

59     /* Novi cvor je nova glava liste */
    *adresa_glave = novi;

61     /* Vraca se indikator uspesnog dodavanja */
    return 0;
63 }

65 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                        int broj)
```

```

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

```

```
119     if (novi == NULL)
120         return 1;
121
122     novi->sledeci = tekuci->sledeci;
123     novi->prethodni = tekuci;
124
125     /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
126        a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
127        na ispravne adrese */
128     if (tekuci->sledeci != NULL)
129         tekuci->sledeci->prethodni = novi;
130     tekuci->sledeci = novi;
131
132     /* Vraca se indikator uspesnog dodavanja */
133     return 0;
134 }
135
136 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
137                    broj)
138 {
139     /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste
140        */
141     if (*adresa_glave == NULL) {
142         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
143         Cvor *novi = napravi_cvor(broj);
144         if (novi == NULL)
145             return 1;
146
147         /* Azuriraju se vrednosti pocetka i kraja liste */
148         *adresa_glave = novi;
149         *adresa_kraja = novi;
150
151         /* Vraca se indikator uspesnog dodavanja */
152         return 0;
153     }
154
155     /* Ukoliko je vrednost glave liste veka ili jednaka od nove
156        vrednosti onda novi cvor treba staviti na pocetak liste */
157     if ((*adresa_glave)->vrednost >= broj) {
158         return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
159     }
160
161     /* Pronazi se cvor iza koga treba uvezati novi cvor */
162     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
163     /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
164     if (dodaj_iza(pomocni, broj) == 1)
165         return 1;
166
167     /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
168        novi cvor poslednji u listi. */
169     if (pomocni == *adresa_kraja)
170         *adresa_kraja = pomocni->sledeci;
```

```
    return 0;
171 }

173 Cvor *pretrazi_listu(Cvor * glava, int broj)
174 {
175     /* Obilaze se cvorovi liste */
176     for (; glava != NULL; glava = glava->sledeci)
177         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
178            se obustavlja */
179         if (glava->vrednost == broj)
180             return glava;
181
182     /* Nema trazenog broja u listi i vraca se NULL */
183     return NULL;
184 }

185 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
186 {
187     /* Obilaze se cvorovi liste */
188     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
189     for (; glava != NULL && glava->vrednost <= broj;
190          glava = glava->sledeci)
191         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
192            se obustavlja */
193         if (glava->vrednost == broj)
194             return glava;
195
196     /* Nema trazenog broja u listi i bice vrateno NULL */
197     return NULL;
198 }

201 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
202    lako realizovati jer on sadrzi pokazivace na svog sledbenika i
203    prethodnika u listi. U funkciji se bise cvor zadat argumentom
204    tekuci */
205 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
206                  tekuci)
207 {
208     /* Ako je tekuci NULL pokazivac, nema sta da se brise */
209     if (tekuci == NULL)
210         return;
211
212     /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da
213        njegov sledbenik bude sledbenik tekuceg cvora */
214     if (tekuci->prethodni != NULL)
215         tekuci->prethodni->sledeci = tekuci->sledeci;
216
217     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
218        treba da bude prethodnik tekuceg cvora */
219     if (tekuci->sledeci != NULL)
220         tekuci->sledeci->prethodni = tekuci->prethodni;
221 }
```

```
223  /* Ako je glava cvor koji se brise, nova glava liste ce biti
    sledbenik stare glave */
225  if (tekuci == *adresa_glave)
    *adresa_glave = tekuci->sledeci;

227  /* Ako je cvor koji se brise poslednji u listi, azurira se i
    pokazivac na kraj liste */
229  if (tekuci == *adresa_kraja)
    *adresa_kraja = tekuci->prethodni;

231  /* Oslobadja se dinamicki alociran prostor za cvor tekuci */
233  free(tekuci);
}

235  void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
    )
237  {
    Cvor *tekuci = *adresa_glave;

239

    /* Sve dok ima cvorova cija je vrednost jednaka zadatom broj, takvi
    cvorovi se brisu iz liste. */
241    while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
243        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}

245  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
    adresa_kraja, int broj)
247  {
249    Cvor *tekuci = *adresa_glave;

251    /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
    takvi cvorovi se brisu iz liste. */
253    while ((tekuci =
        pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
255        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}

257  void ispisi_listu(Cvor * glava)
259  {
    putchar('[');
261    /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
    pocetka prema kraju liste */
263    for (; glava != NULL; glava = glava->sledeci) {
        printf("%d", glava->vrednost);
265        if (glava->sledeci != NULL)
            printf(", ");
267    }

269    printf("]\n");
}

271  void ispisi_listu_unazad(Cvor * kraj)
```

```

273 {
    putchar('[');
275 /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
    prema pocetku liste */
277 for (; kraj != NULL; kraj = kraj->prethodni) {
    printf("%d", kraj->vrednost);
279     if (kraj->prethodni != NULL)
        printf(", ");
281 }
    printf("]\n");
283 }

```

*main\_a.c*

```

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

5  /* 1) Glavni program */
   int main()
7  {
    /* Lista je prazna na pocetku */
9    /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
    da bi operacije poput dodavanja na kraj liste i ispisivanja
11   liste unazad bile efikasne poput dodavanja na pocetak liste i
    ispisivanja liste od pocetnog do poslednjeg cvora. */
13   Cvor *glava = NULL;
    Cvor *kraj = NULL;
15   Cvor *trazeni = NULL;
    int broj;

17   /* Testira se funkcija za dodavanja novog broja na pocetak liste */
19   printf("Unesite brojeve: (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
21       /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
        memorije za novi cvor. Memoriju alociranu za cvorove liste
        treba osloboditi pre napustanja programa */
23       if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
25           fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava, &kraj);
27           exit(EXIT_FAILURE);
        }
29       printf("\tLista: ");
        ispisi_listu(glava);
31     }

33   /* Testira se funkcija za pretragu liste */
    printf("\nUnesite broj koji se trazi u listi: ");
35   scanf("%d", &broj);

37   /* Pokazivac trazeni dobija vrednost rezultata pretrage */

```

```
trazeni = pretrazi_listu(glava, broj);
39 if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
41 else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
43
/* Ispisuje se lista unazad */
45 printf("\nLista ispisana u nazad: ");
ispisi_listu_unazad(kraj);
47
/* Oslobadja se memorija zauzeta za cvorove liste */
49 oslobodi_listu(&glava, &kraj);

51 exit(EXIT_SUCCESS);
}
```

*main\_b.c*

```
#include <stdio.h>
2 #include <stdlib.h>
#include "dvostruko_povezana_lista.h"
4
/* 2) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku. */
    Cvor *glava = NULL;
10    Cvor *kraj = NULL;
    int broj;
12
    /* Testira se funkcija za dodavanja novog broja na kraj liste */
14    printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
16        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
           memorije za novi cvor. Memoriiju alociranu za cvorove liste
           treba osloboditi pre napustanja programa */
18        if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
20            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava, &kraj);
22            exit(EXIT_FAILURE);
        }
24        printf("\tLista: ");
        ispisi_listu(glava);
26    }

28    /* Testira se funkcija za brisanje elemenata iz liste */
    printf("\nUnesite broj koji se brise iz liste: ");
30    scanf("%d", &broj);

32    /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
       procitanom sa ulaza */
}
```



```

34   obrisi_cvor(&glava, &kraj, broj);

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

38   /* Ispisuje se lista unazad */
40   printf("\nLista ispisana u nazad: ");
   ispisi_listu_unazad(kraj);

42   /* Oslobadja se memorija zauzeta za cvorove liste */
44   oslobodi_listu(&glava, &kraj);

46   exit(EXIT_SUCCESS);
}

```

*main.c.c*

```

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

5  /* 3) Glavni program */
   int main()
7  {
   /* Lista je prazna na pocetku */
9   Cvor *glava = NULL;
   Cvor *kraj = NULL;
11  Cvor *trazeni = NULL;
   int broj;

13   /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
15    bude uredjena neopadajuće */
   printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
17   while (scanf("%d", &broj) > 0) {
   /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
19    memorije za novi cvor. Memoriju alociranu za cvorove liste
    treba osloboditi pre napustanja programa */
21    if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
        fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23        oslobodi_listu(&glava, &kraj);
        exit(EXIT_FAILURE);
25    }
        printf("\tLista: ");
27        ispisi_listu(glava);
   }

29   /* Testira se funkcija za pretragu liste */
31   printf("\nUnosite broj koji se trazi u listi: ");
   scanf("%d", &broj);

33   /* Pokazivac trazeni dobija vrednost rezultata pretrage */

```

## 4 Dinamičke strukture podataka

```
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);

41   /* Testira se funkcija za brisanje elemenata iz liste */
42   printf("\nUnesite broj koji se brise iz liste: ");
43   scanf("%d", &broj);

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

49   printf("Lista nakon brisanja: ");
50   ispisi_listu(glava);

51   /* Ispisuje se lista unazad */
52   printf("\nLista ispisana u nazad: ");
53   ispisi_listu_unazad(kraj);

55   /* Oslobadja se memorija zauzeta za cvorove liste */
56   oslobodi_listu(&glava, &kraj);

59   exit(EXIT_SUCCESS);
}
```

### Rešenje 4.14

stabla.h

```
1  #ifndef _STABLA_H_
2  #define _STABLA_H_ 1

4  /* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
5     stabla */
6  typedef struct cvor {
7     int broj;
8     struct cvor *levo;
9     struct cvor *desno;
10 } Cvor;

12 /* b) Funkcija koja alocira memoriju za novi cvor stabla,
13     inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
17     funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
18     do greske */
19 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
   postablo - Koren - Desno podstablo ) */
36 void ispisi_stablo_infiksno(Cvor * koren);

38 /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
   Levo podstablo - Desno podstablo ) */
40 void ispisi_stablo_prefiksno(Cvor * koren);

42 /* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
   podstablo - Desno postablo - Koren ) */
44 void ispisi_stablo_postfiksno(Cvor * koren);

46 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
48 void oslobodi_stablo(Cvor ** adresa_korena);

50 #endif

```

stabla.c

```

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

4 Cvor *napravi_cvor(int broj)
6 {
   /* Alocira se memorija za novi cvor i proverava se uspesnost
   alokacije. */
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   8 if (novi == NULL)
      return NULL;

   /* 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
28         /* Kreira se novi cvor */
        Cvor *novi_cvor = napravi_cvor(broj);

30         /* Proverava se uspesnost kreiranja */
        if (novi_cvor == NULL) {
32
34             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost */
            return 1;
36         }
        /* Inace ... */

38         /* Novi cvor se proglašava korenom stabla */
        *adresa_korena = novi_cvor;

40         /* I vraca se indikator uspesnosti kreiranja */
        return 0;
42     }

44     /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati broj */

46     /* Ako je zadata vrednost manja od vrednosti korena */
    if (broj < (*adresa_korena)->broj)
48
50         /* Broj se dodaje u levo podstablo */
        return dodaj_u_stablo(&(*adresa_korena)->levo, broj);

52     else
54         /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se dodaje u desno podstablo */
        return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
56     }
58 }

60 Cvor *pretrazi_stablo(Cvor * koren, int broj)
62 {

64     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
    if (koren == NULL)
66         return NULL;

68     /* Ako je trazena vrednost sadrzana u korenu */
    if (koren->broj == broj) {
```

```

70     /* Prekidamo pretragu */
72     return koren;
74 }
76 /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
78 if (broj < koren->broj)
80     /* Pretraga se nastavlja u levom podstablu */
82     return pretrazi_stablo(koren->levo, broj);
84 else
86     /* U suprotnom, pretraga se nastavlja u desnom podstablu */
88     return pretrazi_stablo(koren->desno, broj);
90 }
92 Cvor *pronadji_najmanji(Cvor * koren)
94 {
96     /* Ako je stablo prazno, prekida se pretraga */
98     if (koren == NULL)
100         return NULL;
102     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
104         levo od njega */
106     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
108         najmanju vrednost */
110     if (koren->levo == NULL)
112         return koren;
114     /* Inace, pretragu treba nastaviti u levom podstablu */
116     return pronadji_najmanji(koren->levo);
118 }
120 Cvor *pronadji_najveci(Cvor * koren)
122 {
124     /* Ako je stablo prazno, prekida se pretraga */
126     if (koren == NULL)
128         return NULL;
130     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
132         desno od njega */
134     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
136         najveću vrednost */
138     if (koren->desno == NULL)
140         return koren;
142     /* Inace, pretragu treba nastaviti u desnom podstablu */
144     return pronadji_najveci(koren->desno);
146 }

```

```
122 void obrisi_element(Cvor ** adresa_korena, int broj)
124 {
    Cvor *pomocni_cvor = NULL;
126
    /* Ako je stablo prazno, brisanje nije primenljivo */
128 if (*adresa_korena == NULL)
    return;
130
    /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
132 stabla, ona se eventualno nalazi u levom podstablu, pa treba
    rekursivno primeniti postupak na levo podstablo. Koren ovako
134 modifikovanog stabla je nepromenjen. */
    if (broj < (*adresa_korena)->broj) {
136 obrisi_element(&(*adresa_korena)->levo, broj);
    return;
138 }

140 /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
    stabla, ona se eventualno nalazi u desnom podstablu pa treba
142 rekursivno primeniti postupak na desno podstablo. Koren ovako
    modifikovanog stabla je nepromenjen. */
144 if ((*adresa_korena)->broj < broj) {
    obrisi_element(&(*adresa_korena)->desno, broj);
146 return;
    }
148
    /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
150 jednaka broju koji se brise (tj. slucaj kada treba obrisati
    koren) */
152
    /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
154 prazno stablo (vraca se NULL) */
    if ((*adresa_korena)->levo == NULL
156 && (*adresa_korena)->desno == NULL) {
        free(*adresa_korena);
158 *adresa_korena = NULL;
        return;
160 }

162 /* Ako koren ima samo levog sina, tada se brisanje vrši tako sto se
    brise koren, a novi koren postaje levi sin */
164 if ((*adresa_korena)->levo != NULL
    && (*adresa_korena)->desno == NULL) {
166 pomocni_cvor = (*adresa_korena)->levo;
    free(*adresa_korena);
168 *adresa_korena = pomocni_cvor;
    return;
170 }

172 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako sto
    se brise koren, a novi koren postaje desni sin */
```

```

174  if ((*adresa_korena)->desno != NULL
    && (*adresa_korena)->levo == NULL) {
176      pomocni_cvor = (*adresa_korena)->desno;
      free(*adresa_korena);
178      *adresa_korena = pomocni_cvor;
      return;
180  }

182  /* Slučaj kada koren ima oba sina - najpre se potrazi sledbenik
    korena (u smislu poretka) u stablu. To je upravo po vrednosti
184      najmanji cvor u desnom podstablu. On se može pronaći npr.
    funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
186      vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
    broj koji se briše). Zatim se prosto rekurzivno pozove funkcija
188      za brisanje na desno podstablu. S obzirom da u njemu treba
    obrisati najmanji element, a on zasigurno ima najviše jednog
190      potomka, jasno je da će njegovo brisanje biti obavljeno na jedan
    od jednostavnijih načina koji su gore opisani. */
192  pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
    (*adresa_korena)->broj = pomocni_cvor->broj;
194  pomocni_cvor->broj = broj;
    obriši_element(&(*adresa_korena)->desno, broj);
196  }

198  void ispisi_stablo_infiksno(Cvor * koren)
    {
200      /* Ako stablo nije prazno */
      if (koren != NULL) {
202
          /* Prvo se ispisuju svi cvorovi levo od korena */
204          ispisi_stablo_infiksno(koren->levo);

          /* Zatim se ispisiuje vrednost u korenu */
206          printf("%d ", koren->broj);

          /* Na kraju se ispisuju cvorovi desno od korena */
208          ispisi_stablo_infiksno(koren->desno);
210      }
212  }

214  void ispisi_stablo_prefiksno(Cvor * koren)
    {
216      /* Ako stablo nije prazno */
      if (koren != NULL) {
218
          /* Prvo se ispisiuje vrednost u korenu */
220          printf("%d ", koren->broj);

          /* Zatim se ispisuju svi cvorovi levo od korena */
222          ispisi_stablo_prefiksno(koren->levo);

          /* Na kraju se ispisuju svi cvorovi desno od korena */
224

```

```
226     ispisi_stablo_prefiksno(koren->desno);
227 }
228 }
229
230 void ispisi_stablo_postfiksno(Cvor * koren)
231 {
232     /* Ako stablo nije prazno */
233     if (koren != NULL) {
234
235         /* Prvo se ispisuju svi cvorovi levo od korena */
236         ispisi_stablo_postfiksno(koren->levo);
237
238         /* Zatim se ispisuju svi cvorovi desno od korena */
239         ispisi_stablo_postfiksno(koren->desno);
240
241         /* Na kraju se ispisuje vrednost u korenu */
242         printf("%d ", koren->broj);
243     }
244 }
245
246 void oslobodi_stablo(Cvor ** adresa_korena)
247 {
248     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
249     if (*adresa_korena == NULL)
250         return;
251
252     /* Inace ... */
253     /* Oslobadja se memorija zauzeta levim podstablom */
254     oslobodi_stablo(&(*adresa_korena)->levo);
255
256     /* Oslobadja se memorija zauzeta desnim podstablom */
257     oslobodi_stablo(&(*adresa_korena)->desno);
258
259     /* Oslobadja se memorija zauzeta korenom */
260     free(*adresa_korena);
261
262     /* Proglasava se stablo praznim */
263     *adresa_korena = NULL;
264 }
```

*main.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 int main()
6 {
7     Cvor *koren;
8     int n;
```



```
9   Cvor *trazeni_cvor;

11  /* Proglasava se stablo praznim */
   koren = NULL;

13

15  /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
   dodavanja */
   printf("Unesite brojeve (CTRL+D za kraj unosa): ");
17  while (scanf("%d", &n) != EOF) {
       if (dodaj_u_stablo(&koren, n) == 1) {
19         fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
           oslobodi_stablo(&koren);
21         return 0;
       }
23  }

25  /* Generisu se trazeni ispisi: */
   printf("\nInfiksni ispis: ");
27   ispisi_stablo_infiksno(koren);
   printf("\nPrefiksni ispis: ");
29   ispisi_stablo_prefiksno(koren);
   printf("\nPostfiksni ispis: ");
31   ispisi_stablo_postfiksno(koren);

33  /* Demonstrira se rad funkcije za pretragu */
   printf("\nTrazi 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
41       printf("Broj se nalazi u stablu!\n");

43  /* Demonstrira se rad funkcije za brisanje */
   printf("Briše se broj: ");
45   scanf("%d", &n);
   obrisi_element(&koren, n);
47   printf("Rezultujuće stablo: ");
   ispisi_stablo_infiksno(koren);
49   printf("\n");

51  /* Oslobadja se memorija zauzeta stablom */
   oslobodi_stablo(&koren);

53   return 0;
55 }
```

## Rešenje 4.15

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX 50
7
8  /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
9  typedef struct cvor {
10     char *rec;
11     int broj;
12     struct cvor *levo;
13     struct cvor *desno;
14 } Cvor;
15
16 /* Funkcija koja kreira novi cvor stabla */
17 Cvor *napravi_cvor(char *rec)
18 {
19     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
20     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
21     if (novi_cvor == NULL)
22         return NULL;
23
24     /* Alocira se memorija za zadatu rec: potrebno je rezervisati
       memoriju za svaki karakter reci ukljucujuci i terminirajucu nulu
       */
25     novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
26     if (novi_cvor->rec == NULL) {
27         free(novi_cvor);
28         return NULL;
29     }
30
31     /* Inicijalizuju se polja u novom cvoru */
32     strcpy(novi_cvor->rec, rec);
33     novi_cvor->broj = 1;
34     novi_cvor->levo = NULL;
35     novi_cvor->desno = NULL;
36
37     /* Vraca se adresa novog cvora */
38     return novi_cvor;
39 }
40
41 /* Funkcija koja dodaje novu rec u stablo - ukoliko je dodavanje
   uspesno povratna vrednost je 0, u suprotnom povratna vrednost je 1
   */
42 int dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
43 {
44     /* Ako je stablo prazno */
45     if (*adresa_korena == NULL) {
46         /* Kreira se cvor koji sadrzi zadatu rec */
47         *adresa_korena = napravi_cvor(rec);
48         return 0;
49     }
50     /* ... (ostatak funkcije) ... */
51     return 1;
52 }
```

```

Cvor *novi_cvor = napravi_cvor(rec);
54 /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
56         /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
           */
58         return 1;
    }
60 /* Inace... */
    /* Novi cvor se proglašava korenom stabla */
62     *adresa_korena = novi_cvor;

    /* I vraca se indikator uspesnog dodavanja */
64     return 0;
66 }

68 /* Ako stablo nije prazno, trazi 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
76     /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
        desno podstablo */
78     if (strcmp(rec, (*adresa_korena)->rec) > 0)
        return dodaj_u_stablo(&(*adresa_korena)->desno, rec);
80
else
82     /* Ako je rec jednaka reci u korenu, uvecava se njen broj
        pojavljivanja */
84     (*adresa_korena)->brojac++;
}

86 /* Funkcija koja oslobadja memoriju zauzetu stablom */
88 void oslobodi_stablo(Cvor ** adresa_korena)
{
90     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*adresa_korena == NULL)
92         return;

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

    /* Oslobadja se memorija zauzeta desnim podstablom */
98     oslobodi_stablo(&(*adresa_korena)->desno);
100
    /* Oslobadja se memorija zauzeta korenom */
102     free((*adresa_korena)->rec);
    free(*adresa_korena);

```

```
104      /* Stablo se proglašava praznim */
106      *adresa_korena = NULL;
108  }

109  /* Funkcija koja pronalazi cvor koji sadrži najfrekventniju rec (rec
110     sa najvećim brojem pojavljivanja) */
111  Cvor *nadji_najfrekventniju_rec(Cvor * koren)
112  {
113      Cvor *max, *max_levo, *max_desno;
114
115      /* Ako je stablo prazno, prekida se sa pretragom */
116      if (koren == NULL)
117          return NULL;
118
119      /* Pronalazi se najfrekventnija rec u levom podstablu */
120      max_levo = nadji_najfrekventniju_rec(koren->levo);
121
122      /* Pronalazi se najfrekventnija rec u desnom podstablu */
123      max_desno = nadji_najfrekventniju_rec(koren->desno);
124
125      /* Traži se maksimum vrednosti pojavljivanja reci iz levog
126         podstabla, korena i desnog podstabla */
127      max = koren;
128      if (max_levo != NULL && max_levo->brojac > max->brojac)
129          max = max_levo;
130      if (max_desno != NULL && max_desno->brojac > max->brojac)
131          max = max_desno;
132
133      /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
134      return max;
135  }

136
137  /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
138     praveći brojem pojavljivanja */
139  void prikazi_stablo(Cvor * koren)
140  {
141      /* Ako je stablo prazno, završava se sa ispisom */
142      if (koren == NULL)
143          return;
144
145      /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz levog
146         podstabla */
147      prikazi_stablo(koren->levo);
148
149      /* Zatim rec iz korena */
150      printf("%s: %d\n", koren->rec, koren->brojac);
151
152      /* I nastavlja se sa ispisom reci iz desnog podstabla */
153      prikazi_stablo(koren->desno);
154  }
```

```
156 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
157    niz rec. Maksimalna duzina reci je odredjena argumentom max.
158    Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
159    suprotnom. Rec je niz malih ili velikih slova. */
160 int procitaj_rec(FILE * f, char rec[], int max)
161 {
162     /* Karakter koji se cita */
163     int c;
164
165     /* Indeks pozicije na koju se smesta procitani karakter */
166     int i = 0;
167
168     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
169        nije stiglo do kraja datoteke... */
170     while (i < max - 1 && (c = fgetc(f)) != EOF) {
171         /* Proverava se da li je procitani karakter slovo */
172         if (isalpha(c))
173             /* Ako jeste, smesta se u niz - pritom se vrsi konverzija u
174                mala slova jer program treba da bude neosetljiv na razliku
175                izmedju malih i velikih slova */
176             rec[i++] = tolower(c);
177
178         else
179             /* Ako nije, proverava se da li je procitano barem jedno slovo
180                nove reci */
181             /* Ako jeste, prekida se sa citanjem */
182             if (i > 0)
183                 break;
184
185         /* U suprotnom se ide na sledecu iteraciju */
186     }
187
188     /* Dodaje se na rec terminirajuca nula */
189     rec[i] = '\0';
190
191     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
192     return i > 0 ? 0 : EOF;
193 }
194
195 int main(int argc, char **argv)
196 {
197     Cvor *koren = NULL, *max;
198     FILE *f;
199     char rec[MAX];
200
201     /* Provera da li je navedeno ime datoteke prilikom pokretanja
202        programa */
203     if (argc < 2) {
204         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
205         exit(EXIT_FAILURE);
206     }
```

```

208  /* Priprema datoteke za citanje */
    if ((f = fopen(argv[1], "r")) == NULL) {
210      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
                argv[1]);
212      exit(EXIT_FAILURE);
    }

214
    /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
       pretrage uz proveru uspesnosti dodavanja */
216    while (procitaj_rec(f, rec, MAX) != EOF) {
218      if (dodaj_u_stablo(&koren, rec) == 1) {
          fprintf(stderr, "Neuspelo dodavanje reci %s\n", rec);
220          oslobodi_stablo(&koren);
          exit(EXIT_FAILURE);
222      }
    }

224
    /* Posto je citanjem reci zavrшено, zatvara se datoteka */
226    fclose(f);

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

232
    /* Pronalazi se najfrekventnija rec */
    max = najfrekventniju_rec(koren);
234

    /* Ako takve reci nema... */
236    if (max == NULL)

238        /* Ispisuje se odgovarajuće obavestjenje */
        printf("U tekstu nema reci!\n");
240
    else
242        /* Inace, ispisuje se broj pojavljivanja reci */
        printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
244              max->rec, max->brojac);

246
    /* Oslobadja se dinamicki alociran prostor za stablo */
    oslobodi_stablo(&koren);
248
    exit(EXIT_SUCCESS);
250 }

```

### Rešenje 4.16

```

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

```

```

6  #define MAX_IME_DATOTEKE 50
   #define MAX_CIFARA 13
8  #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime, broj
   telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
   char ime_i_prezime[MAX_IME_I_PREZIME];
14   char telefon[MAX_CIFARA];
   struct cvor *levo;
16   struct cvor *desno;
   } Cvor;

18
19 /* Funkcija koja kreira novi cvora stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
   {
22   /* Alocira se memorija za novi cvor i proverava se uspesnost
      alokacije. */
24   Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
   if (novi_cvor == NULL)
26     return NULL;

28   /* Inicijalizuju se polja novog cvora */
   strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
30   strcpy(novi_cvor->telefon, telefon);
   novi_cvor->levo = NULL;
32   novi_cvor->desno = NULL;

34   /* Vraca se adresa novog cvora */
   return novi_cvor;
36 }

38 /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
   ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
40 povratna vrednost je 1 */
   int
42 dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
   char *telefon)
44 {
   /* Ako je stablo prazno */
46   if (*adresa_korena == NULL) {
   /* Kreira se novi cvor */
48   Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
   /* Proverava se uspesnost kreiranja novog cvora */
50   if (novi_cvor == NULL) {
   /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
52    */
   return 1;
54   }
   /* Inace... */
56   /* Novi cvor se proglašava korenom stabla */
   *adresa_korena = novi_cvor;

```

```

58      /* I vraća se indikator uspešnog dodavanja */
60      return 0;
61  }

62
63      /* Ako stablo nije prazno, traži se odgovarajuća pozicija za novi
64      unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
65      treba da bude pretraživacko po ovom polju */
66
67      /* Ako je zadato ime i prezime leksikografski manje od imena i
68      prezimena sadržanog u korenu, podaci se dodaju u levo podstablo
69      */
70      if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
71          < 0)
72          return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
73                                telefon);
74
75      else
76          /* Ako je zadato ime i prezime leksikografski veće od imena i
77          prezimena sadržanog u korenu, podaci se dodaju u desno
78          podstablo */
79          if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
80              return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
81                                    telefon);
82      }

83      /* Funkcija koja oslobadja memoriju zauzetu stablom */
84      void oslobodi_stablo(Cvor ** adresa_korena)
85      {
86          /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
87          if (*adresa_korena == NULL)
88              return;
89
90          /* Inace ... */
91          /* Oslobadja se memorija zauzeta levim podstablom */
92          oslobodi_stablo(&(*adresa_korena)->levo);
93
94          /* Oslobadja se memorija zauzeta desnim podstablom */
95          oslobodi_stablo(&(*adresa_korena)->desno);
96
97          /* Oslobadja se memorija zauzeta korenom */
98          free(*adresa_korena);
99
100         /* Stablo se proglašava praznim */
101         *adresa_korena = NULL;
102     }

103
104     /* Funkcija koja ispisuje imenik u leksikografskom poretku */
105     /* Napomena: ova funkcija nije tražena u zadatku ali se može
106     koristiti za proveru da li je stablo lepo kreirano ili ne */
107     void prikazi_stablo(Cvor * koren)
108     {

```



```

110  /* Ako je stablo prazno, završava se sa ispisom */
111  if (koren == NULL)
112      return;
113
114  /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
115     podstabla */
116  prikazi_stablo(koren->levo);
117
118  /* Zatim se ispisuju podaci iz korena */
119  printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
120
121  /* I nastavlja se sa ispisom podataka iz desnog podstabla */
122  prikazi_stablo(koren->desno);
123  }
124
125  /* Funkcija učitava sledeći kontakt iz zadate datoteke i upisuje ime
126     i prezime i broj telefona u odgovarajuće nizove. Maksimalna dužina
127     imena i prezimena određena je konstantom MAX_IME_PREZIME, a
128     maksimalna dužina broja telefona konstantom MAX_CIFARA. Funkcija
129     vraća EOF ako nema više kontakata ili 0 u suprotnom. */
130  int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
131  {
132      /* Karakter koji se čita */
133      int c;
134
135      /* Indeks pozicije na koju se smesta procitani karakter */
136      int i = 0;
137
138      /* Linije datoteke koje se obrađuju su formata Ime Prezime
139         BrojTelefona */
140
141      /* Preskaku se eventualne praznine sa početka linije datoteke */
142      while ((c = fgetc(f)) != EOF && isspace(c));
143
144      /* Prvo procitano slovo upisuje se u ime i prezime */
145      if (!feof(f))
146          ime_i_prezime[i++] = c;
147
148      /* Naznaka kraja citanja imena i prezimena će biti pojava prve
149         cifre tako da se citanje vrši sve dok se ne nađe na cifru.
150         Pritom treba voditi računa da li ima dovoljno mesta za smestanje
151         procitanog karaktera i da se slučajno ne dodje do kraja datoteke
152         */
153      while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
154          if (!isdigit(c))
155              ime_i_prezime[i++] = c;
156
157          else if (i > 0)
158              break;
159      }
160
161      /* Upisuje se terminirajuća nula na mesto poslednjeg procitanog

```

```

162     blanko karaktera */
ime_i_prezime[--i] = '\0';

164 /* I pocinje se sa citanjem broja telefona */
i = 0;

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

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

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

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

184 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
    */
186 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
188 {
    /* Ako je imenik prazan, zavrшава se sa pretragom */
190     if (koren == NULL)
        return NULL;

192     /* Ako je trazeno ime i prezime sadržano u korenu, takodje se
        završava sa pretragom */
194     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
        return koren;

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

202     else
204         /* u suprotnom, pretraga se nastavlja desno */
        return pretrazi_imenik(koren->desno, ime_i_prezime);
206 }

208 int main(int argc, char **argv)
{
210     char ime_datoteke[MAX_IME_DATOTEKE];
    Cvor *koren = NULL;
212     Cvor *trazeni;

```

```
FILE *f;
214 char ime_i_prezime[MAX_IME_I_PREZIME];
char telefon[MAX_CIFARA];
216 char c;
int i;

218 /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
220 printf("Unesite ime datoteke: ");
scanf("%s", ime_datoteke);
222 getchar();
if ((f = fopen(ime_datoteke, "r")) == NULL) {
224     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
ime_datoteke);
226     exit(EXIT_FAILURE);
}

228 /* Citaju se podaci iz datoteke i smestanju u binarno stablo
pretrage uz proveru uspesnosti dodavanja */
230 while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
232     if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
        fprintf(stderr, "Neuspelo dodavanje podataka za osobu %s\n",
234             ime_i_prezime);
        oslobodi_stablo(&koren);
236         exit(EXIT_FAILURE);
    }

238 /* Zatvara se datoteka */
240 fclose(f);

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

250     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
funktionalnost */
252     if (strcmp(ime_i_prezime, "KRAJ") == 0)
254         break;

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

264 /* Oslobadja se memorija zauzeta imenikom */
```

```
    oslobodi_stablo(&koren);  
266     exit(EXIT_SUCCESS);  
268 }
```

### Rešenje 4.17

```
1  #include <stdio.h>  
   #include <stdlib.h>  
3  #include <string.h>  
  
5  #define MAX 51  
  
7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime  
   studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg  
9  jezika i redom pokazivace na levo i desno podstablo */  
   typedef struct cvor_stabla {  
11     char ime[MAX];  
     char prezime[MAX];  
13     double uspeh;  
     double matematika;  
15     double jezik;  
     struct cvor_stabla *levo;  
17     struct cvor_stabla *desno;  
   } Cvor;  
19  
   /* Funkcija kojom se kreira cvor stabla */  
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 je 0, u suprotnom
```

```

45     povratna_vrednost je 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 vrednost
57             */
            return 1;
59         }
        /* Inace... */
61         /* Novi cvor se proglašava korenom stabla */
        *koren = novi_cvor;
63
        /* I vraca se indikator uspesnog dodavanja */
65         return 0;
    }
67
    /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
69     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);
    else
75        return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
                                matematika, jezik);
77 }

79
/* Funkcija kojom se oslobadja memorija zauzeta stablom */
81 void oslobodi_stablo(Cvor ** koren)
{
83     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*koren == NULL)
85         return;

87     /* Inace ... */
    /* Oslobadja se memorija zauzeta levim podstablom */
89     oslobodi_stablo(&(*koren)->levo);

    /* Oslobadja se memorija zauzeta desnim podstablom */
91     oslobodi_stablo(&(*koren)->desno);
93
    /* Oslobadja se memorija zauzeta korenom */
95     free(*koren);

```

```

97  /* Stablo se proglašava praznim */
    *koren = NULL;
99  }

101
103  /* Funkcija ispisuje sadržaj stabla. Ukoliko je vrednost argumenta
    položili jednaka 0 ispisuju se informacije o učenicima koji nisu
    položili prijemni, a ako je vrednost argumenta različita od nule,
105  ispisuju se informacije o učenicima koji su položili prijemni */
    void stampaJ(Cvor * koren, int položili)
107  {
        /* Stablo je prazno - prekida se sa ispisom */
109        if (koren == NULL)
            return;

111
        /* Stampaju se informacije iz levog podstabla */
113        stampaJ(koren->levo, položili);

115
        /* Stampaju se informacije iz korenog cvora */
        if (položili && koren->matematika + koren->jezik >= 10)
117            printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
                koren->prezime, koren->uspeh, koren->matematika,
119                koren->jezik,
                koren->uspeh + koren->matematika + koren->jezik);
121        else if (!položili && koren->matematika + koren->jezik < 10)
            printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
123                koren->prezime, koren->uspeh, koren->matematika,
                koren->jezik,
125                koren->uspeh + koren->matematika + koren->jezik);

127
        /* Stampaju se informacije iz desnog podstabla */
        stampaJ(koren->desno, položili);
129    }

131  /* Funkcija koja određuje koliko studenata nije položilo prijemni
    ispit */
133  int nisu_položili(Cvor * koren)
    {
135        /* Ako je stablo prazno, broj onih koji nisu položili je 0 */
        if (koren == NULL)
137            return 0;

139
        /* Pretraga se vrši i u levom i u desnom podstablu - ako uslov za
            polaganje nije ispunjen za koreni cvor, broj studenata se
141            uvecava za 1 */
        if (koren->matematika + koren->jezik < 10)
143            return 1 + nisu_položili(koren->levo) +
                nisu_položili(koren->desno);
145
        return nisu_položili(koren->levo) + nisu_položili(koren->desno);
147    }

```

```

149 int main(int argc, char **argv)
150 {
151     FILE *in;
152     Cvor *koren;
153     char ime[MAX], prezime[MAX];
154     double uspeh, matematika, jezik;
155
156     /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
157     in = fopen("prijemni.txt", "r");
158     if (in == NULL) {
159         fprintf(stderr,
160             "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
161         exit(EXIT_FAILURE);
162     }
163
164     /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
165        dodavanja */
166     koren = NULL;
167     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
168         &matematika, &jezik) != EOF) {
169         if (dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik)
170             == 1) {
171             fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
172                 prezime);
173             oslobodi_stablo(&koren);
174             exit(EXIT_FAILURE);
175         }
176     }
177
178     /* Zatvaranje datoteke */
179     fclose(in);
180
181     /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
182     stampaj(koren, 1);
183
184     /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
185        prijemni */
186     if (nisu_polozili(koren) != 0)
187         printf("-----\n");
188
189     /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
190     stampaj(koren, 0);
191
192     /* Oslobadja se memorija zauzeta stablom */
193     oslobodi_stablo(&koren);
194
195     exit(EXIT_SUCCESS);
196 }

```

## 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;
26
27     /* Inicijalizuju se polja strukture */
28     strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
30     novi->dan = dan;
31     novi->mesec = mesec;
32     novi->levo = NULL;
33     novi->desno = NULL;
34
35     /* Vraca se adresa novog cvora */
36     return novi;
37 }
38
39 /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
40    uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
41    dodavanje uspesno povratna vrednost je 0, u suprotnom povratna
42    vrednost je 1 */
43 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
44                    int dan, int mesec)
45 {
46     /* Ako je stablo prazno */
47     if (*koren == NULL) {
48
49         /* Kreira se novi cvor */
50         Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
51         /* Proverava se uspesnost kreiranja novog cvora */
```



```

53     if (novi_cvor == NULL) {
        /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost */
55         return 1;
    }
57     /* Inace... */
    /* Novi cvor se proglašava korenom stabla */
59     *koren = novi_cvor;

61     /* I vraca se indikator uspešnog dodavanja */
    return 0;
63 }

65 /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
    meseca */
67 if (mesec < (*koren)->mesec)
    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, mesec);
71 else
    return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec)
    ;
73 }

75 /* Funkcija vrši pretragu stabla i vraca cvor sa traženim datumom */
    Cvor *pretrazi(Cvor * koren, int dan, int mesec)
77 {
    /* Stablo je prazno, obustavlja se pretraga */
79     if (koren == NULL)
        return NULL;
81
    /* Ako je traženi datum u korenu */
83     if (koren->dan == dan && koren->mesec == mesec)
        return koren;
85
    /* Ako je mesec traženog datuma manji od meseca sadržanog u korenu
    ili ako su meseci isti ali je dan traženog datuma manji od
    aktuelnog datuma, pretražuje se levo podstablo - pre toga se
    svakako proverava da li leva grana postoji - ako ne postoji
    treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
    */
91     if (mesec < koren->mesec
        || (mesec == koren->mesec && dan < koren->dan)) {
93         if (koren->levo == NULL)
            return koren;
        else
95             return pretrazi(koren->levo, dan, mesec);
97     }

99     /* Inace se nastavlja pretraga u desnom delu */
    return pretrazi(koren->desno, dan, mesec);
101 }

```

```
103 /* Funkcija koja pronalazi najmanji datum u stablu */
Cvor *pronadji_najmanji_datum(Cvor * koren)
105 {
    /* Stablo je prazno, obustavlja se pretraga */
107     if (koren == NULL)
        return NULL;
109
    /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
    sadrzi najmanji datum */
111     if (koren->levo == NULL)
        return koren;
113     else
        /* Inace, trazimo manji datum u levom podstablu */
        return pronadji_najmanji_datum(koren->levo);
117 }

119 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
    */
void datum_u_nisku(int dan, int mesec, char datum[])
121 {
    if (dan < 10) {
123         datum[0] = '0';
        datum[1] = dan + '0';
125     } else {
        datum[0] = dan / 10 + '0';
127         datum[1] = dan % 10 + '0';
    }
129     datum[2] = '.';

131     if (mesec < 10) {
        datum[3] = '0';
133         datum[4] = mesec + '0';
    } else {
135         datum[3] = mesec / 10 + '0';
        datum[4] = mesec % 10 + '0';
137     }
    datum[5] = '.';
139     datum[6] = '\\0';
}

141
/* Funkcija koja oslobadja memoriju zauzetu stablom */
143 void oslobodi_stablo(Cvor ** adresa_korena)
{
145     /* Stablo je prazno */
    if (*adresa_korena == NULL)
147         return;

149     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
    if ((*adresa_korena)->levo)
151         oslobodi_stablo(&(*adresa_korena)->levo);
```

```

153  /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
154  if ((*adresa_korena)->desno)
155      oslobodi_stablo(&(*adresa_korena)->desno);

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

159  /* Proglasava se stablo praznim */
160  *adresa_korena = NULL;
161  }

163  int main(int argc, char **argv)
164  {
165      FILE *in;
166      Cvor *koren;
167      Cvor *slavljenik;
168      char ime[MAX_NISKA], prezime[MAX_NISKA];
169      int dan, mesec;
170      char datum[7];

172  /* Provera da li je zadato ime ulazne datoteke */
173  if (argc < 2) {
174      /* Ako nije, ispisuje se poruka i prekida se sa izvršavanjem
175       programa */
176      fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
177      exit(EXIT_FAILURE);
178  }

180  /* Inace, priprema se datoteka za citanje */
181  in = fopen(argv[1], "r");
182  if (in == NULL) {
183      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
184              argv[1]);
185      exit(EXIT_FAILURE);
186  }

188  /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja
189  */
190  koren = NULL;
191  while (fscanf
192      (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
193      if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
194          fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
195                  prezime);
196          oslobodi_stablo(&koren);
197          exit(EXIT_FAILURE);
198      }

199  /* Datoteka se zatvara */
200  fclose(in);

202  /* Omogucuje se pretraga podataka */

```

```
205 while (1) {
    /* Ucitava se novi datum */
207 printf("Unesite datum: ");
    if (scanf("%d.%d.", &dan, &mesec) == EOF)
209 break;

211 /* Pretrazuje se stablo */
    slavljenik = pretrazi(koren, dan, mesec);
213

    /* Ispisuju se pronadjeni podaci */
215

    /* Ako slavljenik nije pronadjen, to moze znaci da: */
217 /* 1. drvo je prazno */
    if (slavljenik == NULL && koren == NULL) {
219 printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
        continue;
221 }
    /* 2. posle datuma koji je unesen, nema podataka u stablu - u
223 ovom slucaju se pretraga vrši pocevši od naredne godine i
    ispisuje se najmanji datum */
225 if (slavljenik == NULL) {
        slavljenik = pronadji_najmanji_datum(koren);
227 datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
        printf("Slavljenik: %s %s %s\n", slavljenik->ime,
229 slavljenik->prezime, datum);
        continue;
231 }

233 /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
    /* 1. Pronadjeni su tacni podaci */
235 if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
        printf("Slavljenik: %s %s\n", slavljenik->ime,
237 slavljenik->prezime);
        continue;
239 }

241 /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
    datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
243 printf("Slavljenik: %s %s %s\n", slavljenik->ime,
        slavljenik->prezime, datum);
245 }

247 /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
249

    exit(EXIT_SUCCESS);
251 }
```

### Rešenje 4.19

```
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 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, jednaka su */
13     if (koren1 == NULL && koren2 == NULL)
14         return 1;
15
16     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
17     if (koren1 == NULL || koren2 == NULL)
18         return 0;
19
20     /* Ako su oba stabla neprazna i u korenu se nalaze različite
21        vrednosti, može se zaključiti da se razlikuju */
22     if (koren1->broj != koren2->broj)
23         return 0;
24
25     /* Inače, proverava se da li vazi i jednakost levih i desnih
26        podstabala */
27     return (identitet(koren1->levo, koren2->levo)
28             && identitet(koren1->desno, koren2->desno));
29 }
30
31 int main()
32 {
33     int broj;
34     Cvor *koren1, *koren2;
35
36     /* Učitavaju 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, "Neuspelo dodavanje broja %d\n", broj);
43             oslobodi_stablo(&koren1);
44             return 0;
45         }
46         scanf("%d", &broj);
47     }
48
49     /* Učitavaju se elementi drugog stabla */
50     koren2 = NULL;
51     printf("Drugo stablo: ");
```

```
52  scanf("%d", &broj);
53  while (broj != 0) {
54      if (dodaj_u_stablo(&koren2, broj) == 1) {
55          fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
56          oslobodi_stablo(&koren2);
57          return 0;
58      }
59      scanf("%d", &broj);
60  }

62  /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
63     njen rezultat. */
64  if (identitet(koren1, koren2))
65      printf("Stabla jesu identicna.\n");
66  else
67      printf("Stabla nisu identicna.\n");
68
69  /* Oslobadja se memorija zauzeta stablima */
70  oslobodi_stablo(&koren1);
71  oslobodi_stablo(&koren2);
72
73  return 0;
74 }
```

### Rešenje 4.20

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

6
7  /* Funkcija kreira novo stablo identicno stablu koje je dato korenom.
8     Povratna vrednost funkcije je 0 ukoliko je kopiranje uspesno,
9     odnosno 1 ukoliko je doslo do greske */
10 int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
11 {
12     /* Izlaz iz rekurzije */
13     if (koren == NULL) {
14         *duplikat = NULL;
15         return 0;
16     }

18     /* Duplira se koren stabla i postavlja da bude koren novog stabla
19        */
20     *duplikat = napravi_cvor(koren->broj);
21     if (*duplikat == NULL) {
22         return 1;
23     }

24     /* Rekurzivno se dupliraju levo i desno podstablo i njihove adrese
```

```

    se cuvaju redom u pokazivacima na levo i desno podstablo korena
    duplikata */
26 int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
28 int kopija_desno =
    kopiraj_stablo(koren->desno, &(*duplikat)->desno);
30
    /* Ako je uspesno duplirano i levo i desno podstablo */
32 if (kopija_levo == 0 && kopija_desno == 0)
    /* Uspesno je duplirano i celo stablo */
34     return 0;
    /* Inace, prijavljuje se da je doslo do greske */
36     return 1;
38 }

40 /* Funkcija izracunava uniju dva skupa predstavljena stablima -
    rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
    Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
    uspesno, odnosno 1 ukoliko je doslo do greske */
42
44 int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
{
    /* Ako drugo stablo nije prazno */
46     if (koren2 != NULL) {
48         /* 1. Dodaje se njegov koren u prvo stablo */
        if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
50             return 1;
        }
52
        /* 2. Rekurzivno se racuna unija levog i desnog podstabla drugog
        stabla sa prvim stablom */
54         int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
56         int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);

        /* Ako je unija podstabala uspesno kreirana */
58         if (unija_levo == 0 && unija_desno == 0)
            /* Uspesno je kreirana i unija stabala */
60             return 0;

        /* U suprotnom se prijavljuje da je doslo do greske */
62         return 1;
64     }
66
    /* Ako je drugo stablo prazno, nista se ne preduzima */
68     return 0;
70 }

72 /* Funkcija izracunava presek dva skupa predstavljana stablima -
    rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
    Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka
    uspesno, odnosno 1 ukoliko je doslo do greske */
74
76 int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
{

```

```

78  /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
    if (*adresa_korena1 == NULL)
        return 0;
80
    /* Inace... */
82  /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.
        iz levog i desnog podstabla prvog stabla brisu se svi oni
84  elementi koji ne postoje u drugom stablu */
    int presek_levo = kreiraj_presek(&(*adresa_korena1)->levo, koren2);
86  int presek_desno =
        kreiraj_presek(&(*adresa_korena1)->desno, koren2);
88  if (presek_levo == 0 && presek_desno == 0) {
        /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on
90  uklanja iz prvog stabla */
        if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
92  obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

94  /* Presek stabala je uspesno kreiran */
        return 0;
96  }
    /* Inece, prijavljuje se da je doslo do greske */
98  return 1;
}

100
/* Funkcija izracunava razliku dva skupa predstavljana stablima -
102 rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
    Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike
104 uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
106 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
108  if (*adresa_korena1 == NULL)
        return 0;
110
    /* Inace... */
112  /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
        tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
114  elementi koji postoje i u drugom stablu */
    int razlika_levo =
116  kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
    int razlika_desno =
118  kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
    if (razlika_levo == 0 && razlika_desno == 0) {
120  /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on
        uklanja se iz prvog stabla */
122  if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
        obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

124  /* Razlika stabala je uspesno kreirana */
        return 0;
126  }
}
128

```



```
130     /* Inece, prijavljuje se da je doslo do greske */
131     return 1;
132 }
133
134 int main()
135 {
136     Cvor *skup1;
137     Cvor *skup2;
138     Cvor *pomocni_skup = NULL;
139     int n;
140
141     /* Ucitavaju se elementi prvog skupa */
142     skup1 = NULL;
143     printf("Prvi skup: ");
144     while (scanf("%d", &n) != EOF) {
145         if (dodaj_u_stablo(&skup1, n) == 1) {
146             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
147             oslobodi_stablo(&skup1);
148             return 0;
149         }
150     }
151
152     /* Ucitavaju se elementi drugog skupa */
153     skup2 = NULL;
154     printf("Drugi skup: ");
155     while (scanf("%d", &n) != EOF) {
156         if (dodaj_u_stablo(&skup2, n) == 1) {
157             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
158             oslobodi_stablo(&skup2);
159             return 0;
160         }
161     }
162
163     /* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
164        bi se isti mogao iskoristiti i za preostale operacije */
165     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
166         oslobodi_stablo(&skup1);
167         oslobodi_stablo(&pomocni_skup);
168         return 0;
169     }
170     if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
171         oslobodi_stablo(&pomocni_skup);
172         oslobodi_stablo(&skup2);
173         return 0;
174     }
175     printf("Unija: ");
176     ispisi_stablo_infiksno(pomocni_skup);
177     putchar('\n');
178
179     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
180        operacije */
181     oslobodi_stablo(&pomocni_skup);
```

```
182  /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
183     kako bi se isti mogao iskoristiti i za preostale operacije */
184  if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
185      oslobodi_stablo(&skup1);
186      oslobodi_stablo(&pomocni_skup);
187      return 0;
188  }
189  if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
190      oslobodi_stablo(&pomocni_skup);
191      oslobodi_stablo(&skup2);
192      return 0;
193  }
194  printf("Presek: ");
195  ispisi_stablo_infiksno(pomocni_skup);
196  putchar('\n');
197
198  /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
199     operacije */
200  oslobodi_stablo(&pomocni_skup);
201
202  /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
203     kako bi se isti mogao iskoristiti i za preostale operacije */
204  if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
205      oslobodi_stablo(&skup1);
206      oslobodi_stablo(&pomocni_skup);
207      return 0;
208  }
209  if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
210      oslobodi_stablo(&pomocni_skup);
211      oslobodi_stablo(&skup2);
212      return 0;
213  }
214  printf("Razlika: ");
215  ispisi_stablo_infiksno(pomocni_skup);
216  putchar('\n');
217
218  /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
219     operacije */
220  oslobodi_stablo(&pomocni_skup);
221
222  /* Oslobadja se memorija zauzeta polaznim skupovima */
223  oslobodi_stablo(&skup1);
224  oslobodi_stablo(&skup2);
225
226  return 0;
227 }
```

### Rešenje 4.21

---

```

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  {
      int r, s;
15
      /* Stablo je prazno - u niz je smesteno 0 elemenata */
17  if (koren == NULL)
      return 0;
19
      /* Dodaju se u niz elementi iz levog podstabla */
21  r = kreiraj_niz(koren->levo, a);
23
      /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
         u niz i na osnovu nje se moze odrediti indeks novog elementa */
25
      /* Smesta se vrednost iz korena */
27  a[r] = koren->broj;
29
      /* Dodaju se elementi iz desnog podstabla */
      s = kreiraj_niz(koren->desno, a + r + 1);
31
      /* Racuna se indeks na koji treba smestiti naredni element */
33  return r + s + 1;
   }
35
   /* Funkcija sortira niz tako sto najpre elemente niza smesti u
      stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
37  desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
      kreiran i sortiran, a 1 ukoliko je doslo do greske.
39
      Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
      sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
43  sortiranje vrši u pomocnoj dinamičkoj strukturi, a ne razmenom
      elemenata niza. */
45  int sortiraj(int a[], int n)
   {
47      int i;
      Cvor *koren;
49
      /* Kreira se stablo smestanjem elemenata iz niza u stablo */
51  koren = NULL;
      for (i = 0; i < n; i++) {

```

```
53     if (dodaj_u_stablo(&koren, a[i]) == 1) {
54         oslobodi_stablo(&koren);
55         return 1;
56     }
57 }
58 /* Infiksni obilaskom stabla elementi iz stabla se prepisuju u niz
59 a */
60 kreiraj_niz(koren, a);
61
62 /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
63 */
64 oslobodi_stablo(&koren);
65
66 /* Vraca se indikator uspesnog sortiranja */
67 return 0;
68 }
69
70 int main()
71 {
72     int a[MAX];
73     int n, i;
74
75     /* Ucitavaju se dimenzija i elementi niza */
76     printf("n: ");
77     scanf("%d", &n);
78     if (n < 0 || n > MAX) {
79         printf("Greska: pogresna dimenzija niza!\n");
80         return 0;
81     }
82
83     printf("a: ");
84     for (i = 0; i < n; i++)
85         scanf("%d", &a[i]);
86
87     /* Poziva se funkcija za sortiranje */
88     if (sortiraj(a, n) == 0) {
89         /* Ako je niz uspesno sortirao, ispisuje se rezultujući niz */
90         for (i = 0; i < n; i++)
91             printf("%d ", a[i]);
92         printf("\n");
93     } else {
94         /* Inace, obavestava se korisnik da je doslo do greske */
95         printf("Greska: problem prilikom sortiranja niza!\n");
96     }
97
98     return 0;
99 }
```

### Rešenje 4.22

---

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
6
7  /* a) Funkcija koja izracunava broj cvorova stabla */
8  int broj_cvorova(Cvor * koren)
9  {
10     /* Ako je stablo prazno, broj cvorova je nula */
11     if (koren == NULL)
12         return 0;
13
14     /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
15        levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
16        zato sto treba racunati i koren */
17     return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
18 }
19
20 /* b) Funkcija koja izracunava broj listova stabla */
21 int broj_listova(Cvor * koren)
22 {
23     /* Ako je stablo prazno, broj listova je nula */
24     if (koren == NULL)
25         return 0;
26
27     /* Proverava se da li je tekuci cvor list */
28     if (koren->levo == NULL && koren->desno == NULL)
29         /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
30            uvecati broj listova za 1 */
31         return 1;
32
33     /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
34        */
35     return broj_listova(koren->levo) + broj_listova(koren->desno);
36 }
37
38 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
39 void pozitivni_listovi(Cvor * koren)
40 {
41     /* Slucaj kada je stablo prazno */
42     if (koren == NULL)
43         return;
44
45     /* Ako je cvor list i sadrzi pozitivnu vrednost */
46     if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
47         /* Stampa se */
48         printf("%d ", koren->broj);
49
50     /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
51     pozitivni_listovi(koren->levo);
52     pozitivni_listovi(koren->desno);
```

```

}
54
/* d) Funkcija koja izracunava zbir cvorova stabla */
56 int zbir_svih_cvorova(Cvor * koren)
{
58     /* Ako je stablo prazno, zbir cvorova je 0 */
    if (koren == NULL)
60         return 0;

62     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
        elemenata u podstablina */
64     return koren->broj + zbir_svih_cvorova(koren->levo) +
        zbir_svih_cvorova(koren->desno);
66 }

68 /* e) Funkcija koja izracunava najveći element stabla */
Cvor *najveći_element(Cvor * koren)
70 {
    /* Ako je stablo prazno, obustavlja se pretraga */
72     if (koren == NULL)
        return NULL;
74
    /* Zbog prirode pretraživackog stabla, vrednosti veće od korena se
76     nalaze u desnom podstablu */

78     /* Ako desnog podstabla nema */
    if (koren->desno == NULL)
80         /* Najveća vrednost je koren */
        return koren;
82
    /* Inace, najveća vrednost se traži desno */
84     return najveći_element(koren->desno);
}
86

/* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
{
90     /* Dubina praznog stabla je 0 */
    if (koren == NULL)
92         return 0;

94     /* Izracunava se dubina levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);
96
    /* Izracunava se dubina desnog podstabla */
98     int dubina_desno = dubina_stabla(koren->desno);

100    /* Dubina stabla odgovara većoj od dubina podstabala - 1 se dodaje
        jer se racuna i koren */
102    return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
104 }

```

```
106 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
107 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
108 {
109     /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazene
110        nivoa */
111
112     /* Ako nema vise cvorova, nema spustanja niz stablo */
113     if (koren == NULL)
114         return 0;
115
116     /* Ako se stiglo do trazene nivoa, vraca se 1 - to ce kasnije zbog
117        rekurzivnih poziva uvecati broj cvorova za 1 */
118     if (i == 0)
119         return 1;
120
121     /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
122        */
123     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
124         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
125 }
126
127 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
128 void ispis_nivo(Cvor * koren, int i)
129 {
130     /* Ideja je slicna ideji iz prethodne funkcije */
131
132     /* Nema vise cvorova, nema spustanja kroz stablo */
133     if (koren == NULL)
134         return;
135
136     /* Ako se stiglo do trazene nivoa - ispisuje se vrednost */
137     if (i == 0) {
138         printf("%d ", koren->broj);
139         return;
140     }
141     /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
142        desnom podstablu */
143     ispis_nivo(koren->levo, i - 1);
144     ispis_nivo(koren->desno, i - 1);
145 }
146
147 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
148    stabla */
149 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
150 {
151     /* Ako je stablo prazno, obustavlja se pretraga */
152     if (koren == NULL)
153         return NULL;
154
155     /* Ako se stiglo do trazene nivoa, takodje se prekida pretraga */
156     if (i == 0)
```

```

156     return koren;

158     /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
    Cvor *a = najveći_element_na_itom_nivou(koren->levo, i - 1);

160     /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
    Cvor *b = najveći_element_na_itom_nivou(koren->desno, i - 1);

164     /* Traži se i vraća maksimum izračunatih vrednosti */
    if (a == NULL && b == NULL)
166         return NULL;
    if (a == NULL)
168         return b;
    if (b == NULL)
170         return a;
    return a->broj > b->broj ? a : b;
172 }

174 /* j) Funkcija koja izračunava zbir cvorova na i-tom nivou */
int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
    /* Ako je stablo prazno, zbir je nula */
178     if (koren == NULL)
        return 0;

180     /* Ako se stiglo do traženog nivoa, vraća se vrednost */
182     if (i == 0)
        return koren->broj;

184     /* Inače, spustanje se nastavlja za jedan nivo nize i traže se sume
186     iz levog i desnog podstabla */
    return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
188         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
190 }

192 /* k) Funkcija koja izračunava zbir svih vrednosti u stablu koje su
    manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
    {
196     /* Ako je stablo prazno, zbir je nula */
    if (koren == NULL)
198         return 0;

200     /* Ako je vrednost u korenu manja od tražene vrednosti, zbog
    prirode pretraživačkog stabla treba obići i levo i desno
202     podstablo */
    if (koren->broj <= x)
204         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
            zbir_manjih_od_x(koren->desno, x);

206     /* Inače, računa se samo suma vrednosti iz levog podstabla jer

```



```
208     medju njima jedino moze biti onih koje zadovoljavaju uslov */
    return zbir_manjih_od_x(koren->levo, x);
210 }

212 int main(int argc, char **argv)
{
214     /* Analiza argumenata komandne linije */
    if (argc != 3) {
216         fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out nivo
            broj_za_pretragu\n");
218         return 1;
    }

220     int i = atoi(argv[1]);
    int x = atoi(argv[2]);

222     /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
    */
    Cvor *koren = NULL;
226     int broj;
    while (scanf("%d", &broj) != EOF) {
228         if (dodaj_u_stablo(&koren, broj) == 1) {
            fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
230             oslobodi_stablo(&koren);
            return 0;
232         }
    }

234     /* ispisuju se rezultati rada funkcija */
    printf("Broj cvorova: %d\n", broj_cvorova(koren));
    printf("Broj listova: %d\n", broj_listova(koren));
238     printf("Pozitivni listovi: ");
    pozitivni_listovi(koren);
240     printf("\n");
    printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
242     if (najveci_element(koren) == NULL)
        printf("Najveci element: ne postoji\n");
244     else
        printf("Najveci element: %d\n", najveci_element(koren)->broj);

246     printf("Dubina stabla: %d\n", dubina_stabla(koren));

248     printf("Broj cvorova na %d. nivou: %d\n", i,
        broj_cvorova_na_itom_nivou(koren, i));
250     printf("Elementi na %d. nivou: ", i);
    ispis_nivo(koren, i);
252     printf("\n");
    if (najveci_element_na_itom_nivou(koren, i) == NULL)
254         printf("Nema elemenata na %d. nivou!\n", i);
    else
256         printf("Maksimalni element na %d. nivou: %d\n", i,
            najveci_element_na_itom_nivou(koren, i)->broj);
258 }
```

```
260 printf("Zbir elemenata na %d. nivou: %d\n", i,  
        zbir_cvorova_na_itom_nivou(koren, i));  
262 printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,  
        zbir_manjih_od_x(koren, x));  
264  
/* Oslobadja se memorija zauzeta stablom */  
266 oslobodi_stablo(&koren);  
  
268 return 0;  
}
```

### Rešenje 4.23

```
#include <stdio.h>  
2 #include <stdlib.h>  
  
4 /* Ukljucuje se biblioteka za rad sa stablima */  
#include "stabla.h"  
6  
/* Funkcija koja izracunava dubinu stabla */  
8 int dubina_stabla(Cvor * koren)  
{  
10     /* Dubina praznog stabla je 0 */  
    if (koren == NULL)  
12         return 0;  
  
14     /* Izracunava se dubina levog podstabla */  
    int dubina_levo = dubina_stabla(koren->levo);  
16  
    /* Izracunava se dubina desnog podstabla */  
18     int dubina_desno = dubina_stabla(koren->desno);  
  
20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje  
        jer se racuna i koren */  
22     return dubina_levo >  
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;  
24 }  
  
26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */  
void ispisi_nivo(Cvor * koren, int i)  
28 {  
    /* Ideja je slicna ideji iz prethodne funkcije */  
30     /* Nema vise cvorova, nema spustanja niz stablo */  
    if (koren == NULL)  
32         return;  
  
34     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */  
    if (i == 0) {  
36         printf("%d ", koren->broj);  
        return;  
    }
```

```

38     }
    /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
40     podstablu */
    ispisi_nivo(koren->levo, i - 1);
42     ispisi_nivo(koren->desno, i - 1);
    }

44     /* Funkcija koja ispisuje stablo po nivoima */
46     void ispisi_stablo_po_nivoima(Cvor * koren)
    {
48         int i;

50         /* Prvo se izracunava dubina stabla */
        int dubina;
52         dubina = dubina_stabla(koren);

54         /* Ispisuje se nivo po nivo stabla */
        for (i = 0; i < dubina; i++) {
56             printf("%d. nivo: ", i);
            ispisi_nivo(koren, i);
58             printf("\n");
        }
60     }

62     int main(int argc, char **argv)
    {
64         Cvor *koren;
        int broj;

66         /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
68         uspesnosti dodavanja */
        koren = NULL;
70         while (scanf("%d", &broj) != EOF) {
            if (dodaj_u_stablo(&koren, broj) == 1) {
72                 fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
                oslobodi_stablo(&koren);
74                 return 0;
            }
76         }

78         /* Ispisuje se stablo po nivoima */
        ispisi_stablo_po_nivoima(koren);

80         /* Oslobadja se memorija zauzeta stablom */
        oslobodi_stablo(&koren);

82         return 0;
84     }
}

```

## Rešenje 4.25

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija koja izracunava dubinu stabla */
8 int dubina_stabla(Cvor * koren)
9 {
10     /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
12         return 0;
13
14     /* Izracunava se dubina levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
16
17     /* Izracunava se dubina desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);
19
20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
21     jer se racuna i koren */
22     return dubina_levo >
23         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }
25
26 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
27 stablo */
28 int avl(Cvor * koren)
29 {
30     int dubina_levo, dubina_desno;
31
32     /* Ako je stablo prazno, zaustavlja se brojanje */
33     if (koren == NULL) {
34         return 0;
35     }
36
37     /* Izracunava se dubina levog podstabla korena */
38     dubina_levo = dubina_stabla(koren->levo);
39
40     /* Izracunava se dubina desnog podstabla korena */
41     dubina_desno = dubina_stabla(koren->desno);
42
43     /* Ako je uslov za AVL stablo ispunjen */
44     if (abs(dubina_desno - dubina_levo) <= 1) {
45         /* Racuna se broj AVL cvorova u levom i desnom podstablu i
46         uvecava za jedan iz razloga sto koren ispunjava uslov */
47         return 1 + avl(koren->levo) + avl(koren->desno);
48     } else {
49         /* Inace, racuna se samo broj AVL cvorova u podstablima */
50         return avl(koren->levo) + avl(koren->desno);
51     }
52 }
```

```

}
53
int main(int argc, char **argv)
55
{
    Cvor *koren;
57    int broj;

59    /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
       uspesnosti dodavanja */
61    koren = NULL;
    while (scanf("%d", &broj) != EOF) {
63        if (dodaj_u_stablo(&koren, broj) == 1) {
            fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
65            oslobodi_stablo(&koren);
            return 0;
67        }
    }
    }

69    /* Racuna se i ispisuje broj AVL cvorova */
71    printf("%d\n", avl(koren));

73    /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);

75    return 0;
77 }

```

#### Rešenje 4.26

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
    #include "stabla.h"

6

8  /* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
   brojeva hip. Ideja koja ce biti implementirana u osnovi ima
   pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
10  desnog podstabla - ako je vrednost u korenu veca od izracunatih
   vrednosti, uoceni fragment stabla zadovoljava uslov za hip. Zato
12  ce funkcija vratiti maksimalne vrednosti iz uocenog podstabala ili
   vrednost -1 ukoliko se zakljuci da stablo nije hip. */
14  int heap(Cvor * koren)
   {
16      int max_levo, max_desno;

18      /* Prazno sablo je hip - kao rezultat se vraća 0 kao najmanji
         pozitivan broj */
20      if (koren == NULL) {
          return 0;
22      }
   }

```

```
/* Ukoliko je stablo list... */
24 if (koren->levo == NULL && koren->desno == NULL) {
    /* Vraca se njegova vrednost */
26     return koren->broj;
    }
28 /* Inace... */

30 /* Proverava se svojstvo za levo podstablo */
max_levo = heap(koren->levo);
32
/* Proverava se svojstvo za desno podstablo */
34 max_desno = heap(koren->desno);

36 /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
    ni celo stablo */
38 if (max_levo == -1 || max_desno == -1) {
    return -1;
40 }

42 /* U suprotnom proverava se da li svojstvo vazii za uoceni cvor */
if (koren->broj > max_levo && koren->broj > max_desno) {
44     /* Ako vazii, vraca se vrednost korena */
    return koren->broj;
46 }

48 /* U suprotnom zakljucuje se da stablo nije hip */
return -1;
50 }

52 int main(int argc, char **argv)
{
54     Cvor *koren;
    int hip_indikator;
56
    /* Kreira se stablo prema zadatoj slici */
58     koren = NULL;
    koren = napravi_cvor(100);
60     koren->levo = napravi_cvor(19);
    koren->levo->levo = napravi_cvor(17);
62     koren->levo->levo->levo = napravi_cvor(2);
    koren->levo->levo->desno = napravi_cvor(7);
64     koren->levo->desno = napravi_cvor(3);
    koren->desno = napravi_cvor(36);
66     koren->desno->levo = napravi_cvor(25);
    koren->desno->desno = napravi_cvor(1);
68

    /* Poziva se funkcija kojom se proverava da li je stablo hip */
70     hip_indikator = heap(koren);

72     /* Ispisuje se rezultat */
    if (hip_indikator == -1) {
74         printf("Zadato stablo nije hip!\n");
    }
```

```
76     } else {  
77         printf("Zadato stablo je hip!\n");  
78     }  
79  
80     /* Oslobadja se memorija zauzeta stablom */  
81     oslobodi_stablo(&koren);  
82  
83     return 0;  
84 }
```





# 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 zatim u narednim linijama po jedna niska ne duža od 50 karaktera. Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom. U slučaju pojave bilo kakve greške na standardnom izlazu za grešku ispisati vrednost  $-1$  i prekinuti izvršavanje programa.

#### *Test 1*

```
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 GREŠKU:
-1
```

### Test 4

```
POKRETANJE: ./a.out
IZLAZ ZA GREŠKU:
-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 sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `sumiraj_n` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za greške ispisati  $-1$ .

NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka [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 kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost  $-1$  na standardni izlaz za greške.

<p><i>Test 1</i></p> <pre> ULAZ: 4 5 1 2 3 4 5 -1 2 -3 4 -5 -5 -4 -3 -2 1 -1 0 0 0 0 IZLAZ: 0 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 2 3 0 0 -5 1 2 -4 IZLAZ: 2 </pre>	<p><i>Test 3</i></p> <pre> ULAZ: -2 IZLAZ ZA GREŠKU: -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.

<p><i>Test 1</i></p> <pre> POKRETANJE: ./a.out ulaz.txt test  ULAZ.TXT Ovo je test primer. U njemu se rec test javlja vise puta. testtesttest  IZLAZ: 5 </pre>	<p><i>Test 2</i></p> <pre> POKRETANJE: ./a.out  IZLAZ ZA GREŠKU: -1 </pre>
<p><i>Test 3</i></p> <pre> POKRETANJE: ./a.out ulaz.txt foo  DATOTEKA ULAZ.TXT NE POSTOJI  IZLAZ ZA GREŠKU: -1 </pre>	<p><i>Test 4</i></p> <pre> 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 učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati  $-1$  na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

*Test 1*

```

TROUGLOVI.TXT
4
  0 0 0 1.2 1 0
  0.3 0.3 0.5 0.5 0.9 1
-2 0 0 0 0 1
-2 0 0 0 0 1

IZLAZ:
2 0 2 2 -1 -1
-2 0 0 0 0 1
0 0 0 1.2 1 0
0.3 0.3 0.5 0.5 0.9 1
    
```

*Test 2*

```

TROUGLOVI.TXT
3
  1.2 3.2 1.1 4.3

IZLAZ ZA GREŠKU:
-1
    
```

*Test 3*

```

DATOTEKA TROUGLOVI.TXT NE POSTOJI

IZLAZ ZA GREŠKU:
-1
    
```

*Test 4*

```

TROUGLOVI.TXT
0

IZLAZ:
    
```

[Rešenje 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.

<p><i>Test 1</i></p> <pre>    ULAZ:   1 5 3 6 1 4 7 9   IZLAZ:   1 </pre>	<p><i>Test 2</i></p> <pre>    ULAZ:   2 5 3 6 1 0 4 7 9   IZLAZ:   2 </pre>	<p><i>Test 3</i></p> <pre>    ULAZ:   0 4 2 5   IZLAZ:   0 </pre>
<p><i>Test 4</i></p> <pre>    ULAZ:   3   IZLAZ:   0 </pre>	<p><i>Test 5</i></p> <pre>    ULAZ:   -1 4 5 1 7   IZLAZ:   0 </pre>	

[Rešenje A.6]

### A.3 Praktični deo ispita, septembar 2015.

[illegible]

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 6 1	ULAZ: 15 3	ULAZ: 31 100
IZLAZ: 3	IZLAZ: 3758096385	IZLAZ: 4026531841

<i>Test 4</i>	<i>Test 5</i>
ULAZ: 4 0	ULAZ: 0 5
IZLAZ: 4	IZLAZ: 0

[Rešenje A.7]

**Zadatak A.8** Data je biblioteka za rad sa listama. Napisati funkciju `int dopuni_listu(Cvor ** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog

sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Povratna vrednost funkcije treba da bude 1 ukoliko je došlo greške pri alokaciji memorije, inače 0. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

### Test 1

```

|| ULAZ:
|| 1 2 3 4 5
|| IZLAZ:
|| 1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00

```

### Test 2

```

|| ULAZ:
|| 12
|| IZLAZ:
|| 12.00

```

### Test 3

```

|| ULAZ:
|| prazna lista
|| IZLAZ:

```

### Test 4

```

|| ULAZ:
|| 13.3 15.8
|| IZLAZ:
|| 13.30 14.55

```

[Rešenje [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 standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati "-". Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati -1 na standardni izlaz za greške. NAPOMENA: *Koristiti biblioteku za rad sa matricama iz zadatka [2.19](#).*

### Test 1

```

|| ULAZ:
|| 4
|| 1 2 3 4
|| 2 1 4 3
|| 3 4 2 1
|| 4 3 1 2
|| IZLAZ:
|| 10

```

### Test 2

```

|| ULAZ:
|| 3
|| 1 1 1
|| 1 1 1
|| 1 1 1
|| IZLAZ:
|| 3

```

### Test 3

```

|| ULAZ:
|| 2
|| 1 1
|| 2 2
|| IZLAZ:
|| -

```

<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 GREŠKU:   -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 GREŠKU:   -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 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:
  matrica.txt 7

MATRICA.TXT
  3 2
  1.1 -5.31
  -3.7 35.24
  1.4 2.09

IZLAZ:
  0

```

## Test 4

```

ULAZ:
  brojevi.txt 12

DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ ZA GREŠKU:
  -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

6 /* Funkcija vrši dinamičku alokaciju memorije potrebne n linija tj.
   n niski 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 stderr i
       program završava. */
16     if (linije == NULL)
17         return NULL;
18     /* Alocira se prostor za svaku vrstu matrice. Niska nije duža od
       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
           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);

```

```
        return NULL;
30    }
    }
32    return linije;
}

34    /* Funkcija oslobadjaja dinamicki alociranu memoriju */
36    char **oslobadjanje_memorije(char **linije, int n)
    {
38        int i;
        /* Oslobadja se prostor rezervisan za svaku vrstu */
40        for (i = 0; i < n; i++) {
            free(linije[i]);
42        }
        /* Oslobadja se memorija za niz pokazivaca na vrste */
44        free(linije);

46        /* Matrica postaje prazna, tj. nealocirana */
        return NULL;
48    }

50    int main(int argc, char *argv[])
    {
52        FILE *ulaz;
        char **linije;
54        int i, j, n;

56        /* Proverava argumenata komandne linije. */
        if (argc != 2) {
58            fprintf(stderr, "-1\n");
            exit(EXIT_FAILURE);
60        }

62        /* Otvaranje datoteke cije ime je navedeno kao argument komandne
            linije neposredno nakon imena programa koji se poziva. U slucaju
64            neuspesnog otvaranja ispisuje se -1 na stderr i program završava
            izvršavanje. */
66        ulaz = fopen(argv[1], "r");
        if (ulaz == NULL) {
68            fprintf(stderr, "-1\n");
            exit(EXIT_FAILURE);
70        }
        /* Ucitavanje broja linija. */
72        fscanf(ulaz, "%d", &n);

74        /* Alociranje memorije na osnovu ucitanog broja linija. */
        linije = alociranje_memorije(n);
76

        /* U slucaju neuspesne alokacije ispisuje se -1 na stderr i program
78            završava. */
        if (linije == NULL) {
80            fprintf(stderr, "-1\n");
```

```

    exit(EXIT_FAILURE);
82 }

/* Ucitavanje svih n linija iz datoteke. */
84 for (i = 0; i < n; i++) {
86     fscanf(ulaz, "%s", linije[i]);
88 }

/* Ispisivanje u odgovarajucem poretку ucitane linije koje
90 zadovoljavaju kriterijum. */
92 for (i = n - 1; i >= 0; i--) {
94     if (isupper(linije[i][0])) {
96         printf("%s\n", linije[i]);
98     }
100 }

/* Oslobadjanje memorije koja je dinamički alocirana. */
102 linije = oslobadjanje_memorije(linije, n);

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

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;
    /* Inace ... */
10     /* Ako je n jednako nula, vraca 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 vraca 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
int main()
22 {
    Cvor *koren = NULL;
    int n;
    int nivo;
26
    /* Ucitava se vrednost nivoa */
    scanf("%d", &nivo);
    while (1) {
28         scanf("%d", &n);
        /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
        if (n == 0)
32             break;
        /* Ako nije, dodaje se procitani broj u stablo. */
        if (dodaj_u_stablo(&koren, n) == 1) {
34             fprintf(stderr, "-1\n", n);
            oslobodi_stablo(&koren);
            exit(EXIT_FAILURE);
36         }
    }
40 }

    /* Ispisuje se rezultat rada trazene funkcije */
    printf("%d\n", sumiraj_n(koren, nivo));
42
    /* Oslobadja se memorija */
    oslobodi_stablo(&koren);
44
    exit(EXIT_SUCCESS);
46
}
48

```

### Rešenje A.3

```

#include <stdio.h>
2 #include <stdlib.h>
#define MAKS 50
4
/* Funkcija ucitava elemente matrice sa standardnog ulaza */
6 void ucitaj_matricu(int m[][MAKS], int v, int k)
{
    int i, j;
    for (i = 0; i < v; i++) {
10         for (j = 0; j < k; j++) {
            scanf("%d", &m[i][j]);
12         }
    }
14 }

/* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
16 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 i, 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-toj 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     /* Ucitavanje broja vrsta matrice */
61     scanf("%d", &v);
62
63     /* Provera validnosti broja vrsta */
64     if (v < 0 || v > MAKS) {
65         fprintf(stderr, "-1\n");
66         exit(EXIT_FAILURE);
67     }
68
69     /* Ucitavanje broja kolona matrice */

```

```
70     scanf("%d", &k);

72     /* Provera validnosti broja kolona */
73     if (k < 0 || k > MAKS) {
74         fprintf(stderr, "-1\n");
75         exit(EXIT_FAILURE);
76     }
77     /* Ucitavanje elemenata matrice */
78     ucitaj_matricu(m, v, k);

80     /* Pronalazenje kolone koja sadrzi najveći broj negativnih
81        elemenata i ispisivanje traženog rezultata */
82     printf("%d\n", maks_indeks(m, v, k));

84     exit(EXIT_SUCCESS);
85 }
```

### Rešenje A.4

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define MAKS 128

6 int main(int argc, char **argv)
7 {
8     FILE *f;
9     int brojac = 0;
10    char linija[MAKS], *p;

12    /* Provera da li je broj argumenata komandne linije 3 */
13    if (argc != 3) {
14        fprintf(stderr, "-1\n");
15        exit(EXIT_FAILURE);
16    }
17    /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18       linije */
19    if ((f = fopen(argv[1], "r")) == NULL) {
20        fprintf(stderr, "-1\n");
21        exit(EXIT_FAILURE);
22    }
23    /* Ucitavanje iz otvorene datoteke - liniju po liniju */
24    while (fgets(linija, MAKS, f) != NULL) {
25        p = linija;
26        while (1) {
27            p = strstr(p, argv[2]);
28
29            /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
30            if (p == NULL)
31                break;
32            /* Inace se uvecava brojac */

```

```

    brojac++;
34    /* p se pomera da bi se u sledecoj iteraciji posmatra ostatak
        linije nakon uocene podniske */
36    p = p + strlen(argv[2]);
    }
38 }

40 /* Zatvaranje datoteke */
fclose(f);

42 /* Ispisivanje vrednosti brojaca */
44 printf("%d\n", brojac);

46 exit(EXIT_SUCCESS);
}

```

### Rešenje A.5

```

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

5  /* Struktura trougao */
   typedef struct _trougao {
7      double xa, ya, xb, yb, xc, yc;
   } trougao;

9

11 /* Funkcija racuna duzinu duzi */
   double duzina(double x1, double y1, double x2, double y2)
   {
13     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
   }

15

17 /* Funkcija racuna povrsinu trougla koristeći Heronov obrazac */
   double povrsina(trougao t)
   {
19     /* Racunanje duzina stranica trougla */
       double a = duzina(t.xb, t.yb, t.xc, t.yc);
21     double b = duzina(t.xa, t.ya, t.xc, t.yc);
       double c = duzina(t.xa, t.ya, t.xb, t.yb);
23     /* Poluobim */
       double s = (a + b + c) / 2;
25     /* Primena Heronovog obrasca */
       return sqrt(s * (s - a) * (s - b) * (s - c));
27 }

29 /* Funkcija poredi dva trougla: ukoliko je povrsina trougla koji je
   prvi argument funkcije manja od povrsine trougla koji je drugi
31 element funkcije funkcija vraca 1, ukoliko je veca -1, a ukoliko
   su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
33 napisana tako da se moze proslediti funkciji qsort da se niz

```

```
    trouglova sortira po površini opadajuće. */
35 int poredi(const void *a, const void *b)
{
37     trougao x = *(trougao *) a;
    trougao y = *(trougao *) b;
39     double xp = površina(x);
    double yp = površina(y);
41     if (xp < yp)
        return 1;
43     if (xp > yp)
        return -1;
45     return 0;
}

47
49 int main()
{
    FILE *f;
51     int n, i;
    trougao *niz;

53     /* Otvaranje datoteke čiji je naziv trouglovi.txt */
55     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
        fprintf(stderr, "-1\n");
57         exit(EXIT_FAILURE);
    }

59     /* Učitavanje podataka o broju trouglova iz datoteke */
61     if (fscanf(f, "%d", &n) != 1) {
        fprintf(stderr, "-1\n");
63         exit(EXIT_FAILURE);
    }

65     /* Dinamička alokacija memorije za niz trouglova dužine n */
67     if ((niz = malloc(n * sizeof(trougao))) == NULL) {
        fprintf(stderr, "-1\n");
69         exit(EXIT_FAILURE);
    }

71     /* Učitavanje podataka u niz iz otvorene datoteke */
73     for (i = 0; i < n; i++) {
        if (fscanf(f, "%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
75             &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
        {
            fprintf(stderr, "-1\n");
77             exit(EXIT_FAILURE);
        }
79     }

81     /* Pozivanje funkcije qsort da sortira niz na osnovu funkcije
        poredi */
83     qsort(niz, n, sizeof(trougao), &poredi);
```



```

85  /* Ispisivanje sortiranog niza na standardni izlaz */
    for (i = 0; i < n; i++)
87      printf("%g %g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
              niz[i].yb, niz[i].xc, niz[i].yc);

89  /* Oslobađanje dinamički alocirane memorije */
91  free(niz);

93  /* Zatvranje datoteke */
    fclose(f);

95  exit(EXIT_SUCCESS);
97 }

```

## 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 učitava brojeve sa standardnog ulaza i smesta ih u
6  stablo. Funkcija vraća 1 u slučaju neuspješnog dodavanja elementa u
   stablo, a inače 0. */
8  int učitaj_stablo(Cvor ** koren)
{
10     *koren = NULL;
    int x;
12     /* Sve dok ima brojeva na standardnom ulazu, učitani brojevi se
       dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
       je i povratna vrednost iz funkcije učitaj_stablo 1. */
14     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. Takođe, ako je n
       negativan broj, na tom nivou nema cvorova (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 vraća 1, inače 0 */

```

```

30  if (n == 0) {
31      if (koren->levo == NULL && koren->desno != NULL)
32          return 1;
33      if (koren->levo != NULL && koren->desno == NULL)
34          return 1;
35      return 0;
36  }
37  /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
38     (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om nivou
39     levog podstabla */
40  return prebroj_n(koren->levo, n - 1) + prebroj_n(koren->desno, n -
41      1);
42  }
43
44  int main()
45  {
46      Cvor *koren;
47      int n;
48      scanf("%d", &n);
49
50      /* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
51         oslobodja se alocirana memorija i izlazi se iz programa. */
52      if (ucitaj_stablo(&koren) == 1) {
53          fprintf(stderr, "-1\n");
54          oslobodi_stablo(&koren);
55          exit(EXIT_FAILURE);
56      }
57
58      /* Ispisivanje rezultata */
59      printf("%d\n", prebroj_n(koren, n));
60
61      /* Oslobadjanje dinamicki alociranog stabla */
62      oslobodi_stablo(&koren);
63
64      exit(EXIT_SUCCESS);
65  }

```

## Rešenje A.7

```

1  #include <stdio.h>
2
3  /* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
4     zapis argumenta x rotira za n mesta udesno */
5  unsigned int rotiraj(unsigned int x, unsigned int n)
6  {
7      int i;
8      unsigned int maska = 1;
9      /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
10         izgledala: 000...00001111 */
11      /* Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
12         nastavku je drugi nacin. */

```

```

14     for (i = 1; i < n; i++)
        maska = (maska << 1) | 1;
16     /* Kada se x poremeri za n mesta udesno x >> n, poslednjih n bitova
        binarne reprezentacije broja x ce "ispasti". Za rotaciju je
        potrebno da se tih n bitova postavi na pocetak broja. Kreirana
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 logicke disjunkcije
        dobija se rotirani broj. */
22     return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
    }

24 int main()
26 {
    unsigned int x, n;

28     /* Ucitavanje brojeva sa standardnog ulaza */
30     scanf("%u%u", &x, &n);

32     /* Ispisivanje rezultata */
    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. */
12 Cvor * napravi_cvor(double broj);

14 /* Funkcija oslobadja dinamicnu memoriju zauzetu za elemente
    liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
16 void oslobodi_listu(Cvor ** adresa_glave);

18 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL kao je lista prazna */
20 Cvor * pronadji_poslednji(Cvor * glava);

22 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo

```

```
    greske pri alokaciji memorije, inace vraca 0. */
24 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);

26 /* Funkcija prikazuje sve elemente liste pocev od glave ka kraju
    liste. */
28 void ispisi_listu(Cvor * glava);
30 #endif
```

*liste.c*

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

4
/* Pomocna funkcija koja kreira cvor. */
6 Cvor *napravi_cvor(double broj)
{
8     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
10         return NULL;
    /* inicijalizacija polja u novom cvoru */
12     novi->vrednost = broj;
    novi->sledeci = NULL;
14     return novi;
16 }

18 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
    ciji se pocetni cvor nalazi na adresi adresa_glave. */
20 void oslobodi_listu(Cvor ** adresa_glave)
{
22     Cvor *pomocni = NULL;
    while (*adresa_glave != NULL) {
24         pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
26         *adresa_glave = pomocni;
28     }
30 }

32 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL kao je lista prazna */
34 Cvor *pronadji_poslednji(Cvor * glava)
{
36     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
        funkcija vraca NULL. */
38     if (glava == NULL)
        return NULL;
40     while (glava->sledeci != NULL)
```

```

    glava = glava->sledeci;
42  return glava;

44  }

46  /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
    greske pri alokaciji memorije, inace vraca 0. */
48  int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
    {
50      Cvor *novi = napravi_cvor(broj);
        if (novi == NULL)
52          return 1;
        if (*adresa_glave == NULL) {
54            *adresa_glave = novi;
            return 0;
56        }
58        Cvor *poslednji = pronadji_poslednji(*adresa_glave);
        poslednji->sledeci = novi;
60    }

62    /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
        */
64    void ispisi_listu(Cvor * glava)
        {
66            for (; glava != NULL; glava = glava->sledeci)
                printf("%.2lf ", glava->vrednost);
68            putchar('\n');
70        }

```

main.c

```

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

4

/* Funkcija umece novi cvor iza tekuceg u listi */
6  void dodaj_iza(Cvor * tekuci, Cvor * novi)
    {
8      /* Novi cvor se dodaje iza tekuceg cvora. */
        novi->sledeci = tekuci->sledeci;
10     tekuci->sledeci = novi;
    }

12

14  /* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka.
    Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace vraca
16  0. */
    int dopuni_listu(Cvor ** adresa_glave)

```

```
18 {
19     Cvor *tekuci;
20     Cvor *novi;
21     double aritmeticka_sredina;
22     /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
23     if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
24         return 1;
25     /* Promenljiva tekuci se inicijalizacije da pokazuje na pocetni
26        cvor */
27     tekuci = *adresa_glave;
28     /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
29        vrednosti u susednim cvorovima i kreira cvor sa tom vrednoscu. U
30        slucaju neupele alokacije novog cvora, funkcija vraca 1. Inace,
31        novi cvor se umece izmedju dva cvora za koje racunata
32        aritmeticka sredina */
33     while (tekuci->sledeci != NULL) {
34         aritmeticka_sredina =
35             ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
36         novi = napravi_cvor(aritmeticka_sredina);
37         if (novi == NULL)
38             return 1;
39         /* Poziva se funkcija koja umece novi cvor iza tekućeg cvora */
40         dodaj_iza(tekuci, novi);
41         /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
42            cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz
43            polazne liste */
44         tekuci = tekuci->sledeci;
45         tekuci = tekuci->sledeci;
46     }
47     return 0;
48 }
49
50 int main()
51 {
52     Cvor *glava = NULL;
53     double broj;
54
55     /* Ucitavanje se vrsi do kraja ulaza. Elementi se dodaju na kraj
56        liste! */
57     while (scanf("%lf", &broj) > 0) {
58         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
59            memorije za nov cvor. Memoriju alociranu za cvorove liste
60            treba osloboditi. */
61         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
62             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
63             oslobodi_listu(&glava);
64             exit(EXIT_FAILURE);
65         }
66     }
67
68     /* Pozivanje funkcije da dopuni listu. Ako je funkcija vratila 1,
69        onda je bilo greske pri alokaciji memorije za nov cvor. Memoriju
```

```

70     alociranu za cvorove liste treba osloboditi. */
71     if (dopuni_listu(&glava) == 1) {
72         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
73         oslobodi_listu(&glava);
74         exit(EXIT_FAILURE);
75     }
76
77     /* Ispisivanje liste */
78     ispisi_listu(glava);
79
80     /* Oslobadjanje liste */
81     oslobodi_listu(&glava);
82
83     exit(EXIT_SUCCESS);
84 }

```

### Rešenje A.9

```

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

/* Funkcija racuna zbir elemenata v-te vrste */
6 int zbir_vrste(int **M, int n, int v)
{
8     int j, zbir = 0;
9     for (j = 0; j < n; j++)
10         zbir += M[v][j];
11     return zbir;
12 }

/* Funkcija racuna zbir elemenata k-te kolone */
14 int zbir_kolone(int **M, int n, int k)
15 {
16     int i, zbir = 0;
17     for (i = 0; i < n; i++)
18         zbir += M[i][k];
19     return zbir;
20 }

/* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
21 argument magican. Ukoliko jeste magican funkcija vraca 1, inace 0.
22 Argument funkcije zbir ce sadrzati zbir elemenata neke vrste ili
23 kolone ukoliko je kvadrat magican. */
24 int magicni_kvadrat(int **M, int n, int *zbir_magicnog)
25 {
26     int i, j;
27     int zbir = 0, zbir_pom;
28     /* Promenljivu zbir inicijalizujemo na zbir 0-te vrese */
29     zbir = zbir_vrste(M, n, 0);
30
31     for (j = 0; j < n; j++)
32         zbir_pom = zbir_kolone(M, n, j);
33     if (zbir == zbir_pom)
34         return 1;
35     return 0;
36 }

```

```
34  /* Racunaju se zbrovi u ostalim vrstama i ako neki razlikuje od
    vrednosti promeljive zbir funkcija vraca 1 */
36  for (i = 1; i < n; i++) {
    zbir_pom = zbir_vrste(M, n, i);
38      if (zbir_pom != zbir)
        return 0;
40  }
    /* Racunaju se zbrovi u svim kolonama i ako neki razlikuje od
    vrednosti promeljive zbir funkcija vraca 1 */
42  for (j = 0; j < n; j++) {
    zbir_pom = zbir_kolone(M, n, j);
44      if (zbir_pom != zbir)
        return 0;
46  }
48  /* Inace su zbrovi svih vrsta i kolona jednaki, postavlja se
    vresnost u zbir_magicnog i funkcija vraca 1 */
50  *zbir_magicnog = zbir;
    return 1;
52 }

54 int main()
{
56     int n, i, j;
    int **matrica = NULL;
58     int zbir = 0, zbir_magicnog = 0;
    scanf("%d", &n);
60
    /* Provera da li je n strogo pozitivan */
62     if (n <= 0) {
        printf("-1\n");
        exit(EXIT_FAILURE);
64     }
66
    /* Dinamicka alokacija matrice dimenzije nxn */
68     matrica = alociraj_matricu(n);
    if (matrica == NULL) {
70         printf("-1\n");
        exit(EXIT_FAILURE);
72     }

74     /* Ucitavanje elemenata matrice sa standardnog ulaza */
    ucitaj_matricu(matrica, n);
76
    /* Ispisivanje rezultata na osnovu fukcije magicni_kvadrat */
78     if (magicni_kvadrat(matrica, n, &zbir_magicnog)) {
        printf("%d\n", zbir_magicnog);
80     } else
        printf("-\n");
82
    /* Oslobadjanje dinamicki alocirane memorije */
84     matrica = dealociraj_matricu(matrica, n);
```



```

86     exit(EXIT_SUCCESS);
    }

```

### Rešenje A.10

```

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

#define BITOVA_U_BAJTU 8

/* Funkcija u datom broju x menja mesta prvom i četvrtom bajtu */
unsigned int zamena(unsigned int x){
    /* Deklaracija promenljivih za odgovarajuće maske i pomoćne
       promenljive*/
    unsigned maska_prvi_bajt, maska_cetvrti_bajt;
    unsigned maska_prvi_bajt_komplement, maska_cetvrti_bajt_komplement;
    unsigned prvi_bajt, cetvrti_bajt;
    unsigned i;

    /* Maska prvi bajt odgovara broju čija je binarna reprezentacija
       00000...000001111111 (8 bitova najmanje težine su jedinice,
       a ostalo su nule) */
    maska_prvi_bajt = 1;
    for(i=1; i<BITOVA_U_BAJTU; i++)
        maska_prvi_bajt = maska_prvi_bajt<<1|1;

    /* Maska četvrti bajt odgovara broju čija je binarna reprezentacija
       111111100000...00000 (8 bitova najveće težine su jedinice,
       a ostalo su nule) */
    maska_cetvrti_bajt = maska_prvi_bajt<<(((sizeof(unsigned)-1)*
        BITOVA_U_BAJTU));

    /* Primenom operatora ~ na maska_prvi_bajt dobija se broj čija je
       binarna reprezentacija 11111...1111100000000 (8 bitova najmanje
       težine su nule, a ostalo su jedinice) */
    maska_prvi_bajt_komplement = ~ maska_prvi_bajt;
    /* Primenom operatora ~ na maska_prvi_bajt dobija se broj čija je
       binarna reprezentacija 0000000011111...11111 (8 bitova najveće
       težine su nule, a ostalo su jedinice) */
    maska_cetvrti_bajt_komplement = ~ maska_cetvrti_bajt;

    /* U promenljivu prvi_bajt smestamo broj koji se dobija kada se
       bitovi prvog bajta broja x pomere ulevo, tako da budu na
       poziciji četvrtog bajta */
    prvi_bajt = (maska_prvi_bajt&x)<<(((sizeof(unsigned)-1)*
        BITOVA_U_BAJTU));
    /* U promenljivu cetvrti_bajt smestamo broj koji se dobija kada se
       bitovi četvrtog bajta broja x pomere udesno, tako da budu na
       poziciji prvog bajta */
    cetvrti_bajt = (maska_cetvrti_bajt&x)>>(((sizeof(unsigned)-1)*
        BITOVA_U_BAJTU));

```

```

44  /* Na nule se postavlja 8 bitova najmanje tezine, a ostali bitovi
46      ostaju nepromenjeni */
    x = x&maska_prvi_bajt_komplement;

48  /* Na nule se postavlja 8 bitova najvece tezine, a ostali bitovi
50      ostaju nepromenjeni */
    x = x&maska_cetvrti_bajt_komplement;

52  /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi
54      iz prvog bajta */
    x = x|prvi_bajt;

56  /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi
58      iz prvog bajta */
    x = x|cetvrti_bajt;

60
62  return x;
63  }

64  int main () {
65      int x;

66
67      /* Sa standardnog ulaza se ucitava ceo broj */
68      scanf("%d", &x);

69
70      /* Provera da li je uneti broj negativan */
71      if(x<0){
72          fprintf(stderr, "-1\n");
73          exit(EXIT_FAILURE);
74      }

75
76      /* Ispisivanje rezultata primene funkcije zamena na uneti broj x */
77      printf("%u\n", zamena(x));

78      exit(EXIT_SUCCESS);
79  }
80  }

```

### 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 "stablo.h"
4
5  /* Funkcija racuna duzinu najduzeg puta od korena do nekog lista */
6  int najduzi_put(Cvor *koren) {
7      /* Pomocne promenljive */
8      int najduzi_u_levom, najduzi_u_desnom;

```

```

10  /* Ako je stablo prazno, povratna vrednost je -1 */
    if(koren==NULL)
12      return -1;

14  /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
    najduzi_u_levom=najduzi_put(koren->levo);

16  /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
    najduzi_u_desnom=najduzi_put(koren->desno);

18  /* Veka od prethodno izracunatih vrednosti za podstabla se uvecava
    za 1 i vraca kao konacan rezultat */
22  return 1 + (najduzi_u_levom > najduzi_u_desnom ? najduzi_u_levom :
    najduzi_u_desnom);
}

24  int main() {
26    Cvor *stablo = NULL;
    int x;

28    /* U svakoj iteraciji se procitani broj dodaje u stablo. */
30    while (scanf("%d", &x) == 1)
        if (dodaj_u_stablo(&stablo, x) == 1) {
32        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
34    }

36    /* Ispisuje se rezultat rada trazene funkcije */
    printf("%d\n", najduzi_put(stablo));

38    /* Oslobadja se memorija */
40    oslobodi_stablo(&stablo);

42    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  /* Funkcija alocira memorijski prostor za matricu sa n vrsta i m
   kolona */
7  float **alociraj_matricu(int n, int m)
9  {
    float **matrica = NULL;
11    int i, j;

13    /* Alocira se prostor za niz vrsta matrice */

```

```
matrica = (float **) malloc(n * sizeof(float *));
15 /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije ce
    biti NULL, sto mora biti provereno u main funkciji */
17 if (matrica == NULL)
    return NULL;
19
/* Alocira se prostor za svaku vrstu matrice */
21 for (i = 0; i < n; i++) {
    matrica[i] = (float *) malloc(m * sizeof(float));
23 /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
    prethodno alocirani resursi, i povratna vrednost je NULL */
25 if (matrica[i] == NULL) {
    for (j = 0; j < i; j++)
27         free(matrica[j]);
    free(matrica);
29     return NULL;
    }
31 }
return matrica;
33 }

/* Funkcija oslobadja alocirani memorijski prostor */
35 float **dealociraj_matricu(float **matrica, int n)
37 {
    int i;
39 /* Oslobadja se prostor rezervisan za svaku vrstu */
    for (i = 0; i < n; i++)
41         free(matrica[i]);

43 /* Oslobadja se memorija za niz pokazivaca na vrste */
    free(matrica);
45
47 /* Matrica postaje prazna, tj. nealocirana */
    return NULL;
49 }

/* Funkcija prebrojava koliko se puta pojavljuje broj x u i-toj
51 vrsti matrice A, gde je m broj elemenata u vrsti */
int prebroj_u_itoj_vrsti(float **A, int i, int m, int x){
53     int j;
    int broj = 0;
55     for(j = 0; j < m; j++){
        if(A[i][j] == x)
57             broj++;
    }
59     return broj;
61 }

/* Funkcija vraca indeks vrste matrice A u kojoj se realan broj x
63 pojavljuje najmanje puta */
int indeks_vrste(float x, float **A, int n, int m){
65 /* Indeks vrste sa minimalnim brojem pojavljivanja broja x */
```

```
int min;
67 /* Broj pojavljivanja broja x u vrsti sa indeksom min */
int min_broj;
69 /* Promenljiva u kojoj ce se racunati broj pojavljivanja broja x u
tekucnoj vrsti */
71 int broj_u_vrsti;
/* Pomocne promenljive */
73 int i,j;

75 /* Promenljiva min se inicijalizuje na nulu, a min_broj na broj
pojavljivanja broja x u nultoj vrsti */
77 min=0;
min_broj = prebroj_u_itoj_vrsti(A, 0, m, x);
79

/* Za svaku vrstu (osim nulte) se racuna broj pojavljivanja broja
81 x u njoj, pa ukoliko je taj broj manji od trenutno najmanjeg
azuriraju se promenljive min i min_broj */
83 for(i=1;i<n;i++){
broj_u_vrsti = prebroj_u_itoj_vrsti(A, i, m, x);
85     if(broj_u_vrsti<min_broj){
min_broj=broj_u_vrsti;
87     min=i;
    }
89 }

91 /* Funkcija vraca odgovarajuci indeks vrste */
return min;
93 }

95 int main () {
FILE *in;
97 char datoteka[MAX];
float broj;
99 float **A=NULL;
int i,j;
101 int m,n;

103 /* Sa standardnog ulaza se ucitava ime datoteke i realan broj*/
scanf("%s",datoteka);
105 scanf("%f",&broj);

107 /* Otvaranje datoteke za citanje */
in=fopen(datoteka,"r");
109

/* Provera da li je datoteka uspesno otvorena */
111 if(in==NULL){
fprintf(stderr,"-1\n");
113     exit(EXIT_FAILURE);
}
115

/* Dimenzije matrice se ucitavaju iz datoteke (prva dva cela broja
117 u datoteci). U slucaju neuspesnog ucitavanja, na standardni
```

```
119     izlaz za greske se ispisuje -1 i prekida se program. */
120     if(fscanf(in, "%d %d", &n, &m) == EOF){
121         fprintf(stderr, "-1\n");
122         exit(EXIT_FAILURE);
123     }
124
125     /* Provera da li su učitani brojevi m i n pozitivni */
126     if(n <= 0 || m <= 0){
127         fprintf(stderr, "-1\n");
128         exit(EXIT_FAILURE);
129     }
130
131     /* Alokacija matrice */
132     A = alociraj_matricu(n, m);
133
134     /* Provera da li je alokacija uspela */
135     if(A == NULL){
136         fprintf(stderr, "-1\n");
137         exit(EXIT_FAILURE);
138     }
139
140     /* Učitavanje elemenata matrice iz datoteke */
141     for(i=0; i<n; i++){
142         for(j=0; j<m; j++){
143             fscanf(in, "%f", &A[i][j]);
144         }
145     }
146
147     /* Zatvaranje datoteke */
148     fclose(in);
149
150     /* Ispisivanje rezultata poziva funkcije */
151     printf("%d\n", indeks_vrste(broj, A, n, m));
152
153     /* Dealokacija matrice */
154     A = dealociraj_matricu(A, n);
155     exit(EXIT_SUCCESS);
156 }
```