

Univerzitet u Beogradu  
Matematički fakultet

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

**PROGRAMIRANJE 2**  
**Zbirka zadataka sa rešenjima**

**Beograd  
2015.**

Autori:

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

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

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Anđelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu

Studentski trg 16, 11000 Beograd

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

Recenzenti:

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

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

Obrada teksta, crteži i korice: *autori*

ISBN XXX-XX-XXXX-XXX-X

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

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



# Predgovor

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

Autori velikog broja zadataka ove zbirke su ujedno i autori same zbirke, ali postoje i zadaci za koje se ne može tačno utvrditi ko je originalni autor jer su zadacima davali svoje doprinose različiti asistenti koji su držali vežbe iz ovog kursa u prethodnih desetak godina, pomenimo tu, pre svega, Milana Bankovića i doc dr Filipa Marića. Zbog toga smatramo da je naš osnovni doprinos što smo objedinili, precizno formulisali i rešili sve najvažnije zadatke koji su potrebni za uspešno savlađivanje koncepata koji se obrađuju u okviru kursa.

...

*Autori*



# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>3</b>
1.1	Podela koda po datotekama . . . . .	3
1.2	Algoritmi za rad sa bitovima . . . . .	6
1.3	Rekurzija . . . . .	9
1.4	Rešenja . . . . .	15
<b>2</b>	<b>Pokazivači</b>	<b>49</b>
2.1	Pokazivačka aritmetika . . . . .	49
2.2	Višedimenzioni nizovi . . . . .	51
2.3	Dinamička alokacija memorije . . . . .	54
2.4	Pokazivači na funkcije . . . . .	59
2.5	Rešenja . . . . .	60
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>89</b>
3.1	Algoritmi pretrage . . . . .	89
3.2	Algoritmi sortiranja . . . . .	92
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	99
3.4	Rešenja . . . . .	102
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>153</b>
4.1	Liste . . . . .	153
4.2	Stabla . . . . .	160
4.3	Rešenja . . . . .	166
<b>5</b>	<b>Ispitni rokovi</b>	<b>233</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	233
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	234
5.3	Programiranje 2, praktični deo ispita, septembar 2015. . . . .	235
5.4	Rešenja . . . . .	236



# Glava 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

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

- (a) Definirati strukturu `KompleksanBroj` koja opisuje kompleksan broj njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `void ucitaj_kompleksan_broj(KompleksanBroj * z)` koja učitava kompleksan broj sa standardnog ulaza.
- (c) Napisati funkciju `void ispisi_kompleksan_broj(KompleksanBroj z)` koja ispisuje kompleksan broj na standardni izlaz u odgovarajućem formatu (npr. broj čiji je realan deo 2, a imaginarni -3 ispisati kao  $(2 - 3i)$  na standardni izlaz).
- (d) Napisati funkciju `float realan_deo(KompleksanBroj z)` koja vraća vrednost realnog dela broja.
- (e) Napisati funkciju `float imaginaran_deo(KompleksanBroj z)` koja vraća vrednost imaginarnog dela broja.
- (f) Napisati funkciju `float moduo(KompleksanBroj z)` koja računa moduo kompleksnog broja.
- (g) Napisati funkciju `KompleksanBroj konjugovan(KompleksanBroj z)` koja računa konjugovano-kompleksni broj svog argumenta  $z$ .
- (h) Napisati funkciju `KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)` koja sabira dva kompleksna broja  $z1$  i  $z2$ .
- (i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja oduzima dva kompleksna broja  $z1$  i  $z2$ .
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja množi dva kompleksna broja  $z1$  i  $z2$ .
- (k) Napisati funkciju `float argument(KompleksanBroj z)` koja računa argument kompleksnog broja  $z$ .

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

*Primer 1*

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

[Rešenje 1.1]

**Zadatak 1.2** Uraditi prethodni zadatak tako da su sve napisane funkcije za rad sa kompleksnim brojevima zajedno sa definicijom strukture `KompleksanBroj` izdvojene u posebnu biblioteku. Test program treba da koristi tu biblioteku da za kompleksan broj unet sa standardnog ulaza ispiše polarni oblik unetog broja.

*Primer 1*

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

[Rešenje 1.2]

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

- (a) Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20. UPUTSTVO: *Struktura sadrži stepen i niz koeficijenata. Redosled navođenja koeficijenata u nizu treba da bude takav da na nultoj poziciji u nizu bude koeficijent uz slobodan član, na prvoj koeficijent uz prvi stepen, itd.*
- (b) Napisati funkciju `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz.
- (c) Napisati funkciju `Polinom ucitaj()` koja učitava polinom sa standardnog ulaza. Za polinom se najpre unosi stepen pa njegovi koeficijenti.
- (d) Napisati funkciju `double izracunaj(const Polinom * p, double x)` za izračunavanje vrednosti polinoma `p` u datoj tački `x` koristeći Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja sabira dva polinoma `p` i `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja množi dva polinoma `p` i `q`.
- (g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja računa izvod polinoma `p`.
- (h) Napisati funkciju `Polinom nIzvod(const Polinom * p, int n)` koja računa `n`-ti izvod polinoma `p`.

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



## Primer 1

```

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

```

[Rešenje 1.3]

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

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

Napisati program koji testira prethodne funkcije tako što se sa standardnog ulaza unose dva razlomka `r1` i `r2` i na standardni izlaz se ispisuju skraćene vrednosti razlomaka koji su dobijeni kao zbir, razlika, proizvod i količnik razlomka `r1` i recipročne vrednosti razlomka `r2`.

## Primer 1

```

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

```



## Test 3

```

ULAZ:
0x00FF00FF
IZLAZ:
Najveci:
111111111111110000000000000000
Najmanji:
000000000000000111111111111111

```

## Test 4

```

ULAZ:
0xFFFFFFFF
IZLAZ:
Najveci:
111111111111111111111111111111
Najmanji:
111111111111111111111111111111

```

[Rešenje 1.7]

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

- Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja  $x$ , počevši od pozicije  $p$ , postave na 0.
- Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja  $x$ , počevši od pozicije  $p$ , postave na 1.
- Napisati funkciju koja određuje broj koji se dobija od  $n$  bitova datog broja  $x$ , počevši od pozicije  $p$ , i vraća ih kao bitove najmanje težine rezultata.
- Napisati funkciju koja vraća broj koji se dobija upisivanjem poslednjih  $n$  bitova broja  $y$  u broj  $x$ , počevši od pozicije  $p$ .
- Napisati funkciju koja vraća broj koji se dobija invertovanjem  $n$  bitova broja  $x$  počevši od pozicije  $p$ .

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve  $x$ ,  $n$ ,  $p$ ,  $y$  koji se unose sa standardnog ulaza. Program treba nakon učitavanja odgovarajućih vrednosti ispiše najpre binarne reprezentacije brojeva  $x$  i  $y$ , a potom i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija. NAPOMENA: *Pozicije se broje počev od pozicije bita najmanje težine, pri čemu je bit najmanje težine na poziciji nula.*

## Primer 1

```

INTERAKCIJA SA PROGRAMOM:
Unesite neoznaceni broj x: 235
Unesite neoznaceni broj n: 5
Unesite neoznaceni broj p: 10
Unesite neoznaceni broj y: 127
x = 235 = 00000000000000000000000011101011
y = 127 = 00000000000000000000000001111111
reset(235, 5, 10) = 0000000000000000000000000101011
set(235, 5, 10) = 0000000000000000000000001111101011
get_bits(235, 5, 10) = 00000000000000000000000000000011
set_n_bits(235, 5, 10, 127) = 0000000000000000000000001111101011
invert(235, 5, 10) = 00000000000000000000000011100101011

```

[Rešenje 1.8]

**Zadatak 1.9** Pod rotiranjem ulevo podrazumeva se pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najviše težine pomera na poziciju najmanje težine. Analogno, rotiranje udesno podrazumeva pomeranje svih bitova za jednu poziciju udesno, s tim što se bit sa pozicije najmanje težine pomera na poziciju najviše težine.

- Napisati funkciju `rotate_left` koja određuje broj koji se dobija rotiranjem  $k$  puta ulevo datog celog broja  $x$ .
- Napisati funkciju `rotate_right` koja određuje broj koji se dobija rotiranjem  $k$  puta udesno datog celog neoznačenog broja  $x$ .
- Napisati funkciju `rotate_right_signed` koja određuje broj koji se dobija rotiranjem  $k$  puta udesno datog celog broja  $x$ .

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

## Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite neoznaceni ceo broj x:  B10011A7
Unesite neoznaceni ceo broj k:  5
x = 10110001000000000000100011010011
rotate_left(2969571751, 5) = 00100000000000100011010011110110
rotate_right(2969571751, 5) = 00111101100010000000000010001101
rotate_right_signed(2969571751, 5) = 00111011000100000000000010001101
```

[Rešenje 1.9]

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

Test 1

```

ULAZ:
    255
IZLAZ:
    0000000000000000000000001001010101
    1010101001000000000000000000000000

```

## Test 2

```
ULAZ:  
-15  
IZLAZ:  
1111111111111111111111101011  
110101111111111111111111111
```

[Rešenje 1.10]

**Zadatak 1.11** Napisati funkciju `int Broj01(unsigned int n)` koja za dati broj `n` vraća 1 ako u njegovom binarnom zapisu ima više jednica 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:
    111111115
IZLAZ:
    0
```

[Rešenje 1.11]

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

## Test 1

```

ULAZ:
    11
IZLAZ:
    1

```

## Test 2

```

ULAZ:
    1024
IZLAZ:
    0

```

### Test 3

ULAZ:  
2147377146  
IZLAZ:  
22

[Rešenje 1.12]

**Zadatak 1.13** Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama  $i$  i  $j$ . Pozicije  $i$  i  $j$  se učitavaju kao parametri komandne linije. Smatrati da je krajnji desni bit binarne reprezentacije 0-ti bit. Pri rešavanju nije dozvoljeno koristiti ni pomoćni niz ni aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ .

*Primer 1*

```

|| Poziv: ./a.out 1 2
||
|| INTERAKCIJA SA PROGRAMOM:
|| ULAZ:
|| 11
|| Izlaz:
|| 13

```

*Primer 2*

```

|| Poziv: ./a.out 1 2
||
|| INTERAKCIJA SA PROGRAMOM:
|| ULAZ:
|| 1024
|| Izlaz:
|| 1024

```

*Primer 2*

```

|| Poziv: ./a.out 12 12
||
|| INTERAKCIJA SA PROGRAMOM:
|| ULAZ:
|| 12345
|| Izlaz:
|| 12345

```

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

*Test 1*

```

|| ULAZ:
|| 11
|| Izlaz:
|| 0000000B

```

*Test 2*

```

|| ULAZ:
|| 1024
|| Izlaz:
|| 00000400

```

*Test 3*

```

|| ULAZ:
|| 12345
|| Izlaz:
|| 00003039

```

[Rešenje 1.14]

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

*Test 1*

```

|| ULAZ:
|| 123 10
|| Izlaz:
|| 4294967285

```

*Test 2*

```

|| ULAZ:
|| 3251 0
|| Izlaz:
|| 4294967295

```

*Test 3*

```

|| ULAZ:
|| 12541 1024
|| Izlaz:
|| 4294966271

```

**Zadatak 1.16** Napisati funkciju koja računa koliko petica bi imao ceo neoznačen broj  $x$  u oktalanom zapisu. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Zadatak rešiti isključivo korišćenjem bitskih operatora.*

*Test 1*

```

|| ULAZ:
|| 123
|| Izlaz:
|| 0

```

*Test 2*

```

|| ULAZ:
|| 3245
|| Izlaz:
|| 2

```

*Test 3*

```

|| ULAZ:
|| 100328
|| Izlaz:
|| 1

```

## 1.3 Rekurzija

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

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

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

*Primer 1*

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

*Primer 2*

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

[Rešenje 1.17]

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

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

Napisati program koji testira napisane funkcije tako što za heksadekadni broj koji se unosi sa standardnog ulaza ispisuje da li je broj njegovih cifara paran ili neparan.

*Test 1*

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

*Test 2*

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

[Rešenje 1.18]

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

*Primer 1*

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

*Primer 2*

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

[Rešenje 1.19]

**Zadatak 1.20** Elementi niza  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

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

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

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

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

*Primer 1*

```

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

```

*Primer 2*

```

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

```

[Rešenje 1.20]

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

*Test 1*

```

ULAZ:
123
IZLAZ:
6

```

*Test 2*

```

ULAZ:
23156
IZLAZ:
17

```

*Test 3*

```

ULAZ:
1432
IZLAZ:
10

```

*Test 4*

```

ULAZ:
1
IZLAZ:
1

```

*Test 5*

```

ULAZ:
0
IZLAZ:
0

```

[Rešenje 1.21]

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

- sabirajući elemente počev od početka niza ka kraju niza,
- sabirajući elemente počev od kraja niza ka početku niza.

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

*Primer 1*

```

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

```

*Primer 2*

```

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

```

[Rešenje 1.22]

**Zadatak 1.23** Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju za niz koji se unosi sa standardnog ulaza. Niz neće imati više od 256 elemenata. Njegovi elementi se unose sve do unosa kraja ulaza (EOF).

*Test 1*

```

ULAZ:
3 2 1 4 21
IZLAZ:
21

```

*Test 2*

```

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

```

*Test 3*

```

ULAZ:
1 11 3 5 8 1
IZLAZ:
11

```

[Rešenje 1.23]

**Zadatak 1.24** Napisati rekurzivnu funkciju koja izračunava skalarni proizvod dva data vektora. Napisati program koji testira ovu funkciju, za nizove koji se unose sa standardnog ulaza. Prvo se unosi dimenzija nizova, a zatim i njihovi elementi. Nizovi neće imati više od 256 elemenata.

*Primer 1*

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

*Primer 2*

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

[Rešenje 1.24]

**Zadatak 1.25** Napisati rekurzivnu funkciju koja računa broj pojavljivanja elementa  $x$  u nizu  $a$  dužine  $n$ . Napisati program koji testira ovu funkciju za broj  $x$  i niz  $a$  koji se unose sa standardnog ulaza. Prvo se unosi  $x$ , a zatim elementi niza sve do unosa kraja ulaza. Niz neće imati više od 256 elemenata.

*Primer 1*

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

*Primer 2*

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

*Primer 3*

```
INTERAKCIJA SA PROGRAMOM:
Unesite ceo broj:
1
Unesite elemente niza:
3 21 5 6
Broj pojavljivanja je 0
```

[Rešenje 1.25]

**Zadatak 1.26** Napisati rekurzivnu funkciju kojom se proverava da li su tri zadata broja uzastopni članovi niza. Potom, napisati program koji je testira. Sa standardnog ulaza se unose najpre tri tražena broja, a zatim elementi niza, sve do kraja ulaza. Pretpostaviti da neće biti uneto više od 256 brojeva.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite tri cela broja:
1 2 3
Unesite elemente niza:
4 1 2 3 4 5
Uneti brojevi jesu uzastopni 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 1.26]

**Zadatak 1.27** Napisati rekurzivnu funkciju koja vraća broj bitova koji su postavljeni na 1, u binarnoj reprezentaciji njenog celobrojnog argumenta. Napisati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza u heksadekadnom formatu.

*Test 1*

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

*Test 2*

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

*Test 3*

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

[Rešenje 1.27]





Test 1	Test 2	Test 3
<pre> ULAZ: 2 IZLAZ: 1 2 2 1 </pre>	<pre> ULAZ: 3 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1 </pre>	<pre> ULAZ: -5 Duzina permutacije mora biti broj iz intervala [0, 50]! </pre>

[Rešenje 1.32]

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

```

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

```

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

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

Test 1	Test 2
<pre> 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 </pre>	<pre> 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 </pre>

[Rešenje 1.33]

**Zadatak 1.34** Napisati rekurzivnu funkciju koja prikazuje sve varijacije sa ponavljanjem dužine  $n$  skupa  $\{a, b\}$ , i program koji je testira, za  $n$  koje se unosi sa standardnog ulaza.

Test 1	Test 2
<pre> ULAZ: 2 IZLAZ: a a a b b a b b </pre>	<pre> 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 </pre>

**Zadatak 1.35** *Hanojske kule*: Data su tri vertikalna štapa, na jednom se nalazi  $n$  diskova poluprečnika 1, 2, 3,... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, a preostali štap se koristi kao pomoćni štap prilikom premeštanja.

Napisati program koji za proizvoljnu vrednost  $n$ , koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

**Zadatak 1.36** *Modifikacija Hanojskih kula*: Data su četiri vertikalna štapa, na jednom se nalazi  $n$  diskova poluprečnika 1, 2, 3,... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala tri štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, premestajući jedan po jedan disk, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, pri čemu se preostala dva štapa koriste kao pomoćni štapovi prilikom premeštanja.

Napisati program koji za proizvoljnu vrednost  $n$ , koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

## 1.4 Rešenja

### Rešenje 1.1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
6     imaginaran deo kompleksnog broja */
7  typedef struct {
8     float real;
9     float imag;
10 } KompleksanBroj;
11
12 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
13    kompleksnog broja i smesta ih u strukturu cija je adresa argument
14    funkcije */
15 void ucitaj_kompleksan_broj(KompleksanBroj * z)
16 {
17     /* Ucitavanje vrednosti sa standardnog ulaza */
18     printf("Unesite realan i imaginaran deo kompleksnog broja: ");
19     scanf("%f", &z->real);
20     scanf("%f", &z->imag);
21 }
22
23 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
24    obliku (x + i y). Ovoj funkciji se argument prenosi po vrednosti
25    jer se u samoj funkciji ne menja njegova vrednost */
26 void ispisi_kompleksan_broj(KompleksanBroj z)
27 {
28     /* Zapocinje se sa ispisom */
29     printf("(");
30
31     /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
32        razlicit od nule: tada se realni deo ispisuje na standardni
33        izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
34        imaginarni deo pozitivan ili negativan, a potom i apsolutna
35        vrednost imaginarnog dela kompleksnog broja 2) realni deo
36        kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
37        s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
38        decimalnih mesta */
39
40     if (z.real != 0) {
41         printf("%.2f", z.real);
42
43         if (z.imag > 0)
44             printf(" + %.2f i", z.imag);
45         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 }

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

/* Funkcija vraća vrednosti realnog dela kompleksnog broja */
60 float realan_deo(KompleksanBroj z)
{
62     return z.real;
64 }

/* Funkcija vraća vrednosti imaginarnog dela kompleksnog broja */
66 float imaginarni_deo(KompleksanBroj z)
{
68     return z.imag;
70 }

/* Funkcija vraća vrednost modula zadatog kompleksnog broja */
72 float moduo(KompleksanBroj z)
{
74     return sqrt(z.real * z.real + z.imag * z.imag);
76 }

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

84     KompleksanBroj z1 = z;

86     z1.imag *= -1;

88     return z1;
90 }

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

100     KompleksanBroj z = z1;

102     z.real += z2.real;
    z.imag += z2.imag;
104
    return z;
106 }

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

116     KompleksanBroj z = z1;

118     z.real -= z2.real;
```

```

120     z.imag -= z2.imag;

122     return z;
123 }

124 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
125    argumenata funkcije */
126 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
127 {
128     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
129        broj ciji se realan i imaginaran deo racunaju po formuli za
130        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;
139 }

140 /* Funkcija vraca argument zadatog kompleksnog broja */
142 float argument(KompleksanBroj z)
143 {
144     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
145        iz biblioteke math.h */

146     return atan2(z.imag, z.real);
147 }

150 int main()
151 {
152     char c;

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

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

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

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

180     /* Analizira se uneti operator */
181     if (c == '+') {
182         /* Racuna se zbir */
183         z = saberi(z1, z2);
184     } else if (c == '-') {
185         /* Racuna se razlika */
186         z = oduzmi(z1, z2);
187     } else {
188         /* Racuna se proizvod */
189         z = mnozi(z1, z2);
190     }

192     /* Ispisuje se rezultat */

```

```

printf("\n");
194 ispisi_kompleksan_broj(z1);
printf(" %c ", c);
196 ispisi_kompleksan_broj(z2);
printf(" = ");
198 ispisi_kompleksan_broj(z);
printf("\n");
200
/* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
202 broja */
printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo: %.f\n",
204 realan_deo(z), imaginaran_deo(z), moduo(z));
206
/* Izracunava se i ispisuje konjugovano kompleksan broj drugog
kompleksnog broja */
208 printf("Njegov konjugovano kompleksan broj: ");
ispisi_kompleksan_broj(konjugovan(z));
210 printf("\n");
212
/* Testira se funkcija koja racuna argument kompleksnog broja */
printf("Argument kompleksnog broja: %.f\n", argument(z));
214
return 0;
216 }

```

## Rešenje 1.2

Datoteka 1.1: *complex.h*

```

1 /* Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
   deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavlje
3 nikada ne treba da sadrzi definicije funkcija. Da bi neki program
mogao da koristi ove brojeve i funkcije iz ove biblioteke,
5 neophodno je da ukljuci ovo zaglavlje. */
6
7 /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i
onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
9 posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
narednoj pretprocesorskoj define direktivi. */
11 #ifndef _COMPLEX_H
#define _COMPLEX_H
13
14 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
15 koje se koriste u definicijama funkcija navedenim u complex.c */
#include <stdio.h>
17 #include <math.h>
18
19 /* Struktura KompleksanBroj */
typedef struct {
21 float real;
float imag;
23 } KompleksanBroj;
24
25 /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
definisane u complex.c */
27
28 /* 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);
32
33 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
obliku (x + i y) */
35 void ispisi_kompleksan_broj(KompleksanBroj z);
36
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 */
#endif

```

Datoteka 1.2: *complex.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 complex.h */
#include "complex.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
22 razlicit od nule: tada se realni deo ispisuje na standardni
izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
24 imaginarni deo pozitivan ili negativan, a potom i apsolutna
vrednost imaginarnog dela kompleksnog broja 2) realni deo
26 kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
28 decimalnih mesta */

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

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

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

```

```
46 }
47
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 }
100
101 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
102 {
103     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
104        broj ciji se realan i imaginaran deo racunaju po formuli za
105        mnozenje kompleksnih brojeva z1 i z2 */
106     KompleksanBroj z;
107
108     z.real = z1.real * z2.real - z1.imag * z2.imag;
109     z.imag = z1.real * z2.imag + z1.imag * z2.real;
110
111     return z;
112 }
113
114 float argument(KompleksanBroj z)
115 {
116     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
117        iz biblioteke math.h */
118     return atan2(z.imag, z.real);
```



```
}

```

Datoteka 1.3: *main.c*

```

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

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

9  Kompilacija se moze uraditi i na sledeci nacin:
11 gcc -Wall -c -o complex.o complex.c
   gcc -Wall -c -o main.o main.c
13 gcc -lm -o complex complex.o main.o

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

23 #include <stdio.h>
   /* Ukljucuje se zaglavlje neophodno za rad sa kompleksnim brojevima */
25 #include "complex.h"

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

33     /* Ucitavamo kompleksan broj */
       ucitaj_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
41     return 0;
   }

```

### Rešenje 1.3

Datoteka 1.4: *polinom.h*

```

1  #ifndef _POLINOM_H
2  #define _POLINOM_H

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

6
   /* Maksimalni stepen polinoma */
8  #define MAX_STEPEN 20

10
   /* Polinomi se predstavljaju strukturom koja cuva koeficijente
12    (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
   typedef struct {
14     double koef[MAX_STEPEN + 1];
       int stepen;
16 } Polinom;

18 /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
   obliku. Polinom se prenosi po adresi da bi se uštedela memorija:
20    ne kopira se cela struktura, vec se samo prenosi adresa na kojoj

```

```

    se nalazi polinom koji ispisujemo */
22 void ispisi(const Polinom * p);

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

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

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

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

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

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

```

Datoteka 1.5: *polinom.c*

```

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

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

28 Polinom ucitaj()
30 {
    int i;
32     Polinom p;

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

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

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

```

```

48  /* Vraca se procitani polinom */
    return p;
50 }

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

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

60     double rezultat = 0;
62     int i = p->stepen;
64     for (; i >= 0; i--)
        rezultat = rezultat * x + p->koef[i];
66     return rezultat;
}

68 Polinom saberi(const Polinom * p, const Polinom * q)
{
70     Polinom rez;
72     int i;

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

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

88     /* Vraca se dobijeni polinom */
    return rez;
90 }

92 Polinom pomnozi(const Polinom * p, const Polinom * q)
94 {
    int i, j;
96     Polinom r;

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

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

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

114    /* Vraca se dobijeni polinom */
    return r;
116 }

118 Polinom izvod(const Polinom * p)
120 {

```

```

122     int i;
    Polinom r;

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

130         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
            polaznog polinoma */
132         for (i = 0; i <= r.stepen; i++)
            r.koef[i] = (i + 1) * p->koef[i + 1];
134     } else
        r.koef[0] = r.stepen = 0;

136     /* Vraca se dobijeni polinom */
138     return r;
}

140 Polinom nIzvod(const Polinom * p, int n)
142 {
    int i;
    Polinom r;

144     /* Provera da li je n nenegativna vrednost */
    if (n < 0) {
146         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
148         exit(EXIT_FAILURE);
150     }

152     /* Multi izvod je bas taj polinom */
    if (n == 0)
154         return *p;

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

162     /* Vraca se dobijeni polinom */
    return r;
164 }

```

Datoteka 1.6: *main.c*

```

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

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

10     /* Unos polinoma p */
    printf
12     ("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 ispisuje se izracunati zbir */

```

```

24  r = saberi(&p, &q);
    printf("Zbir polinoma je: ");
26  ispisi(&r);

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

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

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

46  return 0;
}

```

### Rešenje 1.5

```

#include <stdio.h>

2  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji. Bitove koji predstavljaju binarnu
4  reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
   najvece tezine ka bitu najmanje tezine */
6  void print_bits(unsigned x)
8  {
    /* Broj bitova celog broja */
10   unsigned velicina = sizeof(unsigned) * 8;

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

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

28   putchar('\n');
   }
30

32  int main()
   {
34     int broj;

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

38     /* I ispisuje se njegova binarna reprezentacija */
40     print_bits(broj);

42     return 0;
   }

```

## Rešenje 1.6

```
1 #include <stdio.h>

3 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   kreiranjem odgovarajuće maske i njenim pomeranjem */
5 int count_bits1(int x)
6 {
7     int br = 0;
8     unsigned wl = sizeof(unsigned) * 8 - 1;

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

19     return br;
20 }

21 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
22 formiranjem odgovarajuće maske i pomeranjem promenljive x */
23 int count_bits2(int x)
24 {
25     int br = 0;
26     unsigned wl = sizeof(int) * 8 - 1;

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

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

39     return br;
40 }

42 int main()
43 {
44     int x, i;

46     /* Učitava se broj sa ulaza */
47     printf("Unesite broj:\n");
48     scanf("%x", &x);

50     /* Dozvoljava se korisniku da bira na koji način će biti izračunat
51     broj jedinica u zapisu broja */
52     printf("Unesite redni broj funkcije:\n");
53     scanf("%d", &i);

55     /* Ispisuje se rezultat */
56     printf("Broj jedinica u zapisu je\n");
57     if (i == 1)
58         printf("funkcija count_bits1: %d\n", count_bits1(x));
59     else
60         printf("funkcija count_bits2: %d\n", count_bits2(x));

62     return 0;
63 }
```

## Rešenje 1.7

```
1 #include <stdio.h>

3 /* Funkcija vraca najveći neoznačeni broj sastavljen od istih bitova
```

```

    koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
5 unsigned najveći(unsigned x)
{
7     unsigned velicina = sizeof(unsigned) * 8;

9     /* Formira se maska 100000...0000000 */
    unsigned maska = 1 << (velicina - 1);

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

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

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

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

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

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

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

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

67     putchar('\n');
}

69 int main()
71 {
    int broj;

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

```

```

77  /* Ispisuju se, redom, najveći i najmanji broj formirani od bitova
    unetog broja */
79  printf("Najveći:\n");
    print_bits(najveći(broj));
81
    printf("Najmanji:\n");
83  print_bits(najmanji(broj));
85
    return 0;
}

```

## Rešenje 1.8

```

#include <stdio.h>

2
/* Funkcija postavlja na nulu n bitova počev od pozicije p. */
4 unsigned reset(unsigned x, unsigned n, unsigned p)
{
6  /******
    Formira se maska čija binarna reprezentacija ima n bitova
    postavljenih na 0 počev od pozicije p, dok su svi ostali
    postavljeni na 1. Na primer, za n=5 i p=10 formira se
    maska oblika
    1111 1111 1111 1111 1111 1000 0011 1111
    To se postize na sledeći način:
    ~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 1000 0011 1111
18  *****/
    unsigned maska = ~(~0 << n) << (p - n + 1);
20
    return x & maska;
22 }

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

40
/* Funkcija vraća celobrojno polje bitova, desno poravnato, koje
    predstavlja n bitova počev od pozicije p u binarnoj
42  reprezentaciji broja x. */
unsigned get_bits(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
    0000 0000 0000 0000 0000 0000 0001 1111
50  *****/
    unsigned maska = ~(~0 << n);
52
    /* Najpre se vrednost promenljive x pomera u desno tako da traženo
    polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
    željenih n i funkcija vraća tako dobijenu vrednost */
54
    return maska & (x >> (p - n + 1));
56
}
58

```



```

60 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
   postavljani na vrednosti n bitova najnize tezine binarne
62 reprezentacije broja y */
   unsigned set_n_bits(unsigned x, unsigned n, unsigned p, unsigned y)
64 {
   /* Kreira se maska kod koje su poslednjih n bitova 1, a
66    ostali su 0. */
   unsigned last_n_1 = ~(~0 << n);

68   /* Kao i kod funkcije reset, i ovde se kreira maska koja ima n
   bitova postavljenih na 0 pocevsi od pozicije p, dok su
70    ostali bitovi 1. */
   unsigned middle_n_0 = ~(~0 << n) << (p - n + 1);

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

76   /* U promenljivu y_shift_middle se smesta vrednost dobijena od
   binarne reprezentacije vrednosti promenljive y ciji je n bitova
80    najnize tezine pomera tako da stoje pocev od pozicije p. Ostali
   bitovi su nule. Sa (y & last_n_1) resetuju se svi bitovi osim
82    najnižih n */
   unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);

84   return x_reset ^ y_shift_middle;
86 }
88

90 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
   njih n */
   unsigned invert(unsigned x, unsigned n, unsigned p)
92 {
   /* Formira se maska sa n jedinica pocev od pozicije p. */
94   unsigned maska = ~(~0 << n) << (p - n + 1);

96   /* Operator ekskluzivno ili invertuje sve bitove gde je
   odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
98   return maska ^ x;
100 }

102 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji */
   void print_bits(int x)
104 {
106   unsigned velicina = sizeof(int) * 8;
   unsigned maska;

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

110   putchar('\n');
112 }
114

116 int main()
   {
118   unsigned x, p, n, y;

120   /* Ucitavaju se vrednosti sa standardnog ulaza */
   printf("Unesite neoznaceni broj x:\n");
122   scanf("%u", &x);
   printf("Unesite neoznaceni broj n:\n");
124   scanf("%u", &n);
   printf("Unesite neoznaceni broj p:\n");
126   scanf("%u", &p);
   printf("Unesite neoznaceni broj y:\n");
128   scanf("%u", &y);

130   /* Ispisuje se binarne reprezentacije brojeva x i y */
   printf("x = %u = ", x);
132   print_bits(x);

```

```
134 printf("y = %u = ", y);
    print_bits(y);
136
    /* Testira se rad napisanih funkcija */
138 printf("reset(%u,%u,%u) = ", x, n, p);
    print_bits(reset(x, n, p));
140
    printf("set(%u,%u,%u) = ", x, n, p);
142 print_bits(set(x, n, p));
144
    printf("get_bits(%u,%u,%u) = ", x, n, p);
    print_bits(get_bits(x, n, p));
146
    printf("set_n_bits(%u,%u,%u,%u) = ", x, n, p, y);
148 print_bits(set_n_bits(x, n, p, y));
150
    printf("invert(%u,%u,%5u) = ", x, n, p);
    print_bits(invert(x, n, p));
152
    return 0;
154 }
```

### Rešenje 1.9

```
1 #include <stdio.h>
2
3 /* Funkcija neoznaceni broj x rotira u levo za n mesta. */
4 unsigned rotate_left(unsigned x, unsigned n)
5 {
6     unsigned first_bit;
7
8     /* Maska koja ima samo najvisi bit postavljen na 1 je neophodna da
9        bi pre pomeranja u levo za 1 najvisi bit bio sacuvan */
10    unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
11    int i;
12
13    /* n puta se vrši rotaciju za jedan bit u levo. U svakoj iteraciji
14       se odredi prvi bit, a potom se pomera binarna reprezentacija
15       trenutne vrednosti promenljive x u levo za 1. Nakon toga,
16       najnizi bit se postavlja na vrednost koju je imao prvi bit koji
17       je istisnut pomeranjem */
18    for (i = 0; i < n; i++) {
19        first_bit = x & first_bit_mask;
20        x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
21    }
22
23    /* Vraca se dobijena vrednost */
24    return x;
25 }
26
27 /* Funkcija neoznaceni broj x rotira u desno za n mesta. */
28 unsigned rotate_right(unsigned x, unsigned n)
29 {
30     unsigned last_bit;
31     int i;
32
33    /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
34       iteraciji se odredjuje bit najmanje tezine broja x, zatim tako
35       odredjeni bit se pomera u levo tako da najnizi bit dodje do
36       pozicije najviseg bita. Zatim, nakon pomeranja binarne
37       reprezentacije trenutne vrednosti promenljive x za 1 u desno,
38       najvisi bit se postavlja na vrednost vec zapamcenog bita koji je
39       bio na poziciji najmanje tezine. */
40    for (i = 0; i < n; i++) {
41        last_bit = x & 1;
42        x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
43    }
44
45    /* Vraca se dobijena vrednost */
46    return x;
47 }
```

```

49 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
   argument funkcije x oznaceni ceo broj */
51 int rotate_right_signed(int x, unsigned n)
   {
53     unsigned last_bit;
54     int i;
55
56     /* U svakoj iteraciji se odredjuje bit najmanje tezine i smesta u
57        promenljivu last_bit. Kako je x oznacen ceo broj, tada se
58        prilikom pomeranja u desno vrši aritmeticko pomeranje i cuva se
59        znak broja. Dakle, razlikuju se dva slucaja u zavisnosti od
60        znaka broja x. Nije dovoljno da se ova provera izvrši pre
61        petlje, s obzirom da rotiranjem u desno na mesto najviseg bita
62        moze doci i 0 i 1, nezavisno od pocetnog znaka broja smestenog u
63        promenljivu x. */
64     for (i = 0; i < n; i++) {
65         last_bit = x & 1;
66
67         if (x < 0)
68             /* *****
69              Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
70              na najvisoj poziciji. Na primer ako je x
71              1010 1011 1100 1101 1110 0001 0010 001b
72              (sa b je oznacen ili 1 ili 0 na najnizoj poziciji)
73              Onda je sadrzaj promenljive last_bit:
74              0000 0000 0000 0000 0000 0000 0000 000b
75              Nakon siftovanja sadrzaja promenljive x za 1 u desno
76              1101 0101 1110 0110 1111 0000 1001 0001
77              Kako bi umesto 1 na najvisoj poziciji u trenutnoj binarnoj
78              reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
79              najvisu poziciju jer bi se time dobile 0, a u ovom slucaju su
80              potrebne jedinice zbog bitovskog & zato se prvo vrši
81              komplementiranje, a zatim pomeranje
82              ~last_bit << (sizeof(int)*8 - 1)
83              B000 0000 0000 0000 0000 0000 0000 0000
84              gde B oznacava ~b.
85              Potom se ponovo vrši komplementiranje kako bi se b nalazilo na
86              najvisoj poziciji i sve jedinice na ostalim pozicijama
87              ~(-last_bit << (sizeof(int)*8 - 1))
88              b111 1111 1111 1111 1111 1111 1111 1111
89              ***** */
90             x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
91         else
92             x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
93     }
94
95     /* Vraca se dobijena vrednost */
96     return x;
97 }
98
99 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
100 celog broja u memoriji */
101 void print_bits(int x)
102 {
103     unsigned velicina = sizeof(int) * 8;
104     unsigned maska;
105     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
106         putchar(x & maska ? '1' : '0');
107
108     putchar('\n');
109 }
110
111 int main()
112 {
113     unsigned x, k;
114
115     /* Ucitavaju se vrednosti sa standardnog ulaza */
116     printf("Unesite neoznaceni ceo broj x:");
117     scanf("%x", &x);
118     printf("Unesite neoznaceni ceo broj k:");
119     scanf("%x", &k);

```

```
121 /* Ispisuje se binarna reprezentacija broja x */
123 printf("x = ", "");
125 print_bits(x);

127 /* Testira se rad napisanih funkcija */
129 printf("rotate_left(%u,%u) = ", x, k);
131 print_bits(rotate_left(x, k));

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

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

return 0;
}
```

### Rešenje 1.10

```
1 #include <stdio.h>

3 /* Funkcija vraca vrednost cija je binarna reprezentacija slika
   u ogledalu binarne reprezentacije broja x. */
5 unsigned mirror(unsigned x)
6 {
7     unsigned najnizi_bit;
8     unsigned rezultat = 0;
9
10    int i;
11    /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
       vrednosti broja x se određuje i pamti u promenljivoj
13    najnizi_bit, nakon čega se na promenljivu x primeni pomeranje u
       desno */
15    for (i = 0; i < sizeof(x) * 8; i++) {
16        najnizi_bit = x & 1;
17        x >>= 1;
18        /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
       postavljani bitovi dobijaju veću poziciju. Novi bit se
19        postavlja na najnižu poziciju */
21        rezultat <<= 1;
22        rezultat |= najnizi_bit;
23    }

25    /* Vraca se dobijena vrednost */
27    return rezultat;
29 }

31 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji */
32 void print_bits(int x)
33 {
34     unsigned velicina = sizeof(int) * 8;
35     unsigned maska;
36     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
37         putchar(x & maska ? '1' : '0');
38
39     putchar('\n');
40 }

41 int main()
42 {
43     int broj;

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

48     /* Ispisuje se njegova binarna reprezentacija */
49     print_bits(broj);

51     /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
       funkcije mirror */
}
```

```

53     print_bits(mirror(broj));
55     return 0;
}

```

### Rešenje 1.11

```

#include <stdio.h>
2
/* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
4   jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
int Broj01(unsigned int n)
6 {
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) * 4 - 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) {
26             broj_jedinica++;
28         } else {
                broj_nula++;
            }
30
        /* Pomera se maska u desnu stranu */
32         maska = maska >> 1;
34     }
36
    /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
       suprotnom vraca 0 */
    return (broj_jedinica > broj_nula) ? 1 : 0;
38 }
40
int main()
42 {
    unsigned int n;
44
    /* Ucitava se broj sa ulaza */
46     scanf("%u", &n);
48
    /* Ispisuje se rezultat */
    printf("%d\n", Broj01(n));
50
    return 0;
52 }

```

### Rešenje 1.12

```

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

```

```
10 unsigned int maska;
12 /* Vrednost promenljive koja predstavlja broj parova se
   inicijalizuje na 0 */
   broj_parova = 0;
14
16 /* Postavlja se maska tako da moze da procitamo da li su dva
   najmanja bita u zapisu broja x 11 */
18 /* Binarna reprezentacija broja 3 je 000...00011 */
   maska = 3;
20
22 while (x != 0) {
24     /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par */
   if ((x & maska) == maska) {
26         broj_parova++;
28     }
29
30     /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
       proveravao sledeci par bitova. Pomeranjem u desno bit najvece
       tezine se popunjava nulom jer je x neoznaceni broj */
   x = x >> 1;
32 }
34
36 /* Vraca se dobijena vrednost */
   return broj_parova;
38 }
39
40 int main()
41 {
42     unsigned int x;
43
44     /* Ucitava se broj sa ulaza */
   scanf("%u", &x);
46
48     /* Ispisuje se rezultat */
   printf("%d\n", broj_parova(x));
50
52     return 0;
53 }
```

### Rešenje 1.14

```
1 #include <stdio.h>
2
3 /* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1 jer
4    su za svaku 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 MAX_DUZINA sizeof(unsigned int)*2 +1
7
8 /* Funkcija za neoznaceni broj x formira nisku s koja sadrzi njegov
   heksadekadni zapis */
10 void prevod(unsigned int x, char s[])
12 {
14     int i;
   unsigned int maska;
16     int vrednost;
17
18     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
       maska za citanje 4 uzastopne cifre */
20     maska = 15;
21
22     /******
       Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
       tezine. Na primer za broj cija je binarna reprezentacija
24     0000000001101000100001111010101
26     u prvom koraku se citaju bitovi izdvojeni sa <...>:
       000000000110100010000111101<0101>
28     u drugom koraku:
```

```

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

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

38     /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
40      dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
42      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
44      odgovarajuće slovo 'A', 'B', ... 'F') */
    if (vrednost < 10) {
46        s[i] = vrednost + '0';
    } else {
48        s[i] = vrednost - 10 + 'A';
    }

50     /* Primenljiva x se pomera za 4 bita u desnu stranu i time se u
52      narednoj iteraciji posmatraju sledeca 4 bita */
    x = x >> 4;

54 }

56 /* Upisuje se terminirajuca nula */
s[MAX_DUZINA - 1] = '\0';
58 }

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

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

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

70     /* I stampa se dobijena niska */
72     printf("%s\n", s);

74     return 0;
}

```

### Rešenje 1.17

```

#include <stdio.h>

2
3 *****/
4 Resenje linearne slozenosti:
    x^0 = 1
6    x^k = x * x^(k-1)
    *****/
8 int stepen(int x, int k)
{
10     if (k == 0)
        return 1;

12     return x * stepen(x, k - 1);
14     /* kraci zapis: return k == 0 ? 1 : x * stepen(x,k-1); */
}

16
17 *****/
18 Resenje logaritamske slozenosti:
    x^0 = 1;
20    x^k = x * (x^2)^(k/2) , za neparno k
    x^k = (x^2)^(k/2) , za parno k
22    Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi

```

```
se doslo do rezultata, i stoga je efikasnije.
24 *****/
int stepen2(int x, int k)
26 {
    if (k == 0)
28     return 1;

    /* Ako je stepen paran */
    if ((k % 2) == 0)
32     return stepen2(x * x, k / 2);

    /* Inace (ukoliko je stepen neparan) */
    return x * stepen2(x * x, k / 2);
36 }

int main()
38 {
    int x, k, ind;
40

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

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

    /* Ispisuje se vrednost koju vraca odgovarajuca funkcija */
    if (x == 1)
52         printf("%d\n", stepen(x, k));
    else
54         printf("%d\n", stepen2(x, k));

58     return 0;
}
```

### Rešenje 1.18

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

4 #define MAX 100

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

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

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

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



```

34 int main()
35 {
36     int n;
37
38     /* Ucitava se ceo broj */
39     scanf("%d", &n);
40
41     /* Ispisuje se rezultat */
42     printf("Uneti broj ima %s paran broj cifara.\n",
43           (paran(n) == 1 ? "ne" : "ne"));
44
45     return 0;
46 }

```

### Rešenje 1.19

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Pomocna funkcija koja izracunava n! * result. Koristi repnu
5     rekurziju. Result je argument u kome se akumulira do tada
6     izracunatu vrednost faktorijela. Kada dodje do izlaza iz rekurzije
7     iz rekurzije potrebno je da vratimo result. */
8  int faktorijelRepna(int n, int result)
9  {
10     if (n == 0)
11         return result;
12
13     return faktorijelRepna(n - 1, n * result);
14 }
15
16 /* U sledecim funkcijama je prikazan postupak oslobadjanja od
17     repne rekurzije koja postoji u funkciji faktorijelRepna. */
18
19 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
20     naredbama kojima se vrednost argumenta funkcije postavlja na
21     vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
22     goto naredbe za vracanje na pocetak tela funkcije. */
23 int faktorijelRepna_v1(int n, int result)
24 {
25     pocetak:
26     if (n == 0)
27         return result;
28
29     result = n * result;
30     n = n - 1;
31     goto pocetak;
32 }
33
34 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
35     praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
36     iterativno resenje bez bezuslovnih skokova */
37 int faktorijelRepna_v2(int n, int result)
38 {
39     while (n != 0) {
40         result = n * result;
41         n = n - 1;
42     }
43
44     return result;
45 }
46
47 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
48     broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
49     ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
50     udobnosti korisnika, jer je sasvim prirodno da za faktorijel
51     zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
52     sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
53     faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
54     funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel

```

```
55     poziv faktorijelRepna sa pozivom faktorijelRepna_v1, a zatim sa
56     pozivom funkcije faktorijelRepna_v2. */
57 int faktorijel(int n)
58 {
59     return faktorijelRepna(n, 1);
60 }
61
62 int main()
63 {
64     int n;
65
66     /* Ucitava se ceo broj */
67     printf("Unesite n (<= 12): ");
68     scanf("%d", &n);
69     if (n > 12) {
70         printf("Greska: nedozvoljena vrednost za n!\n");
71         exit(EXIT_FAILURE);
72     }
73
74     /* Ispisuje se rezultat */
75     printf("%d! = %d\n", n, faktorijel(n));
76
77     exit(EXIT_SUCCESS);
78 }
```

## Rešenje 1.20

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
5  int F_iterativna(int n, int a, int b)
6  {
7      int i;
8      int F_0 = 0;
9      int F_1 = 1;
10     int tmp;
11
12     if (n == 0)
13         return 0;
14
15     for (i = 2; i <= n; i++) {
16         tmp = a * F_1 + b * F_0;
17         F_0 = F_1;
18         F_1 = tmp;
19     }
20
21     return F_1;
22 }
23
24 /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
25 int F_rekurzivna(int n, int a, int b)
26 {
27     /* Izlaz iz rekurzije */
28     if (n == 0 || n == 1)
29         return n;
30
31     /* Rekurzivni pozivi */
32     return a * F_rekurzivna(n - 1, a, b) +
33         b * F_rekurzivna(n - 2, a, b);
34 }
35
36 /* 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*F(19) + b*F(18), a F(19) = a*F(18) + b*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 dinamicnim programiranjem.
43 Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
44 memoriji (obicno u nizovima ili matricama), odakle se koriste ako
```

```

47 tokom resavanja ponovo budu potrebni.
48
49 U narednoj funkciji vec izracunati clanovi niza se cuvaju u
50 statickom nizu celih brojeva, jer se staticki niz ne smesta
51 na stek, kao sto je to slucaj sa lokalnim promenljivama, vec na
52 segment podataka, odakle je dostupan svim pozivima
53 rekurzivne funkcije. */
54
55 /* c) Funkcija racuna n-ti broj niza F - napredna rekurzivna
56 verzija */
57 int F_napredna(int n, int a, int b)
58 {
59     /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
60     staticke promenljive na podrazumevane vrednosti. Stoga, elemente
61     celobrojnog niza inicijalizuje na 0 */
62     static int f[20];
63
64     /* Ako je podproblem vec ranije resen, koristi se resenje koje je
65     vec izracunato i */
66     if (f[n] != 0)
67         return f[n];
68
69     /* Izlaz iz rekurzije */
70     if (n == 0 || n == 1)
71         return f[n] = n;
72
73     /* Rekurzivni pozivi */
74     return f[n] =
75         a * F_napredna(n - 1, a, b) + b * F_napredna(n - 2, a, b);
76 }
77
78 int main()
79 {
80     int n, a, b, ind;
81
82     /* Unosi se redni broj funkcije koja ce se primeniti */
83     printf("Unesite redni broj funkcije koju zelite:\n");
84     printf("1 - iterativna\n");
85     printf("2 - rekurzivna\n");
86     printf("3 - rekurzivna napredna\n");
87     scanf("%d", &ind);
88
89     /* Ucitavaju se koeficijenti a i b */
90     printf("Unesite koeficijente:\n");
91     scanf("%d%d", &a, &b);
92
93     /* Ucitava se broj n */
94     printf("Unesite koji clan niza se racuna:\n");
95     scanf("%d", &n);
96
97     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
98     funkcije F_iterativna, F_rekurzivna ili F_napredna */
99     if (ind == 0)
100         printf("F(%d) = %d\n", n, F_iterativna(n, a, b));
101     else if (ind == 1)
102         printf("F(%d) = %d\n", n, F_rekurzivna(n, a, b));
103     else
104         printf("F(%d) = %d\n", n, F_napredna(n, a, b));
105
106     return 0;
107 }

```

### Rešenje 1.21

```

1 #include <stdio.h>
2
3 /* Funkcija odredjuje zbir cifara zadatog broja x */
4 int zbir_cifara(unsigned int x)
5 {
6     /* Izlazak iz rekurzije: ako je broj jednocifren */
7     if (x < 10)
8         return x;
9 }

```

```
10  /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
12     poslednje cifre + poslednja cifra tog broja */
13  return zbir_cifara(x / 10) + x % 10;
14  }
15
16  int main()
17  {
18     unsigned int x;
19
20     /* Ucitava se ceo broj */
21     scanf("%u", &x);
22
23     /* Ispisuje se zbir cifara ucitanog broja */
24     printf("%d\n", zbir_cifara(x));
25
26     return 0;
27 }
```

## Rešenje 1.22

```
1  #include <stdio.h>
2  #define MAX_DIM 1000
3
4  /* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
5     suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
6     element niza. */
7  int sumaNiza(int *a, int n)
8  {
9     if (n <= 0)
10        return 0;
11
12    return sumaNiza(a, n - 1) + a[n - 1];
13 }
14
15 /* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
16    jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
17    elementa niza i sume preostalih n-1 elementa. */
18 int sumaNiza2(int *a, int n)
19 {
20    if (n <= 0)
21        return 0;
22
23    return a[0] + sumaNiza2(a + 1, n - 1);
24 }
25
26 int main()
27 {
28    int a[MAX_DIM];
29    int n, i = 0, ind;
30
31    /* Ucitava se redni broj funkcije */
32    printf("Unesite redni broj funkcije (1 ili 2):\n");
33    scanf("%d", &ind);
34
35    /* Ucitava se broj elemenata niza */
36    printf("Unesite dimenziju niza:");
37    scanf("%d", &n);
38
39    /* Ucitava se n elemenata niza. */
40    printf("Unesite elemente niza:");
41    for (i = 0; i < n; i++)
42        scanf("%d", &a[i]);
43
44    /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
45       funkcije sumaNiza, odnosno sumaNiza2 */
46    if (ind == 1)
47        printf("Suma elemenata je %d\n", sumaNiza(a, n));
48    else
49        printf("Suma elemenata je %d\n", sumaNiza2(a, n));
50
51    return 0;
52 }
```

52 }

## Rešenje 1.23

```

#include <stdio.h>
#define MAX_DIM 256

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

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

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

int main()
{
    int brojevi[MAX_DIM];
    int n;

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

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

    return 0;
}

```

## Rešenje 1.24

```

#include <stdio.h>
#define MAX_DIM 256

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

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

```

```
21 int main()
22 {
23     int i, a[MAX_DIM], b[MAX_DIM], n;
24
25     /* Unosi se dimenzija nizova */
26     printf("Unesite dimenziju nizova:");
27     scanf("%d", &n);
28
29     /* Provera da li je dimenzija niza odgovarajuca */
30     if (n < 0 || n > MAX_DIM) {
31         printf("Dimenzija mora biti prirodan broj <= %d!\n", MAX_DIM);
32         return 0;
33     }
34
35     /* A zatim i elementi nizova */
36     printf("Unesite elemente prvog niza:");
37     for (i = 0; i < n; i++)
38         scanf("%d", &a[i]);
39
40     printf("Unesite elemente drugog niza:");
41     for (i = 0; i < n; i++)
42         scanf("%d", &b[i]);
43
44     /* Ispisuje se rezultat skalarnog proizvoda dva učitana niza */
45     printf("Skalarni proizvod je %d\n", skalarno(a, b, n));
46
47     return 0;
48 }
49
```

### Rešenje 1.25

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

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

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

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

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

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

```

n = i;
42
/* Ispisuje se broj pojavljivanja */
44 printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
46
return 0;
}

```

### Rešenje 1.26

```

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

/* Funkcija koja proverava da li su tri zadata broja uzastopni
   clanovi niza */
6 int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
{
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 */
10     if (n < 3)
        return 0;

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 */
14     return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
        || tri_uzastopna_clana(x, y, z, a, n - 1);
18 }

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

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

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

42     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana ispisuje
       se odgovarajuca poruka */
44     if (tri_uzastopna_clana(x, y, z, a, n))
        printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
46     else
        printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
48

50     return 0;
}

```

### Rešenje 1.27

```

#include <stdio.h>
2

/* Funkcija koja broji bitove postavljene na 1. */
4 int count(int x)
{
6     /* Izlaz iz rekurzije */

```

```

8   if (x == 0)
    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 najviseg bita. Rezultat koliko ima jedinica u ostatku
    binarnog zapisa broja x se uvecava za 1. Najvisi bit je 0. Stoga
14  je broj jedinica u zapisu x isti kao broj jedinica u zapisu
    broja x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0.
16  Za rekurzivni poziv se salje vrednost koja se dobija kada se x
    pomeri u levo. Napomena: argument funkcije x je oznacen ceo
18  broj, usled cega se ne koristi pomeranje udesno, jer funkciji
    moze biti prosledjen i negativan broj. Iz tog razloga,
20  odlucujemo se da proveramo najvisi, umesto najnizeg bita */
    if (x & (1 << (sizeof(x) * 8 - 1)))
22      return 1 + count(x << 1);
    else
24      return count(x << 1);
    /******
26    Krace zapisano
    return ((x & (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1);
28    *****/
}

30
32 int main()
33 {
34     int x;

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

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

40     return 0;
42 }

```

## Rešenje 1.29

```

#include <stdio.h>

2
/* Rekurzivna funkcija za odredjivanje najvece oktalne cifre u broju */
4 int max_oktalna_cifra(unsigned x)
5 {
6     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
    vrednost najvece oktalne cifre u broju 0 */
8     if (x == 0)
        return 0;

10
    /* Odredjivanje poslednje oktalne cifre u broju */
12     int poslednja_cifra = x & 7;

14     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
    izbrise poslednja oktalna cifra */
16     int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);

18     return poslednja_cifra > max_bez_poslednje_cifre ? poslednja_cifra
        : max_bez_poslednje_cifre;
20 }

22
24 int main()
25 {
26     unsigned x;

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

30     /* Ispisuje se vrednost najvece oktalne cifre unetog broja */
    printf("%d\n", max_oktalna_cifra(x));

32     return 0;
33 }

```



## Rešenje 1.30

```

2  #include <stdio.h>

4  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
   broj u */
6  int max_heksadekadna_cifra(unsigned x)
   {
8     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
       vrednost najveće heksadekadne cifre u broju 0 */
10    if (x == 0)
        return 0;

12    /* Odredjivanje poslednje heksadekadne cifre u broju */
14    int poslednja_cifra = x & 15;

16    /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
       njega izbrise poslednja heksadekadna cifra */
18    int max_bez_poslednje_cifre = max_heksadekadna_cifra(x >> 4);

20    return poslednja_cifra > max_bez_poslednje_cifre ? poslednja_cifra
        : max_bez_poslednje_cifre;
22 }

24 int main()
   {
26     unsigned x;

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

30     /* Ispisuje se vrednost najveće heksadekadne cifre unetog broja */
32     printf("%d\n", max_heksadekadna_cifra(x));

34     return 0;
   }

```

## Rešenje 1.31

```

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

4  /* Niska može imati najviše 31 karaktera + 1 za terminalnu nulu */
   #define MAX_DIM 32

6  /* Funkcija ispituje da li je zadata niska dužine n palindrom */
8  int palindrom(char s[], int n)
   {
10     /* Izlaz iz rekurzije - trivijalno, niska dužine 0 ili 1 je
       palindrom */
12     if ((n == 1) || (n == 0))
        return 1;

14     /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
       poslednji karakter i da je palindrom niska koja nastaje kada se
       polaznoj niski otklone prvi i poslednji karakter */
18     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
   }

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

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

28         /* Odredjuje se dužina 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 1.32

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

4
/* Funkcija koja ispisuje elemente niza a duzine n */
6 void ispisiNiz(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} a[] je
    niz u koji se smesta permutacija m oznacava da se na m-tu poziciju
32 u permutaciji smesta jedan od preostalih celih brojeva n je
    velicina skupa koji se permutuje Funkciju se poziva sa argumentom
34 m=1 jer formiranje permutacije pocinje od 1. pozicije. Stoga, nece
    se koristiti a[0]. */
36 void permutacija(int a[], int m, int n)
{
38     int i;

40     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
        premasila velicinu skupa, onda se svi brojevi vec nalaze u
42 permutaciji i ispisuje se permutacija. */
    if (m > n) {
44         ispisiNiz(a, n);
        return;
46     }

48     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
        mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
50 Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
        ovako postavljenom pocetku permutacije. Kada se to zavrshi, vrsi
52 se proverda da li postoji jos neki broj koji moze da se stavi na
        m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
54 funkcija zavrшава sa radom. Ukoliko takav broj postoji, onda se
        ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
56 ali sada sa drugacije postavljenim prefiksom. */
    for (i = 1; i <= n; i++) {
58         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
            pozicije, onda se on postavlja na poziciju m i poziva se
60 ponovo funkcija da dopuni ostatak permutacije posle upisivanja
            i na poziciju m. Inace, nastavlja se dalje, trazeci broj koji
62 se nije pojavio do sada u permutaciji. */
```

```

    if (!koriscen(a, m - 1, i)) {
64         a[m] = i;
        permutacija(a, m + 1, n);
66     }
68 }

70 int main(void)
71 {
72     int n;
73     int a[MAX_DUZINA_PERMUTACIJE+1];
74
75     /* Ucitava se broja n i proverava se da li je u odgovarajucem opsegu */
76     scanf("%d", &n);
77     if (n < 0 || n > MAX_DUZINA_PERMUTACIJE) {
78         fprintf(stderr,
79             "Duzina permutacije mora biti broj iz intervala [0, %d]!\n",
80             MAX_DUZINA_PERMUTACIJE);
81         exit(EXIT_FAILURE);
82     }
83
84     /* Ispisuju se permutacije duzine n */
85     permutacija(a, 1, n);
86
87     exit(EXIT_SUCCESS);
88 }

```

### Rešenje 1.33

```

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

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

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

```

```
44  /* Ucitavaju se brojevi d i r */
    scanf("%d %d", &d, &r);

46

48  /* Ispisuje se Paskalov trougao */
    putchar('\n');
    for (n = 0; n <= d; n++) {
50      for (i = 0; i < d - n; i++)
          printf(" ");
52      for (k = 0; k <= n; k++)
          printf("%4d", binomniKoefficient(n, k));
54      putchar('\n');
    }

56

    /* Provera da li je r nenegativan */
58    if (r < 0) {
        fprintf(stderr,
60            "Redni broj hipotenuze mora biti veci ili jednak od 0!\n");
        exit(EXIT_FAILURE);
62    }

64    /* Ispisuje se suma elemenata hipotenuze */
    printf("%d\n", sumaElemenataHipotenuze(r));
66
    exit(EXIT_SUCCESS);
68 }
```

## Glava 2

# Pokazivači

### 2.1 Pokazivačka aritmetika

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

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

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

#### Primer 1

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

#### Primer 2

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

[Rešenje 2.1]

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

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

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

- korišćenjem indeksne sintakse,
- korišćenjem pokazivačke sintakse.

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

*Primer 1*

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

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

*Primer 2*

```
Poziv: ./a.out

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

[Rešenje 2.4]

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

*Primer 1*

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

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

*Primer 2*

```
Poziv: ./a.out

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

[Rešenje 2.5]

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

*Primer 1*

```
POZIV: ./a.out ulaz.txt 1
ULAZ.TXT
Ovo je 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*

```
POZIV: ./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*

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

[Rešenje 2.6]

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

*Primer 1*

```
POZIV: ./a.out ulaz.txt ke -s
ULAZ.TXT
Ovo je 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*

```
POZIV: ./a.out ulaz.txt sa -p
ULAZ.TXT
Ovo je sadržaj datoteke i u njoj ima reci
koje pocinju sa sa
INTERAKCIJA SA PROGRAMOM:
Broj reci koje pocinju na sa je 3.
```

*Primer 3*

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

*Primer 4*

```
POZIV: ./a.out ulaz.txt
ULAZ.TXT
Ovo je sadržaj ulaza.
INTERAKCIJA SA PROGRAMOM:
Greska: Nedovoljan broj argumenata
komandne linije.
Program se poziva sa
./a.out ime_dat suf/pref -s/-p.
```

[Rešenje 2.7]

## 2.2 Višedimenzioni nizovi

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

- Napisati funkciju koja izračunava trag matrice (sumu elemenata na glavnoj dijagonali).
- Napisati funkciju koja izračunava euklidsku normu matrice (koren sume kvadrata svih elemenata).

- (c) Napisati funkciju koja izračunava gornju vandijagonalnu normu matrice (sumu apsolutnih vrednosti elemenata iznad glavne dijagonale).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimanziju kvadratne matrice  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrice. Na standardni izlaz ispisati učitane matricu, a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.8]

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

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

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

*Primer 1*

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

[Rešenje 2.9]

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

- Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.
- Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).
- Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).



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

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

*Primer 1*

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

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

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

[Rešenje 2.10]

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

- Napisati funkciju koja određuje najveći element matrice na sporednoj dijagonali.
- Napisati funkciju koja određuje indeks kolone koja sadrži najmanji element matrice.
- Napisati funkciju koja određuje indeks vrste koja sadrži najveći element matrice.
- Napisati funkciju koja određuje broj negativnih elemenata matrice.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati elemente celobrojne kvadratne matrice čija se dimenzija  $n$  ( $0 < n \leq 32$ ) zadaje kao argument komandne linije. Na standardni izlaz ispisati rezultat primene prethodno napisanih funkcija.

*Primer 1*

```
Poziv: ./a.out 3

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

*Primer 2*

```
Poziv: ./a.out 4

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

[Rešenje 2.11]

**Zadatak 2.12** Napisati funkciju kojom se proverava da li je zadata kvadratna matrica dimenzije  $n$  ortonormirana. Matrica je ortonormirana ako je skalarni proizvod svakog para različitih vrsta jednak nuli, a skalarni proizvod vrste sa samom sobom jednak jedinici. Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju celobrojne kvadratne matrice  $n$

( $0 < n \leq 32$ ), a zatim i njene elemente. Na standardni izlaz ispisati rezultat primene napisane funkcije na učitanoj matrici.

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.12]

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

- Napisati funkciju koja učitava elemente matrice sa standardnog ulaza
- Napisati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenzije matrice  $n$  ( $0 < n \leq 10$ ) i  $m$  ( $0 < m \leq 10$ ), a zatim i elemente matrice. Na standardni izlaz spiralno ispisati elemente učitane matrice.

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.13]

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

*Primer 1*

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

## 2.3 Dinamička alokacija memorije

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

## Primer 1

```

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

```

## Primer 2

```

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

```

[Rešenje 2.15]

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

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

Od korisnika sa ulaza tražiti da izabere način realokacije memorije.

## Primer 1

```

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

```

## Primer 2

```

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

```

[Rešenje 2.16]

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

## Primer 1

```

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

```

## Primer 2

```

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

```

[Rešenje 2.17]

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

## Primer 1

```

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

```

## Primer 2

```

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

```

[Rešenje 2.18]

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

- Napisati funkciju `int **alociraj_matricu(int n, int m)` koja dinamički alocira memoriju potrebnu za matricu dimenzija  $n \times m$ .
- Napisati funkciju `int **alociraj_kvadratnu_matricu(int n)` koja alocira memoriju za kvadratnu matricu dimenzije  $n$ .

- (c) Napisati funkciju `int **deallociraj_matricu(int **A, int n)` koja dealocira memoriju matrice sa  $n$  vrsta. Povratna vrednost ove funkcije treba da bude "prazna" matrica.
- (d) Napisati funkciju `void ucitaj_matricu(int **A, int n, int m)` koja učitava već alociranu matricu dimenzija  $n \times m$  sa standardnog ulaza.
- (e) Napisati funkciju `void ucitaj_kvadratnu_matricu(int **A, int n)` koja učitava već alociranu kvadratnu matricu dimenzije  $n$  sa standardnog ulaza.
- (f) Napisati funkciju `void ispisi_matricu(int **A, int n, int m)` koja ispisuje matricu dimenzija  $n \times m$  na standardnom izlazu.
- (g) Napisati funkciju `void ispisi_kvadratnu_matricu(int **A, int n)` koja ispisuje kvadratnu matricu dimenzije  $n$  na standardnom izlazu.
- (h) Napisati funkciju `int ucitaj_matricu_iz_datoteke(int **A, int n, int m, FILE * f)` koja učitava već alociranu matricu dimenzija  $n \times m$  iz već otvorene datoteke  $f$ . U slučaju neuspešnog učitavanja vratiti vrednost različitu od 0.
- (i) Napisati funkciju `int ucitaj_kvadratnu_matricu_iz_datoteke(int **A, int n, FILE * f)` koja učitava već alociranu kvadratnu matricu dimenzije  $n$  iz već otvorene datoteke  $f$ . U slučaju neuspešnog učitavanja vratiti vrednost različitu od 0.
- (j) Napisati funkciju `int upisi_matricu_u_datoteku(int **A, int n, int m, FILE * f)` koja upisuje matricu dimenzija  $n \times m$  u već otvorenu datoteku  $f$ . U slučaju neuspešnog upisivanja vratiti vrednost različitu od 0.
- (k) Napisati funkciju `int upisi_kvadratnu_matricu_u_datoteku(int **A, int n, FILE * f)` koja upisuje kvadratnu matricu dimenzije  $n$  u već otvorenu datoteku  $f$ . U slučaju neuspešnog upisivanja vratiti vrednost različitu od 0.

Napisati programe koji testiraju napisanu biblioteku.

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

#### Primer 1

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

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

#### Primer 2

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

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

#### Test 1

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

ULAZ.TXT
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

IZLAZ:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

#### Test 2

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

ULAZ.TXT
dimenzija: 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

IZLAZ:
Neispravan pocetak fajla
```

#### Test 3

```
POZIV: ./a.out

IZLAZ:
Koriscenje programa:
./a.out datoteka
```

[Rešenje 2.19]

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

Primer 1

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

[Rešenje 2.20]

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

Primer 1

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

Primer 2

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

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

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

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

Primer 1

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

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

[Rešenje 2.22]

**Zadatak 2.23** Napisati program koji na osnovu dve realne matrice dimenzija  $m \times n$  formira matricu dimenzije  $2 \cdot m \times n$  tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu

druge matrice. Matrice su zapisane u datoteci `matrice.txt`. U prvom redu se nalaze dimenzije matrica  $m$  i  $n$ , u narednih  $m$  redova se nalaze vrste prve matrice, a u narednih  $n$  redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz.

#### Primer 1

```

Poziv: ./a.out matrice.txt

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

INTERAKCIJA SA PROGRAMOM:
1.1 -2.2 3.3
-1.1 2.2 -3.3
-4.4 5.5 -6.6
4.4 -5.5 6.6
7.7 -8.8 9.9
-7.7 8.8 -9.9

```

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

#### Primer 1

```

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

```

#### Primer 2

```

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

```

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

`redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`

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

#### Primer 1

```

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

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

```

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

#### Primer 1

```

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

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

```

## Primer 2

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

## INTERAKCIJA SA PROGRAMOM:

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

## 2.4 Pokazivači na funkcije

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

## Primer 1

```
POZIV: ./a.out sin

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

## Primer 2

```
POZIV: ./a.out cos

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

[Rešenje 2.27]

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

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

## 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 2

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

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

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

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

#### Primer 1

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

#### Primer 2

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

## 2.5 Rešenja

### Rešenje 2.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija obrće elemente niza koriscenjem indekse sintakse */
7 void obrni_niz_v1(int a[], int n)
8 {
9     int i, j;
10
11     for (i = 0, j = n - 1; i < j; i++, j--) {
12         int t = a[i];
13         a[i] = a[j];
14         a[j] = t;
15     }
16 }
17
18 /* Funkcija obrće elemente niza koriscenjem pokazivacke sintakse */
19 void obrni_niz_v2(int *a, int n)
20 {
21     /* Pokazivaci na elemente niza */
22     int *prvi, *poslednji;
23
24     /* Vrsi se obrtanje niza */
25     for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
26         int t = *prvi;
27
28         /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
29          vrednost koja se nalazi na adresi na koju pokazuje pokazivac
30          "poslednji". Nakon toga se pokazivac "prvi" uvecava za jedan
31          sto za posledicu ima da "prvi" pokazuje na sledeci element u
32          nizu */
33         *prvi++ = *poslednji;
34
35         /* Vrednost promenljive "t" se postavlja na adresu na koju
36          pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
37          umanjuje za jedan, sto za posledicu ima da pokazivac
38          "poslednji" sada pokazuje na element koji mu prethodi u nizu */
39         *poslednji-- = t;
40     }
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;
    *prvi = *poslednji;
49     *poslednji = t;
    }
51     *****/
}
53
54 int main()
55 {
    /* Deklarise se niz od najvise MAX elemenata */
56     int a[MAX];
57
    /* Broj elemenata niza a */
58     int n;
59
    /* Pokazivac na elemente niza */
60     int *p;
61
62     printf("Unesite dimenziju niza: ");
63     scanf("%d", &n);
64
65     /* Proverava se da li je doslo do prekoračenja ograničenja
66        dimenzije */
67     if (n <= 0 || n > MAX) {
68         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
69         exit(EXIT_FAILURE);
70     }
71
72     printf("Unesite elemente niza:\n");
73     for (p = a; p - a < n; p++)
74         scanf("%d", p);
75
76     obrni_niz_v1(a, n);
77
78     printf("Nakon obrtanja elemenata, niz je:\n");
79
80     for (p = a; p - a < n; p++)
81         printf("%d ", *p);
82     printf("\n");
83
84     obrni_niz_v2(a, n);
85
86     printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
87
88     for (p = a; p - a < n; p++)
89         printf("%d ", *p);
90     printf("\n");
91
92     exit(EXIT_SUCCESS);
93 }

```

## Rešenje 2.2

```

#include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija izracunava zbir elemenata niza */
7 double zbir(double *a, int n)
8 {
9     double s = 0;
10    int i;
11
12    for (i = 0; i < n; i++) s += *(a + i);
13
14    return s;
15 }
16
17 /* Funkcija izracunava proizvod elemenata niza */
18 double proizvod(double *a, int n)
19 {

```

```

20     double p = 1;

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

24

26     return p;
27 }

28 /* Funkcija izracunava minimalni element niza */
double min(double *a, int n)
29 {
30     /* Na pocetku, minimalni element je prvi element */
31     double min = *a;
32     int i;

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

40     return min;
41 }

42

44 /* Funkcija izracunava maksimalni element niza */
double max(double *a, int n)
45 {
46     /* Na pocetku, maksimalni element je prvi element */
47     double max = *a;

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

56     return max;
57 }

58

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

64

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

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

74

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

78

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

84

85     exit(EXIT_SUCCESS);
86 }

```

### Rešenje 2.3

```

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

```

```

3 #define MAX 100

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

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

```

## Rešenje 2.4

```

1 #include <stdio.h>

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

```

```

char tip_ispisa;

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

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

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

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

35 printf("Pocetna slova argumenata komandne linije:\n");
37 if (tip_ispisa == 'I') {
    /* koristeci indeksnu sintaksu */
39     for (i = 0; i < argc; i++)
        printf("%c ", argv[i][0]);
41     printf("\n");
} else if (tip_ispisa == 'P') {
    /* koristeci pokazivacku sintaksu */
43     for (i = 0; i < argc; i++)
        printf("%c ", **argv++);
45     printf("\n");
47 }

49 return 0;
}

```

## Rešenje 2.5

```

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

5 /* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
   cita spreda i odpozadi */
7 int palindrom(char *niska)
{
9     int i, j;
    for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
11         if (*(niska + i) != *(niska + j))
            return 0;
13     return 1;
}

15 int main(int argc, char **argv)
{
17     int i, n = 0;

19     /* Multi argument komandne linije je ime izvrsnog programa */
21     for (i = 1; i < argc; i++)
        if (palindrom(*(argv + i)))
23         n++;
}

```

```

25     printf
        ("Broj argumenata komandne linije koji su palindromi je %d.\n",
27         n);
    return 0;
29 }

```

## Rešenje 2.6

```

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

4 #define MAX_KARAKTERA 100

6 /* Implementacija funkcije strlen() iz standardne biblioteke */
int duzina(char *s)
8 {
    int i;
10    for (i = 0; *(s + i); i++);
    return i;
12 }

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

20     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska */
    if (argc < 3) {
22         printf("Greska: ");
        printf("Medovoljan broj argumenata komandne linije.\n");
24         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
            argv[0]);
        exit(EXIT_FAILURE);
26     }

28     /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
        komandne linije. */
    in = fopen(*(argv + 1), "r");
30    if (in == NULL) {
        fprintf(stderr, "Greska: ");
32        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
34    }

36    n = atoi(*(argv + 2));

40    /* Broje se reci cija je duzina jednaka broju zadatom drugim
        argumentom komandne linije */
    while (fscanf(in, "%s", rec) != EOF)
42        if (duzina(rec) == n)
            br++;
44

46    printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);

48    /* Zatvara se datoteka */
    fclose(in);
50
    exit(EXIT_SUCCESS);
52 }

```

## Rešenje 2.7

```

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

4 #define MAX_KARAKTERA 100

6 /* Implementacija funkcije strcpy() iz standardne biblioteke */
void kopiranje_niske(char *dest, char *src)

```

```

8 {
    int i;
10    for (i = 0; *(src + i); i++)
        *(dest + i) = *(src + i);
12 }

14 /* Implementacija funkcije strcmp() iz standardne biblioteke */
int poredjenje_niski(char *s, char *t)
16 {
    int i;
18    for (i = 0; *(s + i) == *(t + i); i++)
        if (*(s + i) == '\0')
            return 0;
20    return *(s + i) - *(t + i);
22 }

24 /* Implementacija funkcije strlen() iz standardne biblioteke */
int duzina_niske(char *s)
26 {
    int i;
28    for (i = 0; *(s + i); i++);
    return i;
30 }

32 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
    sufiks niske zadate prvi argumentom funkcije */
int sufiks_niske(char *niska, char *sufiks)
34 {
    int duzina_sufiksa = duzina_niske(sufiks);
    int duzina_niske_pom = duzina_niske(niska);
36    if (duzina_sufiksa <= duzina_niske_pom &&
        poredjenje_niski(niska + duzina_niske_pom -
40        duzina_sufiksa, sufiks) == 0)
        return 1;
42    return 0;
44 }

46 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
    prefiks niske zadate prvi argumentom funkcije */
int prefiks_niske(char *niska, char *prefiks)
48 {
    int i;
50    int duzina_prefiksa = duzina_niske(prefiks);
    int duzina_niske_pom = duzina_niske(niska);
52    if (duzina_prefiksa <= duzina_niske_pom) {
        for (i = 0; i < duzina_prefiksa; i++)
54            if (*(prefiks + i) != *(niska + i))
                return 0;
56        return 1;
    } else
58        return 0;
60 }

int main(int argc, char **argv)
62 {
    /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
64     greska */
    if (argc < 4) {
66        printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
68        printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
            argv[0]);
70        exit(EXIT_FAILURE);
    }

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

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

```

```

82     exit(EXIT_FAILURE);
83 }
84 /* Provera se opcija kojom je pozvan program a zatim se učitavaju
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 završavaju 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 počinju na %s je %d.\n", *(argv + 2), br);
96 }
97
98 fclose(in);
99
100 exit(EXIT_SUCCESS);
101 }

```

## Rešenje 2.8

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4
5  #define MAX 100
6
7  /* Funkcija izracunava trag matrice */
8  int trag(int M[][MAX], int n)
9  {
10     int trag = 0, i;
11     for (i = 0; i < n; i++)
12         trag += M[i][i];
13     return trag;
14 }
15
16 /* Funkcija izracunava euklidsku normu matrice */
17 double euklidska_norma(int M[][MAX], int n)
18 {
19     double norma = 0.0;
20     int i, j;
21
22     for (i = 0; i < n; i++)
23         for (j = 0; j < n; j++)
24             norma += M[i][j] * M[i][j];
25
26     return sqrt(norma);
27 }
28
29 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
30 int gornja_vandijagonalna_norma(int M[][MAX], int n)
31 {
32     int norma = 0;
33     int i, j;
34
35     for (i = 0; i < n; i++) {
36         for (j = i + 1; j < n; j++)
37             norma += abs(M[i][j]);
38     }
39
40     return norma;
41 }
42
43 int main()
44 {
45     int A[MAX][MAX];
46     int i, j, n;
47
48     printf("Unesite dimenziju matrice: ");

```

```

49     scanf("%d", &n);

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

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

69     /*****
70     Ispisuju se elementi matrice koriscenjem pokazivacke sintakse.
71     Kod ovako definisane matrice, elementi su uzastopno smesteni u
72     memoriju, kao na traci. To znaci da su svi elementi prve vrste
73     redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
74     prve vrste smesten je prvi element druge vrste za kojim slede
75     svi elementi te vrste i tako dalje redom.
76     *****/

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

83     /*****

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

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

93     exit(EXIT_SUCCESS);
94 }

```

## 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 }

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++) {

```



```

    for (j = 0; j < n; j++)
25         printf("%d ", m[i][j]);
        printf("\n");
27     }
}
29
/* Funkcija proverava da li su zadate kvadratne matrice a i b
31   dimenzije n jednake */
int jednake_matrice(int a[][MAX], int b[][MAX], int n)
33 {
    int i, j;
35
    for (i = 0; i < n; i++)
37         for (j = 0; j < n; j++)
            if (a[i][j] != b[i][j])
39                 return 0;

41     /* Prosla je provera jednakosti za sve parove elemenata koji su na
        istim pozicijama. To znaci da su matrice jednake */
43     return 1;
}
45
/* Funkcija izracunava zbir dve kvadratne matrice */
47 void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
{
49     int i, j;

51     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
53         c[i][j] = a[i][j] + b[i][j];
}
55

/* Funkcija izracunava proizvod dve kvadratne matrice */
57 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
{
59     int i, j, k;

61     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
63         /* Mnozi se i-ta vrsta prve sa j-tom kolonom druge matrice */
        c[i][j] = 0;
65         for (k = 0; k < n; k++)
            c[i][j] += a[i][k] * b[k][j];
67     }
}
69

int main()
71 {
    /* Matrice ciji se elementi zadaju sa ulaza */
73     int a[MAX][MAX], b[MAX][MAX];

75     /* Matrice zbira i proizvoda */
    int zbir[MAX][MAX], proizvod[MAX][MAX];
77
    /* Dimenzija matrica */
79     int n;

81     printf("Unesite dimenziju matrica:\n");
    scanf("%d", &n);
83

    /* Proverava se da li je doslo do prekoračenja dimenzije */
85     if (n > MAX || n <= 0) {
        fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
87         fprintf(stderr, "matrica.\n");
        exit(EXIT_FAILURE);
89     }

91     printf("Unesite elemente prve matrice, vrstu po vrstu:\n");
    ucitaj_matricu(a, n);
93     printf("Unesite elemente druge matrice, vrstu po vrstu:\n");
    ucitaj_matricu(b, n);
95

    /* Izracunava se zbir i proizvod matrica */

```

```

97  saberi(a, b, zbir, n);
    pomnozi(a, b, proizvod, n);

99

101 /* Ispisuje se rezultat */
    if (jednake_matrice(a, b, n) == 1)
        printf("Matrice su jednake.\n");
103 else
        printf("Matrice nisu jednake.\n");

105
107 printf("Zbir matrica je:\n");
    ispisi_matricu(zbir, n);

109 printf("Proizvod matrica je:\n");
    ispisi_matricu(proizvod, n);

111
113 exit(EXIT_SUCCESS);
}

```

### Rešenje 2.10

```

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

#define MAX 64

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

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

    return 1;
}

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

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

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

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

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

```

```

54     return 1;
55 }
56
57 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono je
58    odredjeno matricom koja sadrzi sve elemente polazne matrice
59    dopunjene tako da matrica postane simetricna u odnosu na glavnu
60    dijagonalu */
61 void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
62 {
63     int i, j;
64
65     for (i = 0; i < n; i++)
66         for (j = 0; j < n; j++)
67             zatvorenje[i][j] = m[i][j];
68
69     for (i = 0; i < n; i++)
70         for (j = 0; j < n; j++)
71             if (zatvorenje[i][j] == 1)
72                 zatvorenje[j][i] = 1;
73 }
74
75 /* Funkcija proverava da li je relacija tranzitivna. Relacija je
76    tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
77    relaciji sa elementom "j" i element "j" u relaciji sa elementom
78    "k", onda je i element "i" u relaciji sa elementom "k" */
79 int tranzitivnost(int m[][MAX], int n)
80 {
81     int i, j, k;
82
83     for (i = 0; i < n; i++)
84         for (j = 0; j < n; j++)
85             /* Ispituje se da li postoji element koji narušava *
86                tranzitivnost */
87             for (k = 0; k < n; k++)
88                 if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
89                     return 0;
90
91     return 1;
92 }
93
94
95 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
96    relacije koriscenjem Varsalovog algoritma */
97 void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
98 {
99     int i, j, k;
100
101     /* Prepisuju se vrednosti elemenata pocetne matrice */
102     for (i = 0; i < n; i++)
103         for (j = 0; j < n; j++)
104             zatvorenje[i][j] = m[i][j];
105
106     /* Odredjuje se reflektivno zatvorenje matrice */
107     for (i = 0; i < n; i++)
108         zatvorenje[i][i] = 1;
109
110     /* Primenom Varsalovog algoritma odredjuje se tranzitivno
111        zatvorenje matrice */
112     for (k = 0; k < n; k++)
113         for (i = 0; i < n; i++)
114             for (j = 0; j < n; j++)
115                 if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
116                     && (zatvorenje[i][j] == 0))
117                     zatvorenje[i][j] = 1;
118 }
119
120 /* Funkcija ispisuje elemente matrice */
121 void pisi_matricu(int m[][MAX], int n)
122 {
123     int i, j;

```

```

126     for (i = 0; i < n; i++) {
127         for (j = 0; j < n; j++)
128             printf("%d ", m[i][j]);
129         printf("\n");
130     }
131 }
132
133 int main(int argc, char *argv[])
134 {
135     FILE *ulaz;
136     int m[MAX][MAX];
137     int pomocna[MAX][MAX];
138     int n, i, j;
139
140     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska */
141     if (argc < 2) {
142         printf("Greska: ");
143         printf("Nedovoljan broj argumenata komandne linije.\n");
144         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
145         exit(EXIT_FAILURE);
146     }
147
148     /* Otvara se datoteka za citanje */
149     ulaz = fopen(argv[1], "r");
150     if (ulaz == NULL) {
151         fprintf(stderr, "Greska: ");
152         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
153         exit(EXIT_FAILURE);
154     }
155
156     /* Ucitava se dimenzija matrice */
157     fscanf(ulaz, "%d", &n);
158
159     /* Proverava se da li je doslo do prekoračenja dimenzije */
160     if (n > MAX || n <= 0) {
161         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
162         fprintf(stderr, "matrice.\n");
163         exit(EXIT_FAILURE);
164     }
165
166     /* Ucitava se element po element matrice */
167     for (i = 0; i < n; i++)
168         for (j = 0; j < n; j++)
169             fscanf(ulaz, "%d", &m[i][j]);
170
171     /* Ispisuje se rezultat */
172     printf("Relacija %s refleksivna.\n",
173           refleksivnost(m, n) == 1 ? "jeste" : "nije");
174
175     printf("Relacija %s simetricna.\n",
176           simetricnost(m, n) == 1 ? "jeste" : "nije");
177
178     printf("Relacija %s tranzitivna.\n",
179           tranzitivnost(m, n) == 1 ? "jeste" : "nije");
180
181     printf("Refleksivno zatvorenje relacije:\n");
182     ref_zatvorenje(m, n, pomocna);
183     pisi_matricu(pomocna, n);
184
185     printf("Simetricno zatvorenje relacije:\n");
186     sim_zatvorenje(m, n, pomocna);
187     pisi_matricu(pomocna, n);
188
189     printf("Refleksivno-tranzitivno zatvorenje relacije:\n");
190     ref_tran_zatvorenje(m, n, pomocna);
191     pisi_matricu(pomocna, n);
192
193     /* Zatvara se datoteka */
194     fclose(ulaz);
195
196     exit(EXIT_SUCCESS);
197 }

```

## Rešenje 2.11

```

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

#define MAX 32

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

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

    return max_na_sporednoj_dijagonali;
}

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

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

    return indeks_kolone;
}

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

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (m[i][j] > max) {
                max = m[i][j];
                indeks_vrste = i;
            }

    return indeks_vrste;
}

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

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

    return broj_negativnih;
}

int main(int argc, char *argv[])
{
    int m[MAX][MAX];
    int n;
    int i, j;

```

```

72  /* Proverava se broj argumenata komandne linije */
    if (argc < 2) {
74      printf("Greska: ");
      printf("Nedovoljan broj argumenata komandne linije.\n");
76      printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
      exit(EXIT_FAILURE);
78    }

80  /* Ucitava se vrednost dimenzije i proverava se njena korektnost */
    n = atoi(argv[1]);
82

    if (n > MAX || n <= 0) {
84      fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
      fprintf(stderr, "matrice.\n");
86      exit(EXIT_FAILURE);
    }
88

    /* Ucitava se matrica */
90    printf("Unesite elemente matrice dimenzije %d:\n", n);
    for (i = 0; i < n; i++)
92      for (j = 0; j < n; j++)
        scanf("%d", &m[i][j]);
94

    printf("Najveci element sporedne dijagonale je %d.\n",
96      max_sporedna_dijagonala(m, n));

98    printf("Indeks kolone sa najmanjim elementom je %d.\n",
        indeks_min(m, n));
100

    printf("Indeks vrste sa najvećim elementom je %d.\n",
102      indeks_max(m, n));

104    printf("Broj negativnih elemenata matrice je %d.\n",
        broj_negativnih(m, n));
106

    exit(EXIT_SUCCESS);
108 }

```

### Rešenje 2.12

```

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

11     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)
34 {
35     int i, j, k;
36     int proizvod;
37
38     /* Ispituje se uslov normiranosti */
39     for (i = 0; i < n; i++) {
40         proizvod = 0;
41
42         for (j = 0; j < n; j++)
43             proizvod += m[i][j] * m[i][j];
44
45         if (proizvod != 1)
46             return 0;
47     }
48
49     /* Ispituje se uslov ortogonalnosti */
50     for (i = 0; i < n - 1; i++) {
51         for (j = i + 1; j < n; j++) {
52
53             proizvod = 0;
54
55             for (k = 0; k < n; k++)
56                 proizvod += m[i][k] * m[j][k];
57
58             if (proizvod != 0)
59                 return 0;
60         }
61     }
62
63     /* Ako su oba uslova ispunjena, matrica je ortonormirana */
64     return 1;
65 }
66
67 int main()
68 {
69     int A[MAX][MAX];
70     int n;
71
72     printf("Unesite dimenziju matrice: ");
73     scanf("%d", &n);
74
75     if (n > MAX || n <= 0) {
76         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
77         fprintf(stderr, "matrice.\n");
78         exit(EXIT_FAILURE);
79     }
80
81     printf("Unesite elemente matrice, vrstu po vrstu:\n");
82     ucitaj_matricu(A, n);
83
84     printf("Matrica %s ortonormirana.\n",
85           ortonormirana(A, n) ? "je" : "nije");
86
87     exit(EXIT_SUCCESS);
88 }

```

### Rešenje 2.13

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_V 10
5  #define MAX_K 10
6
7  /* Funkcija proverava da li su ispisani svi elementi iz matrice,
8     odnosno da li se nariusio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
10 {
11     return !(top <= bottom && left <= right);
12 }
13

```

```

/* Funkcija spiralno ispisuje elemente matrice */
15 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
{
17     int i, j, top, bottom, left, right;

19     top = left = 0;
    bottom = n - 1;
21     right = m - 1;

23     while (!krajIspisa(top, bottom, left, right)) {

25         for (j = left; j <= right; j++)
            printf("%d ", a[top][j]);

27         /* Spusta se prvi red */
29         top++;

31         if (krajIspisa(top, bottom, left, right))
            break;

33         for (i = top; i <= bottom; i++)
35             printf("%d ", a[i][right]);

37         /* Pomera se desna kolona za naredni krug ispisa blize levom
            kraju */
39         right--;

41         if (krajIspisa(top, bottom, left, right))
            break;

43         /* Ispisuje se donja vrsta */
45         for (j = right; j >= left; j--)
            printf("%d ", a[bottom][j]);

47         /* Podize se donja vrsta za naredni krug ispisa */
49         bottom--;

51         if (krajIspisa(top, bottom, left, right))
            break;

53         /* Ispisuje se prva kolona */
55         for (i = bottom; i >= top; i--)
            printf("%d ", a[i][left]);

57         /* Priprema se leva kolona za naredni krug ispisa */
59         left++;
    }
61     putchar('\n');
}

63 /* Funkcija učitava matricu */
65 void učitaj_matricu(int a[][MAX_K], int n, int m)
{
67     int i, j;

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

73 int main()
75 {
    int a[MAX_V][MAX_K];
77     int m, n;

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

81     if (n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
83         fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
        fprintf(stderr, "matrice.\n");
85         exit(EXIT_FAILURE);
    }
}

```



```

87     printf("Unesite elemente matrice, vrstu po vrstu:\n");
89     ucitaj_matricu(a, n, m);

91     printf("Spiralno ispisana matrica: ");
    ispisi_matricu_spiralno(a, n, m);
93
    exit(EXIT_SUCCESS);
95 }

```

### Rešenje 2.15

```

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

3
   int main()
5  {
       int *p = NULL;
       int i, n;

9     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     }

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

23     printf("Niz u obrnutom poretku je: ");
    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;

10     /* Velicina alocirane memorije */
    int alocirano;

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

16     /* Broj koji se učitava sa ulaza */
    int x;
    int i;
18     int *b = NULL;
    char realokacija;

20
22     /* Inicijalizacija */
    alocirano = n = 0;

```

```

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

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

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

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

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

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

50             free(a);

52             /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
53              bloka koji je prilikom alokacije zapamcen u promenljivoj b
54              */
55             a = b;
56         } else if (realokacija == 'R') {

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

60             a = (int *) realloc(a, alocirano * sizeof(int));
61             if (a == NULL) {
62                 fprintf(stderr, "realloc(): ");
63                 fprintf(stderr, "greska pri alokaciji memorije.\n");
64                 exit(EXIT_FAILURE);
65             }

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

84 exit(EXIT_SUCCESS);
}

```

## Rešenje 2.17

```

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

#define MAX 1000

/* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat

```

```

8     nadovezivanja niski. Adresa niza se vraća kao povratna vrednost. */
char *nadovezi(char *s, char *t)
10 {
12     char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
                               * sizeof(char));

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

20     /* Kopiraju se i nadovezuju niske karaktera */
22     strcpy(p, s);
    strcat(p, t);
24
    return p;
26 }

28 int main()
{
30     char *s = NULL;
    char s1[MAX], s2[MAX];
32
    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 dinamički alociran niz pokazivaca na vrste matrice */
    double **A = NULL;
11
    /* Broj vrsta i broj kolona */
13     int n = 0, m = 0;

15     /* Trag matrice */
    double trag = 0;
17
    printf("Unesite broj vrsta i broj kolona matrice:\n ");
19     scanf("%d%d", &n, &m);

21     /* Dinamički se alocira prostor za n pokazivaca na double */
    A = malloc(sizeof(double *) * n);
23

    /* Provera se da li je doslo do greske pri alokaciji */
25     if (A == NULL) {
        fprintf(stderr, "malloc(): ");
27         fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
29     }

```

```

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

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

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

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

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

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

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

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

59  exit(EXIT_SUCCESS);
60 }

```

### Rešenje 2.19

Datoteka 2.1: *matrica.h*

```

1  #ifndef _MATRICA_H
2  #define _MATRICA_H 1
3
4  /* Funkcija dinamicki alokira memoriju za matricu dimenzija n x m */
5  int **alociraj_matricu(int n, int m);
6
7  /* Funkcija dinamicki alokira memoriju za kvadratnu matricu dimenzije
8     n */
9  int **alociraj_kvadratnu_matricu(int n);
10
11 /* Funkcija dealocira memoriju za matricu sa n vrsta */
12 int **dealociraj_matricu(int **matrica, int n);
13
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
18 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n sa
19    standardnog ulaza */
20 void ucitaj_kvadratnu_matricu(int **matrica, int n);
21
22 /* Funkcija ispisuje matricu dimenzija n x m na standardnom izlazu */
23 void ispisi_matricu(int **matrica, int n, int m);
24
25 /* Funkcija ispisuje kvadratnu matricu dimenzije n na standardnom
26    izlazu */
27 void ispisi_kvadratnu_matricu(int **matrica, int n);

```

```

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

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

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

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

45 #endif

```

Datoteka 2.2: *matrica.c*

```

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

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

9      /* Alocira se prostor za niz vrsti matrice */
11     matrica = (int **) malloc(n * sizeof(int *));
       /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije ce
13     biti NULL, sto mora biti provereno u main funkciji */
       if (matrica == NULL)
15         return NULL;

17     /* Alocira se prostor za svaku vrstu matrice */
       for (i = 0; i < n; i++) {
19         matrica[i] = (int *) malloc(m * sizeof(int));
           /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
21         prethodno alocirani resursi, i povratna vrednost je NULL */
           if (matrica[i] == NULL) {
23             for (j = 0; j < i; j++)
                 free(matrica[j]);
25             free(matrica);
               return NULL;
27         }
       }
29     return matrica;
   }

31 int **alociraj_kvadratnu_matricu(int n)
   {
33     /* Alociranje matrice dimenzije n x n */
35     return alociraj_matricu(n, n);
   }

37 int **dealociraj_matricu(int **matrica, int n)
   {
39     int i;
       /* Oslobadja se prostor rezervisan za svaku vrstu */
       for (i = 0; i < n; i++)
43         free(matrica[i]);
       /* Oslobadja se memorija za niz pokazivaca na vrste */
45     free(matrica);

47     /* Matrica postaje prazna, tj. nealocirana */
       return NULL;
49 }

51 void ucitaj_matricu(int **matrica, int n, int m)

```

```

{
53     int i, j;
    /* Elementi matrice se učitavaju po vrstama */
55     for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
57         scanf("%d", &matrica[i][j]);
}

59 void ucitaj_kvadratnu_matricu(int **matrica, int n)
61 {
    /* Učitavanje matrice n x n */
63     ucitaj_matricu(matrica, n, n);
}

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

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

83 int ucitaj_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 */
91         if (fscanf(f, "%d", &matrica[i][j]) != 1)
            return 1;

93     /* Uspešno učitana matrica */
95     return 0;
}

97 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
99     FILE * f)
101 {
    /* Učitavanje matrice n x n iz datoteke */
    return ucitaj_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 */
113         if (fprintf(f, "%d ", matrica[i][j]) <= 0)
            return 1;
        fprintf(f, "\n");
115     }

117     /* Uspešno upisana matrica */
119     return 0;
}

121 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n, FILE * f)
123 {
    /* Ispis matrice n x n u datoteku */

```

```

125 return upisi_matricu_u_datoteku(matrica, n, n, f);
    }

```

Datoteka 2.3: *main\_a.c*

```

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

int main()
{
    int **matrica = NULL;
    int n, m;
    FILE *f;

    /* Ucitavanje dimenzije matrice */
    printf("Unesi broj vrsta matrice: ");
    scanf("%d", &n);
    printf("Unesi broj kolona matrice: ");
    scanf("%d", &m);

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

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

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

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

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

    /* Zatvaranje fajla */
    fclose(f);

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

    exit(EXIT_SUCCESS);
}

```

Datoteka 2.4: *main\_b.c*

```

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

int main(int argc, char **argv)
{
    int **matrica = NULL;
    int n;

```

```

9 FILE *f;

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

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

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

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

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

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

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

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

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

56 exit(EXIT_SUCCESS);
57 }

```

## Rešenje 2.20

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include "matrica.h"
5
6 /* Funkcija ispisuje elemente matrice ispod glavne dijagonale */
7 void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
8 {
9     int i, j;
10
11     for (i = 0; i < n; i++) {
12         for (j = 0; j <= i; j++)
13             printf("%d ", M[i][j]);
14         printf("\n");
15     }
16 }
17
18 int main()

```



```

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

```

### Rešenje 2.22

```

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

/* Funkcija izvrsava trazene transformacije nad matricom */
void izmeni(float **a, int n)
{
    int i, j;

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

/* 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 */
float zbir_ispod_sporedne_dijagonale(float **m, int n)
{
    int i, j;
    float zbir = 0;

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

    return zbir;
}

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

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

```

```

44 }

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

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

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

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

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

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

82 /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
   n */
84 void oslobodi_memoriju(float **m, int n)
85 {
86     int i;

87     for (i = 0; i < n; i++)
88         free(m[i]);
89     free(m);
90 }

92 int main(int argc, char *argv[])
93 {
94     FILE *ulaz;
95     float **a;
96     int n;

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

104     /* Otvara se datoteka za citanje */
105     ulaz = fopen(argv[1], "r");
106     if (ulaz == NULL) {
107         fprintf(stderr, "Greska: ");
108         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
109         exit(EXIT_FAILURE);
110     }
111 }

```

```

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

120

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

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

126
    float zbir = zbir_ispod_sporedne_dijagonale(a, n);

128

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

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

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

138

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

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

144

    exit(EXIT_SUCCESS);
146 }

```

### Rešenje 2.27

```

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

7 /* Funkcija tabela() prihvata granice intervala a i b, broj
   ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
9   funkciju koja prihvata double argument, i vraca double vrednost.
   Za tako datu funkciju ispisuju se njene vrednosti u intervalu
11  [a,b] u n ekvidistantnih tacaka intervala */
void tabela(double a, double b, int n, double (*fp) (double))
13 {
    int i;
15    double x;

17    printf("-----\n");
    for (i = 0; i < n; i++) {
19        x = a + i * (b - a) / (n - 1);
        printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
21    }
    printf("-----\n");
23 }

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

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

35    char ime_fje[6];

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

```

```

41  /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska */
42  if (argc < 2) {
43      printf("Greska: ");
44      printf("Medovoljan broj argumenata komandne linije.\n");
45      printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
46             argv[0]);
47      exit(EXIT_FAILURE);
48  }
49
50  /* Niska ime_fje sadrzi ime trazene funkcije koja je navedena u
51     komandnoj liniji */
52  strcpy(ime_fje, argv[1]);
53
54  /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
55     */
56  if (strcmp(ime_fje, "sin") == 0)
57      fp = &sin;
58  else if (strcmp(ime_fje, "cos") == 0)
59      fp = &cos;
60  else if (strcmp(ime_fje, "tan") == 0)
61      fp = &tan;
62  else if (strcmp(ime_fje, "atan") == 0)
63      fp = &atan;
64  else if (strcmp(ime_fje, "acos") == 0)
65      fp = &acos;
66  else if (strcmp(ime_fje, "asin") == 0)
67      fp = &asin;
68  else if (strcmp(ime_fje, "exp") == 0)
69      fp = &exp;
70  else if (strcmp(ime_fje, "log") == 0)
71      fp = &log;
72  else if (strcmp(ime_fje, "log10") == 0)
73      fp = &log10;
74  else if (strcmp(ime_fje, "sqrt") == 0)
75      fp = &sqrt;
76  else if (strcmp(ime_fje, "floor") == 0)
77      fp = &floor;
78  else if (strcmp(ime_fje, "ceil") == 0)
79      fp = &ceil;
80  else if (strcmp(ime_fje, "sqr") == 0)
81      fp = &sqr;
82  else {
83      printf("Program jos uvek ne podrzava trazenu funkciju!\n");
84      exit(EXIT_SUCCESS);
85  }
86
87  printf("Unesite krajeve intervala:\n");
88  scanf("%lf %lf", &a, &b);
89
90  printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
91  printf("(ukljucujuci krajeve intervala)?\n");
92  scanf("%d", &n);
93
94  /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
95     uneti broj veci od 2 */
96  if (n < 2) {
97      fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
98      exit(EXIT_FAILURE);
99  }
100
101  /* Ispisuje se ime funkcije */
102  printf("      x %10s(x)\n", ime_fje);
103
104  /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
105     komandne linije */
106  tabela(a, b, n, fp);
107
108  exit(EXIT_SUCCESS);
109 }

```

## Glava 3

# Algoritmi pretrage i sortiranja

### 3.1 Algoritmi pretrage

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

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

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

```
Test 1
Poziv: ./a.out 1000000 23542      VREMENA.TXT
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
Poziv: ./a.out 100000 37842      VREMENA.TXT
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:        Dimenzija niza: 100000
  Element nije u nizu             Linearna: 360803 ns
                                   Binarna: 1187 ns
                                   Interpolaciona: 628 ns
```

[Rešenje 3.1]

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

#### Primer 1

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

#### Primer 2

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

[Rešenje 3.2]

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks ili prezime studenta čije informacije se potom prikazuju na ekranu. U slučaju više studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti `-indeks` ili `-prezime`. U slučaju neuspešnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

#### Primer 1

```
POZIV: ./a.out datoteka.txt -indeks
DATOTEKA.TXT
20140003 Marina Petrovic
20140012 Stefan Mitrovic
20140032 Dejan Popovic
20140049 Mirko Brankovic
20140076 Sonja Stevanovic
20140104 Ivan Popovic
20140187 Vlada Stankovic
20140234 Darko Brankovic
```

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

#### Primer 2

```
POZIV: ./a.out datoteka.txt -prezime
DATOTEKA.TXT
20140003 Marina Petrovic
20140012 Stefan Mitrovic
20140032 Dejan Popovic
20140049 Mirko Brankovic
20140076 Sonja Stevanovic
20140104 Ivan Popovic
20140187 Vlada Stankovic
20140234 Darko Brankovic
```

```
INTERAKCIJA SA PROGRAMOM:
Unesite prezime studenta
cije informacije zelite:
Popovic
Indeks: 20140032,
Ime i prezime: Dejan Popovic
```

[Rešenje 3.3]

**Zadatak 3.4** Modifikovati prethodni zadatak 3.3 tako da tražene funkcije budu rekurzivne.

[Rešenje 3.4]

**Zadatak 3.5** U datoteci koja se zadaje kao prvi argument komandne linije, nalaze se koordinate tačaka. U zavisnosti od prisustva opcija komandne linije (`-x` ili `-y`), pronaći onu koja je najbliža `x`, ili `y` osi, ili koordinatnom početku, ako nije prisutna nijedna opcija. Pretpostaviti da je broj tačaka u datoteci veći od 0 i ne veći od 1024.

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

[Rešenje 3.5]

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

Test 1

```

Izlaz:
1.57031

```

[Rešenje 3.6]

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

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

[Rešenje 3.7]

**Zadatak 3.8** Napisati funkciju koja u opadajuće sortiranom nizu celih brojeva binarnom pretragom pronalazi indeks prvog elementa manjeg od nule. Ukoliko nema elemenata manjih od nule, funkcija kao rezultat vraća -1. Napisati program koji testira ovu funkciju za opadajući niz celih brojeva koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

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

[Rešenje 3.8]

**Zadatak 3.9** Napisati funkciju koja određuje ceo deo logaritma za osnovu 2 datog neoznačenog celog broja koristeći samo bitske i relacione operatore.

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

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

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

[Rešenje 3.9]

**\*\* Zadatak 3.10** U prvom kvadrantu dato je  $1 \leq N \leq 10000$  duži svojim koordinatama (duži mogu da se seku, preklapaju, itd.). Napisati program koji pronalazi najmanji ugao  $0 \leq \alpha \leq 90^\circ$ , na dve decimale, takav da je suma dužina duži sa obe strane polupoluprave iz koordinatnog početka pod uglom  $\alpha$  jednak (neke duži bivaju presečene, a neke ne). Program prvo učitava broj  $N$ , a zatim i same koordinate temena duži. UPUTSTVO: *Vršiti binarnu pretragu intervala  $[0, 90^\circ]$ .*

Primer 1	Primer 2	Primer 3
INTERAKCIJA SA PROGRAMOM: Unesi broj tacaka: 2 Unesi koordinate tacaka: 2 0 2 1 1 2 2 2 26.57	INTERAKCIJA SA PROGRAMOM: Unesi broj tacaka: 2 Unesi koordinate tacaka: 1 0 1 1 0 1 1 1 45	INTERAKCIJA SA PROGRAMOM: Unesi broj tacaka: 3 Unesi koordinate tacaka: 1 0 1 1 2 0 2 1 1 2 2 2 26.57

## 3.2 Algoritmi sortiranja

**Zadatak 3.11** Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom (engl. **selection sort**), sortiranja spajanjem (engl. **merge sort**), brzog sortiranja (engl. **quick sort**), mehurastog sortiranja (engl. **bubble sort**), sortiranja direktnim umetanjem (engl. **insertion sort**) i sortiranja umetanjem sa inkrementom (engl. **shell sort**). Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za sortiranje spajanjem, **-q** za brzo sortiranje, **-b** za mehurasto, **-i** za sortiranje direktnim umetanjem ili **-s** za sortiranje umetanjem sa inkrementom. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati algoritmom sortiranja izborom. Niz koji se sortira generisati neopadajuće ako je prisutna opcija **-r**, nerastuće ako je prisutna opcija **-o** ili potpuno slučajno ako nema nijedne opcije. Vreme meriti programom **time**. Analizirati porast vremena sa porastom dimenzije **n**.

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

[Rešenje 3.11]



**Zadatak 3.12** Dve niske su anagrami ako se sastoje od istog broja istih karaktera. Napisati program koji proverava da li su dve niske karaktera anagrami. Niske se zadaju sa standardnog ulaza i neće biti duže od 127 karaktera. UPUTSTVO: *Napisati funkciju koja sortira slova unutar niske karaktera, a zatim za sortirane niske proveriti da li su identične.*

*Primer 1*

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

*Primer 2*

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

*Primer 3*

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

[Rešenje 3.12]

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

*Test 1*

```
ULAZ:
23 64 123 76 22 7
IZLAZ:
1
```

*Test 2*

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

*Test 3*

```
ULAZ:
34 30
IZLAZ:
4
```

[Rešenje 3.13]

**Zadatak 3.14** Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza i neće biti duži od 256 i kraći od jednog elemenata. UPUTSTVO: *Prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.11.*

*Test 1*

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

*Test 2*

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

*Test 3*

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

[Rešenje 3.14]

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

*Primer 1*

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

*Primer 2*

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

*Primer 3*

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

[Rešenje 3.15]

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

inače vraća 0. Napisati zatim program koji testira ovu funkciju. Nizovi se unose sa standardnog ulaza sve dok se ne unese 0 i može se pretpostaviti da će njihove dimenzije biti manje od 256.

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite elemente prvog niza:
3 6 7 11 14 35 0
Unesite elemente drugog niza:
3 5 8 0
3 3 5 6 7 8 11 14 35
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite elemente prvog niza:
1 4 7 0
Unesite elemente drugog niza:
9 11 23 54 75 0
1 4 7 9 11 23 54 75
```

[Rešenje 3.16]

**Zadatak 3.17** Napisati program koji čita sadržaj dveju datoteka od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima i kreira jedinstven spisak studenata sortiranih takođe po imenu rastuće. Program dobija nazive datoteka iz komandne linije i jedinstveni spisak upisuje u datoteku `ceo-tok.txt`. Pretpostaviti da je ime studenta nije duže od 10, a prezime od 15 karaktera.

#### Test 1

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

PRVI-DEO.TXT
Andrija Petrovic
Anja Ilic
Ivana Markovic
Lazar Micic
Nenad Brankovic
Sofija Filipovic
Vladimir Savic
Uros Milic

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

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

[Rešenje 3.17]

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

#### Test 1

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

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

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

#### Test 2

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

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

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

[Rešenje 3.18]

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

```

Test 1
BIRACKI-SPISAK.TXT
Bojan Golubovic
Andrija Petrovic
Anja Ilic
Aleksandra Cvetic
Dragan Markovic
Ivana Markovic
Lazar Micic
Marija Stankovic
Filip Dukic

IzLAZ:
3

Test 2
BIRACKI-SPISAK.TXT
Milan Milicevic

IzLAZ:
1

Test 3
DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI

IzLAZ:
Problem pri otvaranju
datoteke.
```

[Rešenje 3.19]

**Zadatak 3.20** Definisati strukturu koja čuva imena, prezimena i godišta dece. Pretpostaviti da su imena i prezimena niske karaktera koje nisu duže od 30 karaktera. Napisati funkciju koja sortira niz dece po godištu, a decu istog godišta sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci čije se ime zadaje kao prvi argument komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

```

Test 1
Poziv: ./a.out in.txt out.txt

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

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

Test 2
Poziv: ./a.out in.txt out.txt

IN.OUT
Milijana Maric 2009

OUT.TXT
Milijana Maric 2009
```

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

```

Test 1
NISKE.TXT
ana petar andjela milos nikola aleksandar ljubica matej milica

IzLAZ:
ana matej milos petar milica nikola andjela ljubica aleksandar
```

[Rešenje 3.21]

**Zadatak 3.22** Napisati program koji simulira rad kase u prodavnici. Kupci prilaze kasi, a prodavac unošenjem bar-koda kupljenog proizvoda dodaje njegovu cenu na ukupan račun. Na kraju, program ispisuje ukupnu vrednost svih proizvoda. Sve artikule, 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 3.22]

**Zadatak 3.23** Napisati program koji iz datoteke `aktivnost.txt` čita podatke o aktivnostima studenata na praktikumima i u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt` upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po prezimenu opadajuće. U datoteci se nalazi ime, prezime studenta, broj časova na kojima je prisustvovao, kao i ukupan broj urađenih zadataka. Pretpostaviti da studenata neće biti više od 500 i da je za ime studenta dovoljno 20, a za prezime 25 karaktera.

## Test 1

```

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

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

```

```

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

```

```

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

```

[Rešenje 3.23]

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

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

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

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

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

## Test 1

```

POZIV: ./a.out

PESME.TXT
5
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321
Mika - Kisa, 5341

IZLAZ:
Jelena - Sunce, 92321
Mika - Kisa, 5341
Ana - Nebo, 2342
Pera - Ptice, 327
Laza - Oblaci, 29

```

## Test 2

```

POZIV: ./a.out -i

PESME.TXT
5
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321
Mika - Kisa, 5341

IZLAZ:
Ana - Nebo, 2342
Jelena - Sunce, 92321
Laza - Oblaci, 29
Mika - Kisa, 5341
Pera - Ptice, 327

```

## Test 3

```

POZIV: ./a.out -n

PESME.TXT
5
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321
Mika - Kisa, 5341

IZLAZ:
Mika - Kisa, 5341
Ana - Nebo, 2342
Laza - Oblaci, 29
Pera - Ptice, 327
Jelena - Sunce, 92321

```

[Rešenje 3.24]

**\*\* Zadatak 3.25** Razmatrajmo dve operacije: operacija U je unos novog broja  $x$ , a operacija N određivanje  $n$ -tog po veličini od unetih brojeva. Implementirati program koji izvršava ove operacije. Može postojati najviše 100000 operacija unosa, a uneti elementi se mogu ponavljati, pri čemu se i ponavljanja računaju prilikom brojanja. NAPOMENA: *Brojeve čuvati u sortiranom nizu i svaki naredni element umetati na svoje mesto.* Optimizovati program, ukoliko se zna da neće biti više od 500 različitih unetih brojeva.

*Primer 1*

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

**\*\* Zadatak 3.26** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze najmanja, najveća, itd... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene.

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

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

*Test 1*

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

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

**Zadatak 3.27** Za zadatau celobrojnu matricu dimenzije  $n \times m$  napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrsti. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

*Test 1*

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

*Test 2*

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

[Rešenje 3.27]

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

*Primer 1*

```

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

```

*Primer 2*

```

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

```

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

```

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

```

*Test 1*

```

ULAZ:
R

IZLAZ:
Dusko Radovic

```

*Test 2*

```

ULAZ:
E

IZLAZ:
Dobrica Eric

```

*Test 3*

```

ULAZ:
X

IZLAZ:
-1

```

**Zadatak 3.30** Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova i mogućnost zadavanja različitih kriterijuma sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva, ne veća od 100, a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu. Na standardnom izlazu ispisati odgovarajuću poruku.

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Uneti dimenziju niza: 10
Uneti elemente niza:
5 3 1 6 8 90 34 5 3 432
Sortirani niz u rastucem poretku:
1 3 3 5 5 6 8 34 90 432
Uneti element koji se trazi u nizu: 34
Binarna pretraga:
Element je nadjen na poziciji 7
Linearna pretraga (lfind):
Element je nadjen na poziciji 7

```

*Primer 2*

```

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

```

[Rešenje 3.30]

**Zadatak 3.31** Napisati program koji sa standardnog ulaza učitava dimenziju niza celih brojeva (ne veću od 100), a potom i same elemente niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku prema broju delilaca i tako dobijeni niz odštampati na standardnom izlazu.

#### Primer 1

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

#### Primer 2

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

#### Primer 3

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

[Rešenje 3.31]

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

(a) leksikografski,

(b) po dužini.

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

#### Primer 1

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

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

[Rešenje 3.32]

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

[Rešenje 3.33]

**Zadatak 3.34** Napisati program koji korišćenjem bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova ili poruka ukoliko nema takvog. Potom, sa standardnog ulaza, unosi se prezime traženog studenta i prikazuje se osoba sa tim prezimenom ili poruka da se nijedan student tako ne preziva. Za pretraživanje koristiti odgovarajuće bibliotečke funkcije. Podaci o studentima čitaju se iz datoteke čije se ime zadaje preko argumenata komandne linije. Za svakog studenta u datoteci postoje ime, prezime i bodovi osvojeni na kolokvijumu. Pretpostaviti da neće biti više od 500 studenata i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.



*Primer 1*

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

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

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

```
INTERAKCIJA SA PROGRAMOM:
```

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

[Rešenje 3.34]

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

[Rešenje 3.35]

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

*Primer 1*

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

*Primer 2*

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

*Primer 3*

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

[Rešenje 3.36]

**Zadatak 3.37** Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr15125,mm14001`), ime, prezime i broj poena. Ni ime, ni prezime neće biti duže od 20 karaktera. Napisati 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 smeru, i na kraju na osnovu broja indeksa. Sortirane studente upisati u datoteku `izlaz.txt`. Ukoliko je u komandnoj liniji uz opciju `-n` naveden i nalog nekog studenta, funkcijom `bsearch` potražiti i prijaviti broj poena studenta sa tim nalogom.

*Test 1*

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

```
STUDENTI.TXT
```

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

```
IZLAZ.TXT
```

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

```
IZLAZ:
```

```
mm13321 Marija Radic 12
```

*Test 2*

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

```
STUDENTI.TXT
```

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

```
IZLAZ.TXT
```

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

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

*Primer 1*

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

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

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

## 3.4 Rešenja

## Rešenje 3.1

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

```

    return srednji;
47 }
    /* Ako element x nije pronadjen, vraca se -1 */
49 return -1;
}

51 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
53 pozicije nadjenog elementa ili -1, ako element nije pronadjen */
int interpolaciona_pretraga(int a[], int n, int x)
55 {
    int levi = 0;
57 int desni = n - 1;
    int srednji;
59 /* Dokle god je indeks levi levo od indeksa desni... */
    while (levi <= desni) {
61 /* Ako je trazeni element manji od pocetnog ili veci od
        poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
63 nije u tom delu niza. Ova provera je neophodna, da se ne bi
        dogodilo da se prilikom izracunavanja indeksa srednji izadje
65 izvan opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
67 return -1;
        /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
69 a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
        jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
71 desni). Ova provera je neophodna, jer bi se u suprotnom
        prilikom izracunavanja indeksa srednji pojavilo deljenje
73 nulom. */
        else if (a[levi] == a[desni])
75 return levi;
        /* Racunanje srednjeg indeksa */
77 srednji =
            levi +
79 ((double) (x - a[levi]) / (a[desni] - a[levi])) *
            (desni - levi);
81 /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
        verovatno biti blize trazenoj vrednosti nego da je prosto uvek
83 uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
        porediti sa pretragom recnika: ako neko trazi rec na slovo 'B',
85 sigurno nece da otvori recnik na polovini, vec verovatno negde
        blize pocetku. */
87 /* Ako je element sa indeksom srednji veci od trazenog, tada se
        trazeni element mora nalaziti u levoj polovini niza */
89 if (x < a[srednji])
        desni = srednji - 1;
91 /* Ako je element sa indeksom srednji manji od trazenog, tada se
        trazeni element mora nalaziti u desnoj polovini niza */
93 else if (x > a[srednji])
        levi = srednji + 1;
95 else
97 /* Ako je element sa indeksom srednji jednak trazenom, onda se
        pretraga zavrшава na poziciji srednji */
        return srednji;
99 }
    /* U slucaju neuspesne pretrage vraca se -1 */
101 return -1;
}

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

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

117 /* Dimenzija niza */

```

```

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

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

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

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

146 /* Binarna pretraga */
147 printf("Binarna pretraga:\n");
148 clock_gettime(CLOCK_REALTIME, &time3);
149 i = binarna_pretraga(a, n, x);
150 clock_gettime(CLOCK_REALTIME, &time4);
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 /* Interpolaciona pretraga */
156 printf("Interpolaciona pretraga:\n");
157 clock_gettime(CLOCK_REALTIME, &time5);
158 i = interpolaciona_pretraga(a, n, x);
159 clock_gettime(CLOCK_REALTIME, &time6);
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 /* Podaci o izvršavanju programa bivaju upisani u log fajl */
165 if ((f = fopen("vremena.txt", "a")) == NULL) {
166     fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
167     exit(EXIT_FAILURE);
168 }

169 fprintf(f, "Dimenzija niza: %d\n", n);
170 fprintf(f, "\tLinearna: %10ld ns\n",
171         (time2.tv_sec - time1.tv_sec) * 1000000000 +
172         time2.tv_nsec - time1.tv_nsec);
173 fprintf(f, "\tBinarna: %10ld ns\n",
174         (time4.tv_sec - time3.tv_sec) * 1000000000 +
175         time4.tv_nsec - time3.tv_nsec);
176 fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
177         (time6.tv_sec - time5.tv_sec) * 1000000000 +
178         time6.tv_nsec - time5.tv_nsec);

179 /* Zatvaranje datoteke */
180 fclose(f);

181 exit(EXIT_SUCCESS);
182 }

```

## Rešenje 3.2

```

1  #include <stdio.h>
2
3  #define MAX 1024
4
5  int lin_pretraga_rek_sufiks(int a[], int n, int x)
6  {
7      int tmp;
8      /* Izlaz iz rekurzije */
9      if (n <= 0)
10         return -1;
11     /* Ako je prvi element trazeni */
12     if (a[0] == x)
13         return 0;
14     /* Pretraga ostatka niza */
15     tmp = lin_pretraga_rek_sufiks(a + 1, n - 1, x);
16     return tmp < 0 ? tmp : tmp + 1;
17 }
18
19 int lin_pretraga_rek_prefiks(int a[], int n, int x)
20 {
21     /* Izlaz iz rekurzije */
22     if (n <= 0)
23         return -1;
24     /* Ako je poslednji element trazeni */
25     if (a[n - 1] == x)
26         return n - 1;
27     /* Pretraga ostatka niza */
28     return lin_pretraga_rek_prefiks(a, n - 1, x);
29 }
30
31 int bin_pretraga_rek(int a[], int l, int d, int x)
32 {
33     int srednji;
34     if (l > d)
35         return -1;
36     /* Sredisnja pozicija na kojoj se trazi vrednost x */
37     srednji = (l + d) / 2;
38     /* Ako je element na sredisnjoj poziciji trazeni */
39     if (a[srednji] == x)
40         return srednji;
41     /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
42        pretrazuje se desna polovina niza */
43     if (a[srednji] < x)
44         return bin_pretraga_rek(a, srednji + 1, d, x);
45     /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
46        pretrazuje se leva polovina niza */
47     else
48         return bin_pretraga_rek(a, l, srednji - 1, x);
49 }
50
51 int interp_pretraga_rek(int a[], int l, int d, int x)
52 {
53     int p;
54     if (x < a[l] || x > a[d])
55         return -1;
56     if (a[d] == a[l])
57         return l;
58     /* Pozicija na kojoj se trazi vrednost x */
59     p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
60     if (a[p] == x)
61         return p;
62     if (a[p] < x)
63         return interp_pretraga_rek(a, p + 1, d, x);
64     else
65         return interp_pretraga_rek(a, l, p - 1, x);
66 }
67
68 int main()
69 {
70     int a[MAX];

```

```

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

```

### Rešenje 3.3

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
7
8 /* 0 svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
10 typedef struct {
11     /* Indeks mora biti tipa long jer su podaci u datoteci preveliki za
12        int, npr. 20140123 */
13     long indeks;
14     char ime[MAX_DUZINA];
15     char prezime[MAX_DUZINA];
16 } Student;
17
18 /* Ucitani niz studenata ce biti sortiran rastuce prema indeksu, jer
19    su studenti u datoteci vec sortirani. Iz tog razloga pretraga po
20    indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
21    jer nema garancije da postoji uredjenje po prezimenu. */
22
23 /* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
24    sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
25    ako element nije pronadjen. */
26 int binarna_pretraga(Student a[], int n, long x)
27 {
28     int levi = 0;

```

```

30 int desni = n - 1;
31 int srednji;
32 /* Dokle god je indeks levi levo od indeksa desni */
33 while (levi <= desni) {
34     /* Racuna se srednja pozicija */
35     srednji = (levi + desni) / 2;
36     /* Ako je indeks studenta na toj poziciji veci od traznog, tada
37     se trazeni indeks mora nalaziti u levoj polovini niza */
38     if (x < a[srednji].indeks)
39         desni = srednji - 1;
40     /* Ako je pak manji od traznog, tada se on mora nalaziti u
41     desnoj polovini niza */
42     else if (x > a[srednji].indeks)
43         levi = srednji + 1;
44     else
45         /* Ako je jednak traznom indeksu x, tada je pronadjen student
46         sa traznom indeksom na poziciji srednji */
47         return srednji;
48 }
49 /* Ako nije pronadjen, vraca se -1 */
50 return -1;
51 }

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

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

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

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

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

```

```

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

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

118  /* Pretraga po indeksu */
    if (bin_pretraga) {
120      /* Unos indeksa koji se binarno traži u nizu */
        printf("Unesite indeks studenta čije informacije želite: ");
122      scanf("%ld", &trazen_indeks);
        i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
124      /* Rezultat binarne pretrage */
        if (i == -1)
126          printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
        else
128          printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
130    }
    /* Pretraga po prezimenu */
132    else {
        /* Unos prezimena koje se linearno traži u nizu */
134        printf("Unesite prezime studenta čije informacije želite: ");
        scanf("%s", trazeno_prezime);
136        i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
        /* Rezultat linearne pretrage */
138        if (i == -1)
            printf("Ne postoji student sa prezimenom %s\n",
140                  trazeno_prezime);
        else
142            printf("Indeks: %ld, Ime i prezime: %s %s\n",
                dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
144    }
    exit(EXIT_SUCCESS);
146 }

```

### Rešenje 3.4

```

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

4
#define MAX_STUDENATA 128
6 #define MAX_DUZINA 16

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

14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
16 {
    /* Ako je pozicija elementa na levom kraju veća od pozicije
18    elementa na desnom kraju dela niza koji se pretražuje, onda se
        zapravo pretražuje prazan deo niza. U praznom delu niza nema
20    traženog elementa pa se vraća -1 */
    if (levi > desni)
22        return -1;
    /* Racunanje pozicije srednjeg elementa */
24    int srednji = (levi + desni) / 2;

```



```

26  /* Da li je srednji bas onaj trazeni */
    if (a[srednji].indeks == x) {
28      return srednji;
    }
    /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
30     poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
    je poznato da je niz sortiran po indeksu u rastucem poretku. */
32     if (x < a[srednji].indeks)
        return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34     /* Inace ga treba traziti u desnoj polovini */
    else
36         return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
    }

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

52 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
54 {
    /* Ako je niz prazan, vraca se -1 */
    if (n == 0)
56         return -1;
    /* Ako se trazi poslednji student sa trazanim prezimenom, pocinje
58     se sa poslednjim studentom u nizu. */
    if (strcmp(a[n - 1].prezime, x) == 0)
60         return n - 1;
    return linearna_pretraga_rekurzivna(a, n - 1, x);
62 }

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

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

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

94     /* Otvaranje datoteke */
96     fin = fopen(argv[1], "r");
    if (fin == NULL) {

```

```

98     fprintf(stderr,
        "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
100    exit(EXIT_FAILURE);
    }

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

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

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

126        /* Rezultat binarne pretrage */
        if (i == -1)
128            printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
        else
130            printf("Indeks: %ld, Ime i prezime: %s %s\n",
                dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132    }
    /* Pretraga po prezimenu */
134    else {
        /* Unos prezimena koje se linearno traži u nizu */
136        printf("Unesite prezime studenta čije informacije želite: ");
        scanf("%s", trazeno_prezime);
138        i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
            trazeno_prezime);

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

```

### Rešenje 3.5

```

1  #include <stdio.h>
   #include <string.h>
3  #include <math.h>
   #include <stdlib.h>
5
   /* Struktura koja opisuje tacku u ravni */
7  typedef struct Tacka {
        float x;
9      float y;
   } Tacka;
11
   /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13    pocetka (0,0) */
   float rastojanje(Tacka A)
15 {
        return sqrt(A.x * A.x + A.y * A.y);
17 }

```

```

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

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

49
51 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
   t dimenzije n */
53 Tacka najbliza_y_osi(Tacka t[], int n)
{
55     Tacka najbliza;
   int i;
   najbliza = t[0];
57     for (i = 1; i < n; i++) {
59         if (fabs(t[i].y) < fabs(najbliza.y)) {
           najbliza = t[i];
61         }
   }
   return najbliza;
63 }

65 #define MAX 1024

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

73
   /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
   ime datoteke sa tackama */
75     if (argc < 2) {
77         fprintf(stderr,
           "koriscenje programa: %s ime_datoteke\n", argv[0]);
79         exit(EXIT_FAILURE);
   }

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

89
   /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa

```

```

91     tackama; i predstavlja indeks tekuće tacke */
i = 0;
93 while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
    i++;
95 }
n = i;

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

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

125 exit(EXIT_SUCCESS);
}

```

### Rešenje 3.6

```

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

4 /* Tacnost */
#define EPS 0.001

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

10
    /* Kako je u pitanju interval [0, 2] leva granica je 0, a desna 2 */
12     l = 0;
    d = 2;

14
    /* Sve dok se ne pronadje trazena vrednost argumenta */
16     while (1) {
        /* Polovi se interval */
18         s = (l + d) / 2;
        /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
20         tacnosti, prekida se pretraga */
        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;
        else
28             /* Inace, na intervalu [s, d] */
            l = s;
30     }

32
    /* Stampanje vrednost trazene tacke */

```

```

34     printf("%g\n", s);
36     return 0;
}

```

### Rešenje 3.7

```

#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, pretrazuje 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 3.8

```

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

#define MAX 256

int prvi_manji_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 manji od nule, a eventualni njegov
        prethodnik veci ili jednak nuli, pretraga se završava */
        if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))

```

```

    return s;
19  /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
    niza */
21  if (niz[s] >= 0)
    l = s + 1;
23  /* A inace leva */
    else
25      d = s - 1;
    }
27  return -1;
}
29
int main()
31 {
    int niz[MAX];
33     int n = 0;

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

    /* Stampanje rezultata */
39     printf("%d\n", prvi_manji_od_nule(niz, n));
41
    return 0;
43 }

```

## Rešenje 3.9

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   unsigned int logaritam_a(unsigned int x)
   {
7      /* Izlaz iz rekurzije */
       if (x == 1)
           return 0;
9      /* Rekurzivni korak */
       return 1 + logaritam_a(x >> 1);
11 }

13 unsigned int logaritam_b(unsigned int x)
   {
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
       najvece vaznosti, tj. najlevlja. Pretragu se vrsi od pozicije 0
       do 31 */
17     int d = 0, l = sizeof(unsigned int) * 8 - 1;
       int s;
19     /* Sve dok je desna granica pretrage desnije od leve */
       while (d <= l) {
21         /* Racuna se sredisnja pozicija */
           s = (l + d) / 2;
23         /* Proverava se da li je na toj poziciji trazena jedinica */
           if ((1 << s) <= x && (1 << (s + 1)) > x)
25             return s;
           /* Pretraga desne polovine binarnog zapisa */
           if ((1 << s) > x)
27             l = s - 1;
           /* Pretraga leve polovine binarnog zapisa */
           else
31             d = s + 1;
       }
33     return s;
35 }

37 int main()
   {
39     unsigned int x;

41     /* Unos podatka */
       scanf("%u", &x);
43

```

```

45  /* Provera da li je uneti broj pozitivan */
    if (x == 0) {
        fprintf(stderr, "Logaritam od nule nije definisan\n");
        exit(EXIT_FAILURE);
    }
49
    /* Ispis povratnih vrednosti funkcija */
51    printf("%u %u\n", logaritam_a(x), logaritam_b(x));
53
    exit(EXIT_SUCCESS);
}

```

### Rešenje 3.11

Datoteka 3.1: *sort.h*

```

#ifndef _SORT_H_
#define _SORT_H_ 1

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

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

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

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

```

```

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

67 /* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
68    i r. Njena ideja sortiranja je izbor jednog elementa niza, koji se
69    naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
70    svi elementi levo od njega bice manji, a svi desno bice veci od
71    njega. Kako je pivot doveden na svoje mesto, da bi niz bio
72    kompletno sortiran, potrebno je sortirati elemente levo (manje) od
73    njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
74    manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
75    deo moze se uraditi rekursivno. */
76 void quick_sort(int a[], int l, int r);
77
78 #endif

```

Datoteka 3.2: *sort.c*

```

1  #include "sort.h"

3  #define MAX 1000000

5  void selection_sort(int a[], int n)
6  {
7      int i, j;
8      int min;
9      int pom;

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

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 }

33 void insertion_sort(int a[], int n)
34 {
35     int i, j;

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

41         /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
42            potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...,a[i]
43            bude sortiran. Indeks j je trenutna pozicija na kojoj se
44            element koji se umece nalazi. Petlja se završava ili kada
45            element dodje do levog kraja (j==0) ili kada se naidje na

```



```

    element a[j-1] koji je manji od a[j]. */
47 for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
    int temp = a[j];
49     a[j] = a[j - 1];
    a[j - 1] = temp;
51 }
}
53 }

55 void bubble_sort(int a[], int n)
{
    int i, j;
    int ind;

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

61         /* Poput "mehurica" potiskuje se najveći element među elementima
63         od a[0] do a[i-1] na poziciju i-1 upoređujući susedne
        elemente niza i potiskujući veći u desno */
65         for (j = 0, ind = 0; j < i - 1; j++)
            if (a[j] > a[j + 1]) {
67                 int temp = a[j];
                a[j] = a[j + 1];
69                 a[j + 1] = temp;

71                 /* Promenljiva ind registruje da je bilo premestanja. Samo u
73                 tom slučaju ima smisla ici na sledeću iteraciju, jer ako
                nije bilo premestanja, znači da su svi elementi već u
75                 dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
                niza. Algoritam može biti i bez ovoga, sortiranje bi bilo
77                 ispravno, ali manje efikasno, jer bi se često nepotrebno
                vrsila mnoga upoređivanja, kada je već jasno da je
                sortiranje završeno. */
79                 ind = 1;
            }
81 }

83 void shell_sort(int a[], int n)
{
    int h = n / 2, i, j;
    while (h > 0) {
85         /* Insertion sort sa korakom h */
87         for (i = h; i < n; i++) {
            int temp = a[i];
            j = i;
91             while (j >= h && a[j - h] > temp) {
                a[j] = a[j - h];
93                 j -= h;
            }
95             a[j] = temp;
        }
97         h = h / 2;
    }
99 }

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

107     /* Izlaz iz rekurzije */
    if (l >= d)
109         return;

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

113     /* Rekurzivni pozivi */
    merge_sort(a, l, s);
    merge_sort(a, s + 1, d);
117     /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu

```

```

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

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

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

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

145     /* Prepisuje se "ucesljani" niz u originalni niz */
    for (k = 0, i = l; i <= d; i++, k++)
147         a[i] = b[k];
149 }

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

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

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

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

179     /* Postavljanje pivota na svoje mesto */
    swap(a, l, pivot_position);
181

183     /* Rekurzivno sortiranje elementa manjih od pivota */
    quick_sort(a, l, pivot_position - 1);
    /* Rekurzivno sortiranje elementa vecih od pivota */
185     quick_sort(a, pivot_position + 1, r);
}

```

Datoteka 3.3: *main.c*

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

```

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

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

8
9
10 int main(int argc, char *argv[])
11 {
12     /******
13      tip_sortiranja == 0 => selectionsort, (podrazumevano)
14      tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
15      tip_sortiranja == 2 => bubblesort, -b opcija komandne linije
16      tip_sortiranja == 3 => shellsort, -s opcija komandne linije
17      tip_sortiranja == 4 => mergesort, -m opcija komandne linije
18      tip_sortiranja == 5 => quicksort, -q opcija komandne linije
19      *****/
20     int tip_sortiranja = 0;
21     /******
22      tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
23      tip_niza == 1 => rastuce sortirani nizovi, -r opcija
24      tip_niza == 2 => opadajuće sortirani nizovi, -o opcija
25      *****/
26     int tip_niza = 0;

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

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

38
39     /* Ocitanje opcija i argumenata prilikom poziva programa */
40     for (i = 1; i < argc; i++) {
41         /* Ako je u pitanju opcija... */
42         if (argv[i][0] == '-') {
43             switch (argv[i][1]) {
44                 case 'i':
45                     tip_sortiranja = 1;
46                     break;
47                 case 'b':
48                     tip_sortiranja = 2;
49                     break;
50                 case 's':
51                     tip_sortiranja = 3;
52                     break;
53                 case 'm':
54                     tip_sortiranja = 4;
55                     break;
56                 case 'q':
57                     tip_sortiranja = 5;
58                     break;
59                 case 'r':
60                     tip_niza = 1;
61                     break;
62                 case 'o':
63                     tip_niza = 2;
64                     break;
65                 default:
66                     printf("Pogresna opcija -%c\n", argv[i][1]);
67                     return 1;
68                     break;
69             }
70         }
71     }
72     /* Ako je u pitanju argument, onda je to duzina niza koji treba
73      da se sortira */
74     else {
75         dimenzija = atoi(argv[i]);

```

```

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

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

109
110 /* Sortiranje niza na odgovarajuci nacin */
111 if (tip_sortiranja == 0)
112     selection_sort(niz, dimenzija);
113 else if (tip_sortiranja == 1)
114     insertion_sort(niz, dimenzija);
115 else if (tip_sortiranja == 2)
116     bubble_sort(niz, dimenzija);
117 else if (tip_sortiranja == 3)
118     shell_sort(niz, dimenzija);
119 else if (tip_sortiranja == 4)
120     merge_sort(niz, 0, dimenzija - 1);
121 else
122     quick_sort(niz, 0, dimenzija - 1);
123
124 /* Ispis elemenata niza */
125 /*****
126    Ovaj deo je iskomentaran jer vreme potrebno za njegovo
127    izvršavanje ne bi trebalo da bude ukljuceno u vreme izmereno
128    programom time. Takodje, kako je svrha ovog programa da prikaze
129    vremena razlicitih algoritama sortiranja, dimenzije nizova ce
130    biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
131    od toliko elemenata. Ovaj deo je koriscen u razvoju programa
132    zarad testiranja korektnosti.
133
134    printf("Sortiran niz je:\n");
135    for (i = 0; i < dimenzija; i++)
136        printf("%d\n", niz[i]);
137    *****/
138
139 exit(EXIT_SUCCESS);
140 }

```

## Rešenje 3.12

```

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

```

```

4 #define MAX_DIM 128

6 /* Funkcija za sortiranje niza karaktera */
void selectionSort(char s[])
8 {
    int i, j, min;
    char pom;
10    for (i = 0; s[i] != '\0'; i++) {
12        min = i;
        for (j = i + 1; s[j] != '\0'; j++)
14            if (s[j] < s[min])
                min = j;
16        if (min != i) {
            pom = s[i];
            s[i] = s[min];
            s[min] = pom;
20        }
    }
22 }

24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
26 {
    int i;

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

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

36

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

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

48

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

54

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

60

62    return 0;
}

```

### Rešenje 3.13

```

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

5 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
    sortiranom nizu celih brojeva */
7 int najmanje_rastojanje(int a[], int n)
{
9     int i, min;

```

```

    min = a[1] - a[0];
11  for (i = 2; i < n; i++)
    if (a[i] - a[i - 1] < min)
13      min = a[i] - a[i - 1];
    return min;
15 }

17
18
19 int main()
20 {
    int i, a[MAX];

21
    /* Ucitavaju se elementi niza sve do kraja ulaza */
22
23  i = 0;
    while (scanf("%d", &a[i]) != EOF)
24      i++;

25
26  /* Za sortiranje niza moze se koristiti bilo koja od funkcija
    sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
27  se selection sort. */
    selection_sort(a, i);

28
29  /* Ispis rezultata */
30  printf("%d\n", najmanje_rastojanje(a, i));

31
32  return 0;
33 }

```

## Rešenje 3.14

```

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

5  /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
   najvise puta pojavio u tom nizu */
7  int najvise_puta(int a[], int n)
8  {
9      int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
    /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
11     for (i = 0; i < n; i = j) {
        br_pojava = 1;
12         for (j = i + 1; j < n && a[i] == a[j]; j++)
            br_pojava++;
13         /* Ispitivanje da li se do tog trenutka i-ti element pojavio
            najvise puta u nizu */
14         if (br_pojava > max_br_pojava) {
            max_br_pojava = br_pojava;
15             i_max_pojava = i;
        }
16     }
    /* Vraca se element koji se najvise puta pojavio u nizu */
21     return a[i_max_pojava];
22 }

23
24
25 int main()
26 {
    int a[MAX_DIM], i;

27
28     /* Ucitavanje elemenata niza sve do kraja ulaza */
29
30     i = 0;
    while (scanf("%d", &a[i]) != EOF)
31         i++;

32
33     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
    sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
34     se merge sort. */
    merge_sort(a, 0, i - 1);

35
36     /* Odredjuje se broj koji se najvise puta pojavio u nizu */
37     printf("%d\n", najvise_puta(a, i));
38 }

```

```

43     return 0;
}

```

### Rešenje 3.15

```

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

5  /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
   u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
   poretku */
7  int binarna_pretraga(int a[], int n, int x)
9  {
   int levi = 0, desni = n - 1, srednji;
11
   while (levi <= desni) {
13     srednji = (levi + desni) / 2;
     if (a[srednji] == x)
15         return 1;
     else if (a[srednji] > x)
17         desni = srednji - 1;
     else if (a[srednji] < x)
19         levi = srednji + 1;
   }
21     return 0;
}

23
25 int main()
{
   int a[MAX_DIM], n = 0, zbir, i;
27
   /* Ucitava se trazeni zbir */
29   printf("Unesite trazeni zbir: ");
   scanf("%d", &zbir);
31
   /* Ucitavaju se elementi niza sve do kraja ulaza */
33   i = 0;
   printf("Unesite elemente niza: ");
35   while (scanf("%d", &a[i]) != EOF)
       i++;
37   n = i;

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

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

```

### Rešenje 3.16

```

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

4  int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
           int dim3)
6  {
   int i = 0, j = 0, k = 0;

```

```

8  /* U slucaju da je dimenzija treceg niza manja od neophodne,
   funkcija vraca -1 */
10 if (dim3 < dim1 + dim2)
    return -1;

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

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

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

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

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

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

60     return 0;
62 }

```

## Rešenje 3.17

```

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

4
int main(int argc, char *argv[])
6  {
    FILE *fin1 = NULL, *fin2 = NULL;
8    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) {

```



```

16     fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
    exit(EXIT_FAILURE);
18 }
19
20 /* Otvaranje datoteke zadate prvim argumentom komandne linije */
21 fin1 = fopen(argv[1], "r");
22 if (fin1 == NULL) {
23     fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
24     exit(EXIT_FAILURE);
25 }
26
27 /* Otvaranje datoteke zadate drugim argumentom komandne linije */
28 fin2 = fopen(argv[2], "r");
29 if (fin2 == NULL) {
30     fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
31     exit(EXIT_FAILURE);
32 }
33
34 /* Otvaranje datoteke za upis rezultata */
35 fout = fopen("ceo-tok.txt", "w");
36 if (fout == NULL) {
37     fprintf(stderr,
38         "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
39     exit(EXIT_FAILURE);
40 }
41
42 /* Citanje narednog studenta iz prve datoteke */
43 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
44     kraj1 = 1;
45
46 /* Citanje narednog studenta iz druge datoteke */
47 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
48     kraj2 = 1;
49
50 /* Sve dok nije dostignut kraj neke datoteke */
51 while (!kraj1 && !kraj2) {
52     if (strcmp(ime1, ime2) < 0) {
53         /* Ime i prezime iz prve datoteke je leksikografski ranije, i
54            biva upisano u izlaznu datoteku */
55         fprintf(fout, "%s %s\n", ime1, prezime1);
56         /* Citanje narednog studenta iz prve datoteke */
57         if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
58             kraj1 = 1;
59     } else {
60         /* Ime i prezime iz druge datoteke je leksikografski ranije, i
61            biva upisano u izlaznu datoteku */
62         fprintf(fout, "%s %s\n", ime2, prezime2);
63         /* Citanje narednog studenta iz druge datoteke */
64         if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
65             kraj2 = 1;
66     }
67 }
68
69 /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
70    druge datoteke, onda ima jos studenata u prvoj datoteci, koje
71    treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
72    */
73 while (!kraj1) {
74     fprintf(fout, "%s %s\n", ime1, prezime1);
75     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
76         kraj1 = 1;
77 }
78
79 /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
80    datoteke, onda ima jos studenata u drugoj datoteci, koje treba
81    prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
82 while (!kraj2) {
83     fprintf(fout, "%s %s\n", ime2, prezime2);
84     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
85         kraj2 = 1;
86 }
87
88 /* Zatvaranje datoteka */

```

```
88     fclose(fin1);
89     fclose(fin2);
90     fclose(fout);
91
92     exit(EXIT_SUCCESS);
93 }
```

### Rešenje 3.18

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  #define MAX_BR_TACAKA 128
7
8  /* Struktura koja reprezentuje koordinate tacke */
9  typedef struct Tacka {
10     int x;
11     int y;
12 } Tacka;
13
14 /* Funkcija racuna rastojanje zadate tacke od koordinatnog pocetka
15    (0,0) */
16 float rastojanje(Tacka A)
17 {
18     return sqrt(A.x * A.x + A.y * A.y);
19 }
20
21 /* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
22    pocetka */
23 void sortiraj_po_rastojanju(Tacka t[], int n)
24 {
25     int min, i, j;
26     Tacka tmp;
27
28     for (i = 0; i < n - 1; i++) {
29         min = i;
30         for (j = i + 1; j < n; j++) {
31             if (rastojanje(t[j]) < rastojanje(t[min])) {
32                 min = j;
33             }
34         }
35         if (min != i) {
36             tmp = t[i];
37             t[i] = t[min];
38             t[min] = tmp;
39         }
40     }
41 }
42
43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
44 void sortiraj_po_x(Tacka t[], int n)
45 {
46     int min, i, j;
47     Tacka tmp;
48
49     for (i = 0; i < n - 1; i++) {
50         min = i;
51         for (j = i + 1; j < n; j++) {
52             if (abs(t[j].x) < abs(t[min].x)) {
53                 min = j;
54             }
55         }
56         if (min != i) {
57             tmp = t[i];
58             t[i] = t[min];
59             t[min] = tmp;
60         }
61     }
62 }
63 }
```

```

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

69     for (i = 0; i < n - 1; i++) {
71         min = i;
        for (j = i + 1; j < n; j++) {
73             if (abs(t[j].y) < abs(t[min].y)) {
                min = j;
75             }
        }
77         if (min != i) {
            tmp = t[i];
79             t[i] = t[min];
            t[min] = tmp;
81         }
    }
83 }

85 int main(int argc, char *argv[])
{
87     FILE *ulaz;
    FILE *izlaz;
89     Tacka tacke[MAX_BR_TACAKA];
    int i, n;

91     /* Proveravanje broja argumenata komandne linije: ocekuje se ime
93     izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
    datoteke, tj. 4 argumenta */
95     if (argc != 4) {
        fprintf(stderr,
97             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
        return 0;
99     }

101     /* Otvaranje datoteke u kojoj su zadate tacke */
    ulaz = fopen(argv[2], "r");
103     if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
105             argv[2]);
        return 0;
107     }

109     /* Otvaranje datoteke u koju treba upisati rezultat */
    izlaz = fopen(argv[3], "w");
111     if (izlaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
113             argv[3]);
        return 0;
115     }

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

125     /* Ukupan broj procitanih tacaka */
    n = i;
127

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

```

```

137     case 'y':
        /* Sortiranje po vrednosti y koordinate */
139         sortiraj_po_y(tacke, n);
        break;
141     case 'o':
        /* Sortiranje po udaljenosti od koordinatnog pocetka */
143         sortiraj_po_rastojanju(tacke, n);
        break;
145     }

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

151     /* Zatvaranje otvorenih datoteka */
153     fclose(ulaz);
    fclose(izlaz);

155     return 0;
157 }

```

### Rešenje 3.19

```

#include <stdio.h>
2 #include <string.h>
#include <stdlib.h>
4
#define MAX 1000
6 #define MAX_DUZINA 16

8 /* Struktura koja reprezentuje jednog gradjanina */
typedef struct gr {
10     char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
12 } Gradjanin;

14 /* Funkcija sortira niz gradjana rastuce po imenima */
void sort_ime(Gradjanin a[], int n)
16 {
    int i, j;
18     int min;
    Gradjanin pom;

20     for (i = 0; i < n - 1; i++) {
22         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
            najmanji od elemenata a[i].ime,...,a[n-1].ime. */
24         min = i;
        for (j = i + 1; j < n; j++)
26             if (strcmp(a[j].ime, a[min].ime) < 0)
                min = j;
28         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
            su (i) i min razliciti, inace je nepotrebno. */
30         if (min != i) {
            pom = a[i];
32             a[i] = a[min];
            a[min] = pom;
34         }
    }
36 }

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

44     for (i = 0; i < n - 1; i++) {
46         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
            najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
48         min = i;

```

```

50     for (j = i + 1; j < n; j++)
51         if (strcmp(a[j].prezime, a[min].prezime) < 0)
52             min = j;
53     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
54        su (i) i min razliciti, inace je nepotrebno. */
55     if (min != i) {
56         pom = a[i];
57         a[i] = a[min];
58         a[min] = pom;
59     }
60 }

62 /* Pretraga niza Gradjana */
63 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64 {
65     int i;
66     for (i = 0; i < n; i++)
67         if (strcmp(a[i].ime, x->ime) == 0
68             && strcmp(a[i].prezime, x->prezime) == 0)
69             return i;
70     return -1;
71 }

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

```

```

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

```

## Rešenje 3.21

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  #define MAX_BR_RECII 128
6  #define MAX_DUZINA_RECII 32
7
8  /* Funkcija koja izracunava broj suglasnika u reci */
9  int broj_suglasnika(char s[])
10 {
11     char c;
12     int i;
13     int suglasnici = 0;
14     /* Prolazi karakter po karakter kroz zadatu nisku */
15     for (i = 0; s[i]; i++) {
16         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
17            pokriven slucaj i malih i velikih suglasnika. */
18         if (isalpha(s[i])) {
19             c = toupper(s[i]);
20             /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika. */
21             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
22                 suglasnici++;
23         }
24     }
25     /* Vraca se izracunata vrednost */
26     return suglasnici;
27 }
28
29 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
30    duzini reci se mora proslediti zbog pravilnog upravljanja
31    memorijom */
32 void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)
33 {
34     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35         duzina_j, duzina_min;
36     char tmp[MAX_DUZINA_RECII];
37     for (i = 0; i < n - 1; i++) {
38         min = i;
39         for (j = i; j < n; j++) {
40             /* Prvo se upoređuje broj suglasnika */
41             broj_suglasnika_j = broj_suglasnika(reci[j]);
42             broj_suglasnika_min = broj_suglasnika(reci[min]);
43             if (broj_suglasnika_j < broj_suglasnika_min)
44                 min = j;
45             else if (broj_suglasnika_j == broj_suglasnika_min) {
46                 /* Zatim, recima koje imaju isti broj suglasnika upoređuju
47                    se duzine */
48                 duzina_j = strlen(reci[j]);
49                 duzina_min = strlen(reci[min]);
50
51                 if (duzina_j < duzina_min)
52                     min = j;
53             }
54         }
55         /* Zamena mesta */
56         strcpy(tmp, reci[i]);
57         strcpy(reci[i], reci[min]);
58         strcpy(reci[min], tmp);
59     }
60 }

```

```

55         /* Ako reci imaju i isti broj suglasnika i iste duzine,
           uporedjuju se leksikografski */
56         if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
57             min = j;
58     }
59 }
60 if (min != i) {
61     strcpy(tmp, reci[min]);
62     strcpy(reci[min], reci[i]);
63     strcpy(reci[i], tmp);
64 }
65 }
66 }
67
68 int main()
69 {
70     FILE *ulaz;
71     int i = 0, n;
72
73     /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
       a drugi maksimalnu duzinu pojedinačne reci */
74     char reci[MAX_BR_RECII][MAX_DUZINA_RECII];
75
76     /* Otvaranje datoteke niske.txt za citanje */
77     ulaz = fopen("niske.txt", "r");
78     if (ulaz == NULL) {
79         fprintf(stderr,
80             "Greska prilikom otvaranja datoteke niske.txt!\n");
81         return 0;
82     }
83
84     /* Sve dok se moze procitati sledeca rec */
85     while (fscanf(ulaz, "%s", reci[i]) != EOF) {
86         /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
           prekida se ucitavanje */
87         if (i == MAX_BR_RECII)
88             break;
89         /* Priprema brojac za narednu iteraciju */
90         i++;
91     }
92
93     /* n je duzina niza reci i predstavlja poslednju vrednost
       koriscenog brojac */
94     n = i;
95     /* Poziv funkcije za sortiranje reci */
96     sortiraj_reci(reci, n);
97
98     /* Ispis sortiranog niza reci */
99     for (i = 0; i < n; i++) {
100         printf("%s ", reci[i]);
101     }
102     printf("\n");
103
104     /* Zatvaranje datoteke */
105     fclose(ulaz);
106
107     return 0;
108 }

```

### Rešenje 3.22

```

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

#define MAX_ARTIKALA 100000

/* Struktura koja predstavlja jedan artikal */
typedef struct art {
    long kod;
    char naziv[20];
    char proizvođjac[20];

```

```

12     float cena;
13 } Artiklal;
14
15 /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
16    traženim bar kodom */
17 int binarna_pretraga(Artikal a[], int n, long x)
18 {
19     int levi = 0;
20     int desni = n - 1;
21
22     /* Dokle god je indeks levi levo od indeksa desni */
23     while (levi <= desni) {
24         /* Racuna se sredisnji indeks */
25         int srednji = (levi + desni) / 2;
26         /* Ako je sredisnji element veci od traženog, tada se traženi
27            mora nalaziti u levoj polovini niza */
28         if (x < a[srednji].kod)
29             desni = srednji - 1;
30         /* Ako je sredisnji element manji od traženog, tada se traženi
31            mora nalaziti u desnoj polovini niza */
32         else if (x > a[srednji].kod)
33             levi = srednji + 1;
34         else
35             /* Ako je sredisnji element jednak traženom, tada je artikal sa
36                bar kodom x pronađen na poziciji srednji */
37             return srednji;
38     }
39     /* Ako nije pronađen artikal za traženim bar kodom, vraća se -1 */
40     return -1;
41 }
42
43 /* Funkcija koja sortira niz artikala po bar kodovima rastuće */
44 void selection_sort(Artikal a[], int n)
45 {
46     int i, j;
47     int min;
48     Artiklal pom;
49
50     for (i = 0; i < n - 1; i++) {
51         min = i;
52         for (j = i + 1; j < n; j++)
53             if (a[j].kod < a[min].kod)
54                 min = j;
55         if (min != i) {
56             pom = a[i];
57             a[i] = a[min];
58             a[min] = pom;
59         }
60     }
61 }
62
63 int main()
64 {
65     Artiklal asortiman[MAX_ARTIKALA];
66     long kod;
67     int i, n;
68     float racun;
69
70     FILE *fp = NULL;
71
72     /* Otvaranje datoteke */
73     if ((fp = fopen("artikli.txt", "r")) == NULL) {
74         fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
75         exit(EXIT_FAILURE);
76     }
77
78     /* Ucitavanje artikala */
79     i = 0;
80     while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
81                  asortiman[i].naziv, asortiman[i].proizvodjac,
82                  &asortiman[i].cena) == 4)
83         i++;
84 }

```



```

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

88  n = i;

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

106  /* Ispis stanja u prodavnici */
printf
108  ("Asortiman:\nKOD          Naziv artikla      Ime proizvođjaca      Cena\
n");
for (i = 0; i < n; i++)
110  printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
asortiman[i].naziv, asortiman[i].proizvođjac,
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) {
128  printf("\tGRESKA: Ne postoji proizvod sa trazanim kodom!\n");
} else {
130  printf("\tTrazili ste:\t%s %s %12.2f\n",
asortiman[i].naziv, asortiman[i].proizvođjac,
132  asortiman[i].cena);
/* I dodavanje na ukupan racun */
134  racun += asortiman[i].cena;
}
136  /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
nema vise artikla */
138  printf("Unesite kod artikla [ili 0 za prekid]: \t");
scanf("%ld", &kod);
140  if (kod == 0)
break;
142  }
/* Stampanje ukupnog racuna trenutnog kupca */
144  printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
}

146  printf("Kraj rada kase!\n");

148  exit(EXIT_SUCCESS);
150 }

```

## Rešenje 3.23

```

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

```

```

3 #include <string.h>

5 #define MAX 500

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

15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
    rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
18 {
19     int i, j;
20     int min;
21     Student pom;

23     for (i = 0; i < n - 1; i++) {
24         min = i;
25         for (j = i + 1; j < n; j++)
26             if (strcmp(niz[j].ime, niz[min].ime) < 0)
27                 min = j;

29         if (min != i) {
30             pom = niz[min];
31             niz[min] = niz[i];
32             niz[i] = pom;
33         }
34     }
35 }

37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
    zadataka opadajuće, a ukoliko neki studenti imaju isti broj
39 uradjenih zadataka sortiraju se po dužini imena rastuce. */
void sort_zadatke_pa_imena(Student niz[], int n)
41 {
42     int i, j;
43     int max;
44     Student pom;

45     for (i = 0; i < n - 1; i++) {
46         max = i;
47         for (j = i + 1; j < n; j++)
48             if (niz[j].zadaci > niz[max].zadaci)
49                 max = j;
50             else if (niz[j].zadaci == niz[max].zadaci
51                     && strlen(niz[j].ime) < strlen(niz[max].ime))
52                 max = j;
53         if (max != i) {
54             pom = niz[max];
55             niz[max] = niz[i];
56             niz[i] = pom;
57         }
58     }
59 }

61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
    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 {
68     int i, j;
69     int max;
70     Student pom;

71     for (i = 0; i < n - 1; i++) {
72         max = i;
73         for (j = i + 1; j < n; j++)
74             if (niz[j].prisustvo > niz[max].prisustvo)
75                 max = j;

```

```

77     else if (niz[j].prisustvo == niz[max].prisustvo
78         && niz[j].zadaci > niz[max].zadaci)
79         max = j;
80     else if (niz[j].prisustvo == niz[max].prisustvo
81         && niz[j].zadaci == niz[max].zadaci
82         && strcmp(niz[j].prezime, niz[max].prezime) > 0)
83         max = j;
84     if (max != i) {
85         pom = niz[max];
86         niz[max] = niz[i];
87         niz[i] = pom;
88     }
89 }

91 int main(int argc, char *argv[])
92 {
93     Student praktikum[MAX];
94     int i, br_studenata = 0;
95
96     FILE *fp = NULL;
97
98     /* Otvaranje datoteke za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
100         fprintf(stderr, "Neuspesno otvaranje datoteke aktivnost.txt.\n");
101         exit(EXIT_FAILURE);
102     }
103
104     /* Ucitavanje sadrzaja */
105     for (i = 0;
106         fscanf(fp, "%s%s%d", praktikum[i].ime,
107             praktikum[i].prezime, &praktikum[i].prisustvo,
108             &praktikum[i].zadaci) != EOF; i++);
109
110     /* Zatvaranje datoteke */
111     fclose(fp);
112     br_studenata = i;
113
114     /* Kreiranje prvog spiska studenata po prvom kriterijumu */
115     sort_ime_leksikografski(praktikum, br_studenata);
116     /* Otvaranje datoteke za pisanje */
117     if ((fp = fopen("dat1.txt", "w")) == NULL) {
118         fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
119         exit(EXIT_FAILURE);
120     }
121     /* Upis niza u datoteku */
122     fprintf
123     (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
124     for (i = 0; i < br_studenata; i++)
125         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
126             praktikum[i].prezime, praktikum[i].prisustvo,
127             praktikum[i].zadaci);
128     /* Zatvaranje datoteke */
129     fclose(fp);
130
131     /* Kreiranje drugog spiska studenata po drugom kriterijumu */
132     sort_zadatke_pa_imena(praktikum, br_studenata);
133     /* Otvaranje datoteke za pisanje */
134     if ((fp = fopen("dat2.txt", "w")) == NULL) {
135         fprintf(stderr, "Neuspesno otvaranje datoteke dat2.txt.\n");
136         exit(EXIT_FAILURE);
137     }
138     /* Upis niza u datoteku */
139     fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
140     fprintf(fp, "pa po duzini imena rastuce:\n");
141     for (i = 0; i < br_studenata; i++)
142         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
143             praktikum[i].prezime, praktikum[i].prisustvo,
144             praktikum[i].zadaci);
145     /* Zatvaranje datoteke */
146     fclose(fp);
147
148     /* Kreiranje treceg spiska studenata po trecem kriterijumu */
149     sort_prisustvo_pa_zadatke_pa_prezimana(praktikum, br_studenata);

```

```

149  /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat3.txt", "w")) == NULL) {
151      fprintf(stderr, "Neuspesno 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 zadatka,\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 3.24

```

#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)
{
18   Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

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

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

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

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

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

44 int main(int argc, char *argv[])
{
46   FILE *ulaz;
    Pesma *pesme;                                /* Pokazivac na deo memorije za
48   cuvanje pesama */

    int alocirano_za_pesme;                       /* Broj mesta alociranih za pesme */
50   int i;                                         /* Redni broj pesme cije se
    informacije citaju */

```

```

52  int n;                                /* Ukupan broj pesama */
53  int j, k;
54  char c;
55  int alocirano;                        /* Broj mesta alociranih za propratne
56                                     informacije o pesmama */
57
58  int broj_gledanja;
59
60  /* Priprema datoteke za citanje */
61  ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
62  if (ulaz == NULL) {
63      printf("Greska pri otvaranju ulazne datoteke!\n");
64      return 0;
65  }
66
67  /* Citanje informacija o pesmama */
68  pesme = NULL;
69  alocirano_za_pesme = 0;
70  i = 0;
71
72  while (1) {
73
74      /* Proverava da li je dostignut kraj datoteke */
75      c = fgetc(ulaz);
76      if (c == EOF) {
77          /* Nema vise sadrzaja za citanje */
78          break;
79      } else {
80          /* Inace, vracamo procitani karakter nazad */
81          ungetc(c, ulaz);
82      }
83
84      /* Provera da li postoji dovoljno memorije za citanje nove pesme */
85      if (alocirano_za_pesme == i) {
86
87          /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
88             novih KORAK mesta */
89          alocirano_za_pesme += KORAK;
90          pesme =
91              (Pesma *) realloc(pesme,
92                               alocirano_za_pesme * sizeof(Pesma));
93
94          /* Proverava da li je nova memorija uspesno realocirana */
95          if (pesme == NULL) {
96              /* Ako nije ispisuje se obavestenje */
97              printf("Problem sa alokacijom memorije!\n");
98              /* I oslobadja sva memorija zauzeta do ovog koraka */
99              for (k = 0; k < i; k++) {
100                  free(pesme[k].izvodjac);
101                  free(pesme[k].naslov);
102              }
103              free(pesme);
104              return 0;
105          }
106      }
107
108      /* Ako jeste, nastavlja se sa citanjem pesama ... */
109      /* Cita se ime izvodjaca */
110      j = 0;                                /* Pozicija na koju treba smestiti
111                                           procitani karakter */
112      alocirano = 0;                        /* Broj alociranih mesta */
113      pesme[i].izvodjac = NULL;            /* Memorija za smestanje procitanih
114                                           karaktera */
115
116      /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
117         izvodjaca) citaju se karakteri iz datoteke */
118      while ((c = fgetc(ulaz)) != ' ') {
119          /* Proverav da li postoji dovoljno memorije za smestanje
120             procitanog karaktera */
121          if (j == alocirano) {
122
123              /* Ako ne, ako je potrosena sva alocirana memorija, alocira
124                 se novih KORAK mesta */
125              alocirano += KORAK;

```

```

126     pesme[i].izvodjac =
        (char *) realloc(pesme[i].izvodjac,
            alocirano * sizeof(char));
128
129     /* Provera da li je nova alokacija uspesna */
130     if (pesme[i].izvodjac == NULL) {
131         /* Ako nije oslobadja se sva memorija zauzeta do ovog
132            koraka */
133         for (k = 0; k < i; k++) {
134             free(pesme[k].izvodjac);
135             free(pesme[k].naslov);
136         }
137         free(pesme);
138         /* I prekida sa izvršavanjem programa */
139         return 0;
140     }
141
142 }
143
144 /* Ako postoji dovoljno memorije, smestamo procitani karakter */
145 pesme[i].izvodjac[j] = c;
146 j++;
147 /* I nastavlja se sa citanjem */
148 }
149
150 /* Upis terminirajuće nule na kraj reci */
151 pesme[i].izvodjac[j] = '\0';
152
153 /* Preskace se karakter - */
154 fgetc(ulaz);
155
156 /* Preskace se razmak */
157 fgetc(ulaz);
158
159 /* Cita se naslov pesme */
160 j = 0;
161
162 alocirano = 0;
163 pesme[i].naslov = NULL;
164
165 /* Pozicija na koju treba smestiti
166    procitani karakter */
167 /* Broj alociranih mesta */
168 /* Memorija za smestanje procitanih
169    karaktera */
170
171 /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
172    karakteri iz datoteke */
173 while ((c = fgetc(ulaz)) != ',') {
174     /* Provera da li postoji dovoljno memorije za smestanje
175        procitanog karaktera */
176     if (j == alocirano) {
177         /* Ako ne, ako je potrošena sva alocirana memorija, alocira
178            se novih KORAK mesta */
179         alocirano += KORAK;
180         pesme[i].naslov =
181             (char *) realloc(pesme[i].naslov,
182                 alocirano * sizeof(char));
183
184         /* Provera da li je nova alokacija uspesna */
185         if (pesme[i].naslov == NULL) {
186             /* Ako nije, oslobadja se sva memorija zauzeta do ovog
187                koraka */
188             for (k = 0; k < i; k++) {
189                 free(pesme[k].izvodjac);
190                 free(pesme[k].naslov);
191             }
192             free(pesme[i].izvodjac);
193             free(pesme);
194
195             /* I prekida izvršavanje programa */
196             return 0;
197         }
198     }
199     pesme[i].naslov[j] = c;
200     j++;
201     /* I nastavlja dalje sa citanjem */
202 }

```

```

198     /* Upisuje se terminirajuca nula na kraj reci */
    pesme[i].naslov[j] = '\0';

200     /* Preskace se razmak */
    fgetc(ulaz);

202     /* Cita se broj gledanja */
    broj_gledanja = 0;

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

212     /* Prelazi se na citanje sledece pesme */
    i++;

216 }

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

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

240 }

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

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

254 return 0;

256 }

```

### Rešenje 3.27

```

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

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

```

```

10     for (i = 0; i < m; i++) {
12         zbir += a[v][i];
13     }
14     return zbir;
15 }
16
17 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
18    osnovu zbira koriscenjem selection sort algoritma */
19 void sortiraj_vrste(int **a, int n, int m)
20 {
21     int i, j, min;
22
23     for (i = 0; i < n - 1; i++) {
24         min = i;
25         for (j = i + 1; j < n; j++) {
26             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
27                 min = j;
28             }
29         }
30         if (min != i) {
31             int *tmp;
32             tmp = a[i];
33             a[i] = a[min];
34             a[min] = tmp;
35         }
36     }
37 }
38
39 int main(int argc, char *argv[])
40 {
41     int **a;
42     int n, m;
43
44     /* Unos dimenzija matrice */
45     printf("Unesite dimenzije matrice: ");
46     scanf("%d %d", &n, &m);
47
48     /* Alokacija memorije */
49     a = alociraj_matricu(n, m);
50
51     /* Ucitavanje elementa matrice */
52     printf("Unesite elemente matrice po vrstama:\n");
53     ucitaj_matricu(a, n, m);
54
55     /* Poziv funkcije koja sortira vrste matrice prema zbiru */
56     sortiraj_vrste(a, n, m);
57
58     /* Ispis rezultujuce matrice */
59     printf("Sortirana matrica je:\n");
60     ispisi_matricu(a, n, m);
61
62     /* Oslobadjanje memorije */
63     a = deallociraj_matricu(a, n);
64
65     return 0;
66 }

```

### Rešenje 3.30

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <search.h>
5
6  #define MAX 100
7
8  /* Funkcija poredjenja dva cela broja */
9  int compare_int(const void *a, const void *b)
10 {
11     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
12        se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */

```



```

13      /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
14         oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */
15
16
17     int b1 = *((int *) a);
18     int b2 = *((int *) b);
19
20     if (b1 > b2)
21         return 1;
22     else if (b1 < b2)
23         return -1;
24     else
25         return 0;
26 }
27
28 int compare_int_desc(const void *a, const void *b)
29 {
30     /* Za obrnuti poredak treba samo oduzimati a od b */
31     /* return *((int *)b) - *((int *)a); */
32
33     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
34        funkcija */
35     return -compare_int(a, b);
36 }
37
38 int main()
39 {
40     size_t n;
41     int i, x;
42     int a[MAX], *p = NULL;
43
44     /* Unos dimenzije */
45     printf("Uneti dimenziju niza: ");
46     scanf("%ld", &n);
47     if (n > MAX)
48         n = MAX;
49
50     /* Unos elementa niza */
51     printf("Uneti elemente niza:\n");
52     for (i = 0; i < n; i++)
53         scanf("%d", &a[i]);
54
55     /* Sortiranje niza celih brojeva */
56     qsort(a, n, sizeof(int), &compare_int);
57
58     /* Prikaz sortiranog niz */
59     printf("Sortirani niz u rastucem poretku:\n");
60     for (i = 0; i < n; i++)
61         printf("%d ", a[i]);
62     putchar('\n');
63
64     /* Pretrazivanje niza */
65     /* Vrednost koja ce biti trazena u nizu */
66     printf("Uneti element koji se trazi u nizu: ");
67     scanf("%d", &x);
68
69     /* Binarna pretraga */
70     printf("Binarna pretraga: \n");
71     p = bsearch(&x, a, n, sizeof(int), &compare_int);
72     if (p == NULL)
73         printf("Elementa nema u nizu!\n");
74     else
75         printf("Element je nadjen na poziciji %ld\n", p - a);
76
77     /* Linearna pretraga */
78     printf("Linearna pretraga (lfind): \n");
79     p = lfind(&x, a, &n, sizeof(int), &compare_int);
80     if (p == NULL)
81         printf("Elementa nema u nizu!\n");
82     else
83         printf("Element je nadjen na poziciji %ld\n", p - a);
84
85     return 0;

```

}

## Rešenje 3.31

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <search.h>
5
6  #define MAX 100
7
8  /* Funkcija racuna broj delilaca broja x */
9  int no_of_deviders(int x)
10 {
11     int i;
12     int br;
13
14     /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
15     if (x < 0)
16         x = -x;
17     if (x == 0)
18         return 0;
19     if (x == 1)
20         return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22     br = 2;
23     for (i = 2; i < sqrt(x); i++)
24         if (x % i == 0)
25             /* Ako i deli x onda su delioci: i, x/i */
26             br += 2;
27     /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
28        promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29        jos jednog delioca */
30     if (i * i == x)
31         br++;
32
33     return br;
34 }
35
36 /* Funkcija poredjenja dva cela broja po broju delilaca */
37 int compare_no_deviders(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = no_of_deviders(ak);
42     int n_d_b = no_of_deviders(bk);
43
44     if (n_d_a > n_d_b)
45         return 1;
46     else if (n_d_a < n_d_b)
47         return -1;
48     else
49         return 0;
50 }
51
52 int main()
53 {
54     size_t n;
55     int i;
56     int a[MAX];
57
58     /* Unos dimenzije */
59     printf("Uneti dimenziju niza: ");
60     scanf("%ld", &n);
61     if (n > MAX)
62         n = MAX;
63
64     /* Unos elementa niza */
65     printf("Uneti elemente niza:\n");
66     for (i = 0; i < n; i++)
67         scanf("%d", &a[i]);

```

```

69  /* Sortiranje niza celih brojeva prema broju delilaca */
    qsort(a, n, sizeof(int), &compare_no_dividers);

71

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

77
79  return 0;
}

```

### Rešenje 3.32

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <search.h>
5
6  #define MAX_NISKI 1000
7  #define MAX_DUZINA 30
8
9  /******
   Niz nizova karaktera ovog potpisa
   char niske[3][4];
   se moze graficki predstaviti ovako:
13  -----
   | a | b | c | \0 | | d | e | \0 |   | f | g | h | \0 | |
   -----
15
   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
17  svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
   nalazi na adresi koja je za 4 veca od prve reci, a za 4 manja od
19  adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
   i ona je tipa char*.
21
   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23  koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
   reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
   nad njima.
27  *****/
   int poredi_leksikografski(const void *a, const void *b)
29  {
       return strcmp((char *) a, (char *) b);
31  }

33  /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
35  int poredi_duzine(const void *a, const void *b)
   {
37       return strlen((char *) a) - strlen((char *) b);
   }

39
   int main()
41  {
       int i;
43       size_t n;
       FILE *fp = NULL;
45       char niske[MAX_NISKI][MAX_DUZINA];
       char *p = NULL;
47       char x[MAX_DUZINA];

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

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

57       /* Zatvaranje datoteke */

```

```

59  fclose(fp);
    n = i;

61

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

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

71

    /* Unos trazene niske */
73  printf("Uneti trazenu nisku: ");
    scanf("%s", x);

75

    /* Binarna pretraga */
77  /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
        je niz vec sortiran leksikografski. */
79  p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
        &poredi_leksikografski);

81

    if (p != NULL)
83        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
            p, (p - (char *) niske) / MAX_DUZINA);
    else
85        printf("Niska nije pronadjena u nizu\n");

87

    /* Sortiranje po duzini */
89  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

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

95

    /* Linearna pretraga */
97  p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
        &poredi_leksikografski);

99

    if (p != NULL)
101        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
            p, (p - (char *) niske) / MAX_DUZINA);
    else
103        printf("Niska nije pronadjena u nizu\n");

105  exit(EXIT_SUCCESS);
107 }

```

## Rešenje 3.33

```

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

5

    #define MAX_NISKI 1000
7  #define MAX_DUZINA 30

9  /*****
    Niz pokazivaca na karaktere ovog potpisa
11  char *niske[3];
    posle alokacije u main-u se moze graficki predstaviti ovako:
13  -----
    | X | -----> | a | b | c | \0|
15  -----
    | Y | -----> | d | e | \0|
17  -----
    | Z | -----> | f | g | h | \0|
19  -----

    Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti

```

```

21  njegov element pokazivac koji pokazuje na alocirane karaktere i-te
    reci. Njegov tip je char*.
23
    Kako pokazivaci a i b u sledecoj funkciji sadrže adrese elemenata
25  koji trebaju biti upoređeni (recimo adresu od X i adresu od Z), i
    kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
27  moraju i kastovati. Da bi se leksikografski uporedili elementi X i
    Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
29  u sledecoj funkciji poziva strcmp() nad onim na šta pokazuju a i b,
    kastovani na odgovarajući tip.
31  *****/
    int poredi_leksikografski(const void *a, const void *b)
33  {
        return strcmp(*(char **) a, *(char **) b);
35  }

37  /* Funkcija slicna prethodnoj, osim sto elemente ne upoređuje
    leksikografski, vec po duzini */
39  int poredi_duzine(const void *a, const void *b)
    {
41      return strlen(*(char **) a) - strlen(*(char **) b);
    }
43
    /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
45  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) * 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        niski 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");

/* Unos trazene niske */
printf("Uneti trazenu nisku: ");
99 scanf("%s", x);

/* 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");

/* 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");

/* Sortiramo po duzini */
qsort(niske, n, sizeof(char *), &poredi_duzine);
119

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

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

129 exit(EXIT_SUCCESS);
}

```

### Rešenje 3.34

```

#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 poredi. */
36 int poredi2(const void *a, const void *b)
{
38     int bodovi = *(int *) a;
   Student *s = (Student *) b;
40     return s->bodovi - bodovi;
}

42 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
44 Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
   parametar je element niza cije se prezime poredi. */
46 int poredi3(const void *a, const void *b)
{
48     char *prezime = (char *) a;
   Student *s = (Student *) b;
50     return strcmp(prezime, s->prezime);
}

52 int main(int argc, char *argv[])
54 {
   Student kolokvijum[MAX];
56     int i;
   size_t br_studenata = 0;
58     Student *nadjen = NULL;
   FILE *fp = NULL;
60     int bodovi;
   char prezime[21];

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

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

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

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

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

92     printf("Studenti sortirani po broju poena opadajuće, ");
94     printf("pa po prezimenu rastuće:\n");
   for (i = 0; i < br_studenata; i++)
96         printf("%s %s %d\n", kolokvijum[i].ime,
               kolokvijum[i].prezime, kolokvijum[i].bodovi);

98     /* Pretraživanje studenata po broju bodova se vrši binarnom
100        pretragom jer je niz sortiran po broju bodova. */
   printf("Unesite broj bodova: ");
102     scanf("%d", &bodovi);

104     nadjen =
        bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),

```

```

106         &poredi2);
108     if (nadjen != NULL)
109         printf
110             ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
111              nadjen->ime, nadjen->prezime, nadjen->bodovi);
112     else
113         printf("Nema studenta sa unetim brojem bodova\n");
114
115     /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116        sortiran po bodovima. */
117     printf("Unesite prezime: ");
118     scanf("%s", prezime);
119
120     nadjen =
121         lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
122              &poredi3);
123
124     if (nadjen != NULL)
125         printf
126             ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
127              nadjen->ime, nadjen->prezime, nadjen->bodovi);
128     else
129         printf("Nema studenta sa unetim prezimenom\n");
130
131     exit(EXIT_SUCCESS);
132 }

```

### Rešenje 3.35

```

#include <stdio.h>
2 #include <string.h>
#include <stdlib.h>
4
#define MAX 128
6
/* Funkcija poredi dva karaktera */
8 int uporedi_char(const void *pa, const void *pb)
9 {
10     return *(char *) pa - *(char *) pb;
11 }
12
/* 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
int main()
29 {
30     char s[MAX], t[MAX];
31
32     /* Unos niski */
33     printf("Unesite prvu nisku: ");
34     scanf("%s", s);
35     printf("Unesite drugu nisku: ");
36     scanf("%s", t);
37
38     /* Ispituje se da li su niske anagrami */
39     if (anagrami(s, t))
40         printf("jesu\n");
41     else
42

```



```

    printf("nisu\n");
44
    return 0;
46 }

```

### Rešenje 3.36

```

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

5  #define MAX 10
   #define MAX_DUZINA 32

7
   /* Funkcija porenjenja */
9  int uporedi_niske(const void *pa, const void *pb)
   {
11     return strcmp((char *) pa, (char *) pb);
   }

13
15 int main()
   {
17     int i, n;
     char S[MAX][MAX_DUZINA];

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

23     /* Unos niza niski */
     printf("Unesite niske:\n");
25     for (i = 0; i < n; i++)
         scanf("%s", S[i]);

27
     /* Sortiranje niza niski */
29     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

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

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

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

47
     printf("nema\n");
49     return 0;
   }

```

### Rešenje 3.37

```

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

5  #define MAX 21

7  /* Struktura koja predstavlja jednog studenta */
   typedef struct student {
9     char nalog[8];
     char ime[MAX];

```

```

11     char prezime[MAX];
12     int poeni;
13 } Student;

14
15 /* Funkcija poredi studente prema broju poena, rastuce */
16 int uporedi_poeni(const void *a, const void *b)
17 {
18     Student s = *(Student *) a;
19     Student t = *(Student *) b;
20     return s.poeni - t.poeni;
21 }

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

48
49 int uporedi_bsearch(const void *a, const void *b)
50 {
51     /* Nalog studenta koji se trazi */
52     char *nalog = (char *) a;
53     /* Kljuc pretrage */
54     Student s = *(Student *) b;
55
56     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
57     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
58     char smer1 = nalog[1];
59     char smer2 = s.nalog[1];
60     int indeks1 =
61         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0';
62     int indeks2 =
63         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
64         s.nalog[6] - '0';
65     if (godina1 != godina2)
66         return godina1 - godina2;
67     else if (smer1 != smer2)
68         return smer1 - smer2;
69     else
70         return indeks1 - indeks2;
71 }

72
73 int main(int argc, char **argv)
74 {
75     Student *nadjen = NULL;
76     char nalog_trazeni[8];
77     Student niz_studenata[100];
78     int i = 0, br_studenata = 0;
79     FILE *in = NULL, *out = NULL;

80
81     /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
82        korisnik nije ispravno pozvao program i prijavljuje se greska. */
83     if (argc != 2 && argc != 3) {

```

```

85     fprintf(stderr,
        "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n");
86     exit(EXIT_FAILURE);
87 }

89 /* Otvaranje datoteke za citanje */
in = fopen("studenti.txt", "r");
91 if (in == NULL) {
    fprintf(stderr,
93         "Greska prilikom otvaranja datoteke studenti.txt!\n");
    exit(EXIT_FAILURE);
95 }

97 /* Otvaranje datoteke za pisanje */
out = fopen("izlaz.txt", "w");
99 if (out == NULL) {
    fprintf(stderr,
101        "Greska prilikom otvaranja datoteke izlaz.txt!\n");
    exit(EXIT_FAILURE);
103 }

105 /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
while (fscanf
107     (in, "%s %s %s %d", niz_studenata[i].nalog,
        niz_studenata[i].ime, niz_studenata[i].prezime,
109         &niz_studenata[i].poeni) != EOF)
    i++;

111 br_studenata = i;

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

123 /* Sortirani studenti se ispisuju u izlaznu datoteku */
for (i = 0; i < br_studenata; i++)
125     fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
        niz_studenata[i].ime, niz_studenata[i].prezime,
127         niz_studenata[i].poeni);

129 /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
    studenta... */
131 if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
    strcpy(nalog_trazeni, argv[2]);
133
    /* ... pronalazi se student sa tim nalogom... */
135     nadjen =
        (Student *) bsearch(nalog_trazeni, niz_studenata,
137                             br_studenata, sizeof(Student),
                                &uporedi_bsearch);
139
    if (nadjen == NULL)
141        printf("Nije nadjen!\n");
    else
143        printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
            nadjen->prezime, nadjen->poeni);
145 }

147 /* Zatvaranje datoteka */
fclose(in);
149 fclose(out);

151 exit(EXIT_SUCCESS);
}

```



## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

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

- (a) Definirati strukturu `Cvor` kojom se predstavlja čvor liste. Čvor treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.
- (b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje mu polja i vraća njegovu adresu.
- (c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.
- (d) Napisati funkciju `Cvor *pronadji_poslednji(Cvor * glava)` koja pronalazi poslednji čvor u listi.
- (e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste.
- (f) Napisati funkciju `Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.
- (g) Napisati funkciju `int dodaj_iza(Cvor * tekuci, int broj)` koja iza čvora `tekuci` dodaje novi čvor sa vrednošću `broj`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi element 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 zarezima.
- (j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.
- (k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.
- (l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.

- (n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. NAPOMENA: *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

##### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [3, 2]
14
Lista: [14, 3, 2]
5
Lista: [5, 14, 3, 2]
3
Lista: [3, 5, 14, 3, 2]
17
Lista: [17, 3, 5, 14, 3, 2]

Unosite broj koji se trazi: 5
Trazeni broj 5 je u listi!
```

##### Primer 2

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

Unosite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!
```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

##### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 14, 3]
3
Lista: [2, 3, 14, 3, 3]
17
Lista: [2, 3, 14, 3, 3, 17]
3
Lista: [2, 3, 14, 3, 3, 17, 3]

Unosite broj koji se brise: 3
Lista nakon brisanja: [2, 14, 17]
```

##### Primer 2

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

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

- (3) U glavnom programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretaku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 3, 14]
3
Lista: [2, 3, 3, 3, 14]
5
Lista: [2, 3, 3, 3, 5, 14]

Unesite broj koji se trazi: 14
Trazeni broj 14 je u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 5, 14]

```

*Primer 2*

```

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

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

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

```

[Rešenje 4.1]

**Zadatak 4.2** Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekurzivno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu:

```

typedef struct _Element
{
    unsigned broj_pojavljivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;

```

*Test 1*

```

POZIV: ./a.out datoteka.html

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

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

```

*Test 2*

```

POZIV: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja
datoteke datoteka.html.

```

[Rešenje 4.3]

**Zadatak 4.4** U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan

po jedan, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku **da** ili **ne**, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

<p><i>Primer 1</i></p> <pre>Poziv: ./a.out studenti.txt  STUDENTI.TXT 123/2014 Marko Lukic 3/2014 Ana Sokic 43/2013 Jelena Ilic 41/2009 Marija Zaric 13/2010 Milovan Lazic  INTERAKCIJA SA PROGRAMOM: 3/2014 da: Ana Sokic 235/2008 ne 41/2009 da: Marija Zaric</pre>	<p><i>Primer 2</i></p> <pre>Poziv: ./a.out studenti.txt  DATOTEKA STUDENTI.TXT JE PRAZNA  INTERAKCIJA SA PROGRAMOM: 3/2014 ne 235/2008 ne 41/2009 ne</pre>
---	--

[Rešenje 4.4]

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

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

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

<p><i>Test 1</i></p> <pre>BROJEVI.TXT 43 12 15 16 4 2 8  IZLAZ: REZULTAT.TXT 12 15 16</pre>	<p><i>Test 2</i></p> <pre>DATOTEKA BROJEVI.TXT NE POSTOJI.  IZLAZ: REZULTAT.TXT Greska prilikom otvaranja datoteke brojevi.txt.</pre>	<p><i>Test 3</i></p> <pre>DATOTEKA BROJEVI.TXT JE PRAZNA  IZLAZ: REZULTAT.TXT Rezultat.txt ce biti prazna.</pre>
---	---	--

**Zadatak 4.6** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

<p><i>Test 1</i></p> <pre>Poziv: ./a.out dat1.txt dat2.txt  DAT1.TXT 2 4 6 10 15  DAT2.TXT 5 6 11 12 14 16  IZLAZ: [2, 4, 5, 6, 6, 10, 11, 12, 14, 15, 16]</pre>	<p><i>Test 2</i></p> <pre>Poziv: ./a.out dat1.txt dat2.txt  DAT1.TXT 2 4 6 10 15  DATOTEKA DAT2.TXT NE POSTOJI.  IZLAZ: Greska prilikom otvaranja datoteke dat2.txt.</pre>
<p><i>Test 3</i></p> <pre>Poziv: ./a.out dat1.txt dat2.txt  DATOTEKA DAT1.TXT JE PRAZNA  DAT2.TXT 5 6 11 12 14 16  IZLAZ: [5, 6, 11, 12, 14, 16]</pre>	<p><i>Test 4</i></p> <pre>Poziv: ./a.out dat1.txt  IZLAZ: Program se poziva sa: ./a.out dat1.txt dat2.txt!</pre>



[Rešenje 4.6]

**Zadatak 4.7** Date su dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od ovih listi formira novu listu L koja sadrži naizmenično raspoređene čvorove listi L1 i L2: prvi čvor iz L1, prvi čvor iz L2, drugi čvor L1, drugi čvor L2, itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *Iskoristiti testove 2 - 6 za zadatak 4.6.*

```
Test 1
| Poziv: ./a.out dat1.txt dat2.txt
|
| DAT1.TXT
| 2 4 6 10 15
|
| DAT2.TXT
| 5 6 11 12 14 16
|
| Izlaz:
| 2 5 4 6 6 11 10 12 15 14 16
```

**Zadatak 4.8** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

```
Test 1
| IZRAZ.TXT
| {[23 + 5344] * (24 - 234)} - 23
|
| Izlaz:
| Zagrade su ispravno uparene.
```

```
Test 2
| IZRAZ.TXT
| {[23 + 5] * (9 * 2)} - {23}
|
| Izlaz:
| Zagrade su ispravno uparene.
```

```
Test 3
| IZRAZ.TXT
| {[2 + 54] / (24 * 87)} + (234 + 23)
|
| Izlaz:
| Zagrade nisu ispravno uparene.
```

```
Test 4
| IZRAZ.TXT
| {(2 - 14) / (23 + 11)} * (2 + 13)
|
| Izlaz:
| Zagrade nisu ispravno uparene.
```

```
Test 5
| DATOTEKA IZRAZ.TXT JE PRAZNA
|
| Izlaz:
| Zagrade su ispravno uparene.
```

```
Test 6
| DATOTEKA IZRAZ.TXT NE POSTOJI.
|
| Izlaz:
| Greska prilikom otvaranja
| datoteke izraz.txt!
```

[Rešenje 4.8]

**Zadatak 4.9** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže HTML etikete.*

*Test 1*

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

DATOTEKA.HTML
<html>
  <head>
    <title>Primer</title>
  </head>
  <body>
  </body>

IZLAZ:
Etikete nisu pravilno uparene
(etiketa <html> nije zatvorena)
```

*Test 2*

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

DATOTEKA.HTML
<head>
  <title>Primer</title>
</head>
<body>
</body>
</html>

IZLAZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html>
koja nije otvorena)
```

*Test 3*

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

DATOTEKA.HTML
<html>
  <head>
    <title>Primer</title>
  </head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    Sutra ce biti jos lepsi.
    <a link='http://www.math.rs'>Link</a>
  </body>
</html>

IZLAZ:
Etikete su pravilno uparene!
```

*Test 4*

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

DATOTEKA.HTML
<html>
  <head>
    <title>Primer</title>
  </head>
  <body>
  </html>

IZLAZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html>, a poslednja
otvorena je <body>)
```

*Test 5*

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

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja
datoteke datoteka.html.
```

*Test 6*

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

DATOTEKA.HTML JE PRAZNA

IZLAZ:
Etikete su pravilno uparene!
```

[Rešenje 4.9]

**Zadatak 4.10** Napisati program koji pomaže službeniku u radu na šalteru. Službenik najpre evidentira sve korisničke JMBG brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (EOF). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije. Program mu postavlja pitanje *Da li korisnika vratate na kraj reda?* i ukoliko on da odgovor *Da*, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije *Da*, službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke *izvestaj.txt*. Ova datoteka, za svaki obrađen zahtev, sadrži JMBG i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red 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.

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

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

[Rešenje 4.11]

**Zadatak 4.12** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju

sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

Test 1	Test 2	Test 3
ULAZ: 5 3	ULAZ: 8 4	ULAZ: 3 8
IZLAZ: 3 1 5 2 4	IZLAZ: 4 8 5 2 1 3 7 6	IZLAZ: n mora biti uvek vece od k, a 3 < 8!

**Zadatak 4.13** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smeru. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

Test 1	Test 2	Test 3
ULAZ: 5 3	ULAZ: 8 4	ULAZ: 5 8
IZLAZ: 3 5 4 2 1	IZLAZ: 4 8 5 7 6 3 2 1	IZLAZ: 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 alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.
- Napisati funkciju `int dodaj_u_stablo(Cvor ** adresa_korena, int broj)` koja u stablo na koje pokazuje argument `adresa_korena` dodaje ceo broj `broj`. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.
- Napisati funkciju `Cvor *pretrazi_stablo(Cvor * koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.
- Napisati funkciju `Cvor *pronadji_najmanji(Cvor * koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- Napisati funkciju `Cvor *pronadji_najveci(Cvor * koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- Napisati funkciju `void obrisi_element(Cvor ** adresa_korena, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `adresa_korena`.
- 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
Traži se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuće stablo: 1 2 9 18 32
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Traži se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuće stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

**Zadatak 4.15** Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku *Nedostaje ime ulazne datoteke!*. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

*Test 1*

```
POZIV: ./a.out test.txt
TEST.TXT
Sunce utorak raCunar SUNCE programiranje
jabuka PROGramiranje sunCE JABUka
IZLAZ:
jabuka: 2
programiranje: 2
racunar: 1
sunce: 3
utorak: 1
Najcesca rec: sunce (pojavljuje se 3 puta)
```

*Test 2*

```
POZIV: ./a.out suma.txt
SUMA.TXT
lipa zova hrast ZOVA breza LIPA
IZLAZ:
breza: 1
hrast: 1
lipa: 2
zova: 2
Najcesca rec: lipa
(pojavljuje se 2 puta)
```

*Test 3*

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

*Test 4*

```
POZIV: ./a.out
IZLAZ:
Nedostaje ime ulazne datoteke!
```

[Rešenje 4.15]

**Zadatak 4.16** U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. *Pera Peric 064/123-4567*. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči *KRAJ*, a za svaki od unetih podataka ispisuje se ili broj telefona ili

obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

*Primer 1*

```
IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858

INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik.txt
Unesite ime i prezime: Pera Peric
Broj je: 011/3240-987
Unesite ime i prezime: Marko Markovic
Broj nije u imeniku!
Unesite ime i prezime: KRAJ
```

*Primer 2*

```
DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik1.txt
Greska: Neuspesno otvaranje datoteke
imenik1.txt.
```

[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

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

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

## Primer 2

```
POZIV: ./a.out rodjendani.txt
DATOTEKA RODJENDANI.TXT NE POSTOJI

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

[Rešenje 4.18]

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

## Primer 1

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

## Primer 2

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

[Rešenje 4.19]

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

## Primer 1

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

## Primer 2

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

[Rešenje 4.20]

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

*Primer 1*

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

*Primer 2*

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

[Rešenje 4.21]

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

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

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

*Test 1*

```
POZIV: ./a.out 2 15
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*

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

[Rešenje 4.22]

**Zadatak 4.23** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivou. 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	Test 2
<pre> ULAZ:   10 5 15 2 11 16 1 13  IZLAZ:   7 </pre>	<pre> ULAZ:   16 30 40 24 10 18 45 22  IZLAZ:   6 </pre>

[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. 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.*

```

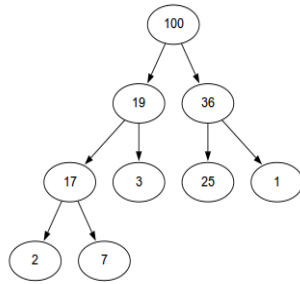
Test 1
IZLAZ:
  Zadato stablo je hip!

```

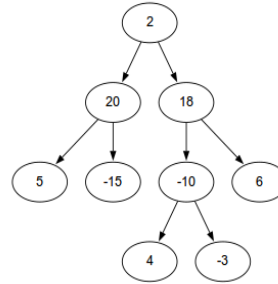
[Rešenje 4.26]

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

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

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

Test 1

```

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

## 4.3 Rešenja

### Rešenje 4.1

Datoteka 4.1: lista.h

```

1  #ifndef _LISTA_H_
2  #define _LISTA_H_

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

10
11 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12 dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   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,
   inace vraca 0. */
42 int dodaj_sortirano(Cvor ** adresa_glave, int broj);

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

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

52 /* 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. */
54 void obrisi_cvor(Cvor ** adresa_glave, int broj);

56 /* 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. */
58 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

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

64 #endif

```

Datoteka 4.2: lista.c

```

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

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

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

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

21 void oslobodi_listu(Cvor ** adresa_glave)
   {
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 cvor koji predstavlja glavu liste */
29      pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);

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

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

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

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

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

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

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

65
67     return glava;
69 }

71 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
73     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
75     if (novi == NULL)
        return 1;

77
    /* Ako je lista prazna */
79     if (*adresa_glave == NULL) {
        /* Glava nove liste je upravo novi cvor */
81         *adresa_glave = novi;
    } else {
83         /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
            se dodaje na kraj liste kao sledbenik poslednjeg */
85         Cvor *poslednji = pronadji_poslednji(*adresa_glave);
            poslednji->sledeci = novi;
87     }

89
    /* Vraca se indikator uspesnog dodavanja */
    return 0;
91 }

93 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
95     /* U praznoj listi nema takvog mesta i vraca se NULL */
    if (glava == NULL)

```

```

97     return NULL;

99     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
101     pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
    vecu ili jednaku vrednosti novog cvora.

103     Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
    provera da li se doslo do poslednjeg cvora liste pre nego sto se
105     proveru vrednost u sledecem cvoru, jer u slucaju poslednjeg,
    sledeci ne postoji, pa ni njegova vrednost. */
107     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
        glava = glava->sledeci;

109

111     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
    poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
    vrednost vecu od broj. */
113     return glava;
115 }

117 int dodaj_iza(Cvor * tekuci, int broj)
118 {
119     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
120     Cvor *novi = napravi_cvor(broj);
121     if (novi == NULL)
122         return 1;

123     /* Novi cvor se dodaje iza tekuceg cvora. */
124     novi->sledeci = tekuci->sledeci;
125     tekuci->sledeci = novi;

127     /* Vraca se indikator uspesnog dodavanja */
128     return 0;
129 }

131 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
132 {
133     /* Ako je lista prazna */
134     if (*adresa_glave == NULL) {
135         /* Glava nove liste je novi cvor */

137         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
138         Cvor *novi = napravi_cvor(broj);
139         if (novi == NULL)
140             return 1;

141         *adresa_glave = novi;

143         /* Vraca se indikator uspesnog dodavanja */
144         return 0;
145     }

147     /* Inace... */

149     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
    treba dodati na pocetak liste */
151     if ((*adresa_glave)->vrednost >= broj) {
153         return dodaj_na_pocetak_liste(adresa_glave, broj);
154     }

155     /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
    tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
157     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
159     return dodaj_iza(pomocni, broj);
160 }

161 Cvor *pretrazi_listu(Cvor * glava, int broj)
162 {
163     /* Obilaze se cvorovi liste */
164     for (; glava != NULL; glava = glava->sledeci)
165     /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
    se obustavlja */
167     if (glava->vrednost == broj)
169         return glava;

```

```

171  /* Nema trazenog broja u listi i vraca se NULL */
172  return NULL;
173 }

175 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
176 {
177     /* Obilaze se cvorovi liste */
178     /* U uslovu ostanka u petlji, bitan je redosled proveru u
179     konjunkciji */
180     for (; glava != NULL && glava->vrednost <= broj;
181         glava = glava->sledeci)
182         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
183         se obustavlja */
184         if (glava->vrednost == broj)
185             return glava;

187     /* Nema trazenog broja u listi i vraca se NULL */
188     return NULL;
189 }

191 void obrisi_cvor(Cvor ** adresa_glave, int broj)
192 {
193     Cvor *tekuci = NULL;
194     Cvor *pomocni = NULL;

195     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
196     broju i azurira se pokazivac na glavu liste */
197     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj) {
198         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
199         adresi adresa_glave */
200         pomocni = (*adresa_glave)->sledeci;
201         free(*adresa_glave);
202         *adresa_glave = pomocni;
203     }

205     /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
206     funkcije */
207     if (*adresa_glave == NULL)
208         return;

209     /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
210     pokazuje na cvor cija je vrednost razlicita od trazenog broja.
211     Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
212     sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
213     ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
214     Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora. */
215     tekuci = *adresa_glave;
216     while (tekuci->sledeci != NULL)
217     {
218         if (tekuci->sledeci->vrednost == broj) {
219             /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
220             prvo cuva u promenljivoj pomocni. */
221             pomocni = tekuci->sledeci;
222             /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
223             njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
224             sledeci od cvora koji se brise. */
225             tekuci->sledeci = pomocni->sledeci;
226             /* Sada treba osloboditi cvor sa vrednoscu broj. */
227             free(pomocni);
228         } else {
229             /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
230             pomera na sledeci. */
231             tekuci = tekuci->sledeci;
232         }
233     }
234     return;
235 }

237 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
238 {
239     Cvor *tekuci = NULL;
240     Cvor *pomocni = NULL;

241     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom

```

```

243     broju i azurira se pokazivac na glavu liste. */
while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj) {
245     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
        adresi adresa_glave. */
247     pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
249     *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
        je veca od broja, jer kako je lista sortirana rastuce nema
255     potrebe broj traziti u repu liste. */
    if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
257         return;

259     /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
        na cvor cija vrednost je manja od trazenog broja, kao i svim
261     cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
        je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
263     ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
        cija vrednost je veca od trazenog broja. */
    tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj)
267         if (tekuci->sledeci->vrednost == broj) {
            pomocni = tekuci->sledeci;
269            tekuci->sledeci = tekuci->sledeci->sledeci;
            free(pomocni);
271        } else {
            /* Ne treba brisati sledeceg od tekuceg jer je manji od
273            trazenog i tekuci se pomera na sledeci cvor. */
            tekuci = tekuci->sledeci;
275        }
    return;
277 }

279 void ispisi_listu(Cvor * glava)
{
281     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
        jer se lista nece menjati */
283     putchar('[');
    /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
285     pocetka prema kraju liste. */
    for (; glava != NULL; glava = glava->sledeci) {
287         printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
289             printf(", ");
    }
    printf("]\n");
291 }

```

Datoteka 4.3: *main\_a.c*

```

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

4
/* 1) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku */
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
10     int broj;

12
    /* Testiranje dodavanja novog broja na pocetak liste */
14     printf("Unesite brojeve: (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
            memorije za nov cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa. */
18         if (dodaj_na_pocetak_liste(&glava, broj) == 1) {

```

```
20     fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
    oslobodi_listu(&glava);
22     exit(EXIT_FAILURE);
    }
24     printf("\tLista: ");
    ispisi_listu(glava);
26 }

/* Testiranje funkcije za pretragu liste */
28 printf("\nUnesite broj koji se trazi: ");
30 scanf("%d", &broj);

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

/* Oslobadja se memorija zauzeta listom */
38 oslobodi_listu(&glava);
40
42 exit(EXIT_SUCCESS);
}
```

Datoteka 4.4: *main\_b.c*

```
#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"
4

/* 2) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku */
    Cvor *glava = NULL;
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) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
16         memorije za nov cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa. */
        if (dodaj_na_kraj_liste(&glava, broj) == 1) {
18             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
22         }
        printf("\tLista: ");
24         ispisi_listu(glava);
    }

26

    /* Testira se funkcije kojom se brise cvor liste */
28     printf("\nUnesite broj koji se brise: ");
    scanf("%d", &broj);
30

    /* Brisu se cvorovi iz liste cije je polje vrednost jednako broju
32     procitanom sa ulaza */
    obrisi_cvor(&glava, broj);
34

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

38     /* Oslobadja se memorija zauzeta listom */
    oslobodi_listu(&glava);
40

42     exit(EXIT_SUCCESS);
}
```

Datoteka 4.5: *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) {
21             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
22             oslobodi_listu(&glava);
23             exit(EXIT_FAILURE);
24         }
25         printf("\tLista: ");
26         ispisi_listu(glava);
27     }
28
29     /* Testira se funkcija za pretragu liste */
30     printf("\nUnesite broj koji se trazi: ");
31     scanf("%d", &broj);
32
33     trazeni = pretrazi_listu(glava, broj);
34     if (trazeni == NULL)
35         printf("Broj %d se ne nalazi u listi!\n", broj);
36     else
37         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
38
39     /* Testira se funkcija kojom se brise cvor liste */
40     printf("\nUnesite broj koji se brise: ");
41     scanf("%d", &broj);
42
43     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
44        procitanom sa ulaza */
45     obrisi_cvor_sortirane_liste(&glava, broj);
46
47     printf("Lista nakon brisanja: ");
48     ispisi_listu(glava);
49
50     /* Oslobadja se memorija zauzeta listom */
51     oslobodi_listu(&glava);
52
53     exit(EXIT_SUCCESS);
54 }

```

## Rešenje 4.2

Datoteka 4.6: *lista.h*

```

1  #ifndef _LISTA_H_
2  #define _LISTA_H_
3
4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
5     podatak vrednost i pokazivac na sledeci cvor liste. */
6  typedef struct cvor {
7     int vrednost;
8     struct cvor *sledeci;
9 } Cvor;
10
11 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac

```

```

14     na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
15 Cvor *napravi_cvor(int broj);
16
17 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
18    ciji se pokazivac glava nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave);
20
21 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
22    bilo greske pri alokaciji memorije, inace vraca 0. */
23 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
24
25 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
26    pri alokaciji memorije, inace vraca 0. */
27 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
28
29 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
30    ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
31    memorije, inace vraca 0. */
32 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
33
34 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
35    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
36    NULL u slučaju da takav cvor ne postoji u listi. */
37 Cvor *pretrazi_listu(Cvor * glava, int broj);
38
39 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
40    U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
41    neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
42    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji. */
43 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
44
45 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
46    pokazivac na glavu liste, koji može biti promenjen u slučaju da se
47    obriše stara glava liste. */
48 void obrisi_cvor(Cvor ** adresa_glave, int broj);
49
50 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
51    oslanjajući se na cinjenicu da je prosledjena lista sortirana
52    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
53    promenjen ukoliko se obriše stara glava liste. */
54 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
55
56 /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
57    zaptama. */
58 void ispisi_vrednosti(Cvor * glava);
59
60 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
61    liste, razdvojene zaptama i uokvirene zagradama. */
62 void ispisi_listu(Cvor * glava);
63
64 #endif

```

Datoteka 4.7: lista.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  Cvor *napravi_cvor(int broj)
6  {
7      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8         alokacije */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;
12
13     /* Inicijalizacija polja strukture */
14     novi->vrednost = broj;
15     novi->sledeci = NULL;
16
17     /* Vraca se adresa novog cvora */
18     return novi;

```

```

19 }

21 void oslobodi_listu(Cvor ** adresa_glave)
22 {
23     /* Ako je lista vec prazna */
24     if (*adresa_glave == NULL)
25         return;

27     /* Ako lista nije prazna, treba osloboditi memoriju. Pre
28        oslobadjanja memorije za glavu liste, treba osloboditi rep liste
29     */
30     oslobodi_listu(&(*adresa_glave)->sledeci);
31     /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
32        glava u pozivajucoj funkciji tako da odgovara praznoj listi */
33     free(*adresa_glave);
34     *adresa_glave = NULL;
35 }

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

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

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

51 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
52 {
53     /* Ako je lista prazna */
54     if (*adresa_glave == NULL) {

56         /* Novi cvor postaje glava liste */
57         Cvor *novi = napravi_cvor(broj);
58         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */
59         if (novi == NULL)
60             return 1;

61         /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
62            azurira i pokazivacka promenljiva u pozivajucoj funkciji */
63         *adresa_glave = novi;

65         /* Vraca se indikator uspesnog dodavanja */
66         return 0;
67     }

69     /* Ako lista nije prazna, broj se dodaje u rep liste. */
70     /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
71        novi broj se dodaje u rep liste u rekurzivnom pozivu.
72        Informaciju o uspesnosti alokacije u rekurzivnom pozivu funkcija
73        prosledjuje visem rekurzivnom pozivu koji tu informaciju vraca u
74        rekurzivni poziv iznad, sve dok se ne vrati u main. Tek je iz
75        main funkcije moguće pristupiti pravom pocetku liste i
76        osloboditi je celu, ako ima potrebe. Ako je funkcija vratila 0,
77        onda nije bilo greske. */
78     return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
79 }

81 }

83 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
84 {
85     /* Ako je lista prazna */
86     if (*adresa_glave == NULL) {

88         /* Novi cvor postaje glava liste */
89         Cvor *novi = napravi_cvor(broj);

91         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */

```

```

    if (novi == NULL)
        return 1;

    /* Azurira se glava liste */
    *adresa_glave = novi;

    /* Vraca se indikator uspesnog dodavanja */
    return 0;
}

/* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
   glavi liste, onda se dodaje na pocetak liste */
if ((*adresa_glave)->vrednost >= broj)
    return dodaj_na_pocetak_liste(adresa_glave, broj);

/* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
   bude sortirana lista. */
return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
}

Cvor *pretrazi_listu(Cvor * glava, int broj)
{
    /* U praznoj listi nema vrednosti */
    if (glava == NULL)
        return NULL;

    /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
    if (glava->vrednost == broj)
        return glava;

    /* Inace, pretraga se nastavlja u repu liste */
    return pretrazi_listu(glava->sledeci, broj);
}

Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
    /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
       vrednosti u glavi liste, jer je lista neopadajuće sortirana */
    if (glava == NULL || glava->vrednost > broj)
        return NULL;

    /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
    if (glava->vrednost == broj)
        return glava;

    /* Inace, pretraga se nastavlja u repu liste */
    return pretrazi_listu(glava->sledeci, broj);
}

void obrisi_cvor(Cvor ** adresa_glave, int broj)
{
    /* U praznoj listi nema cvorova za brisanje. */
    if (*adresa_glave == NULL)
        return;

    /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
    obrisi_cvor(&(*adresa_glave)->sledeci, broj);

    /* Preostaje proveriti da li glavu liste treba obrisati */
    if ((*adresa_glave)->vrednost == broj) {
        /* Pomocni pokazuje na cvor koji treba da se obrise */
        Cvor *pomocni = *adresa_glave;
        /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
           brise se cvor koji je bio glava liste. */
        *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
    }
}

void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
    /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca

```

```

165     od broja, kako je lista sortirana rastuce nema potrebe broj
        traziti u repu liste i zato se funkcija prekida */
167     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;
169
        /* Brisu se cvorovi iz repa koji imaju vrednost broj */
171     obrisi_cvor(&(*adresa_glave)->sledeci, broj);

173     /* Preostaje provera da li glavu liste treba obrisati */
    if ((*adresa_glave)->vrednost == broj) {
175         /* Pomocni pokazuje na cvor koji treba da se obrise */
        Cvor *pomocni = *adresa_glave;
177         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
            brise se cvor koji je bio glava liste */
179         *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
181     }
    }
183
    void ispisi_vrednosti(Cvor * glava)
185 {
        /* Prazna lista */
187         if (glava == NULL)
            return;
189
        /* Ispisuje se vrednost u glavi liste */
191         printf("%d", glava->vrednost);

193         /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
            se poziva ista funkcija za ispis ostalih. */
195         if (glava->sledeci != NULL) {
            printf(", ");
197             ispisi_vrednosti(glava->sledeci);
        }
199     }

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

```

### Rešenje 4.3

```

1  #include <stdio.h>
2  #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 izvedena */
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));

```

```

23     if (novi == NULL)
24         return NULL;

25     /* Inicijalizacija polja strukture */
26     novi->broj_pojavljivanja = br;
27     strcpy(novi->etiketa, etiketa);
28     novi->sledeci = NULL;

29     /* Vraca se adresa novog cvora */
30     return novi;
31 }

32
33 /* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
34 void oslobodi_listu(Cvor ** adresa_glave)
35 {
36     Cvor *pomocni = NULL;

37     /* Sve dok lista ni bude prazna, brise se tekuca glava liste i
38        azurira se vrednost glave liste */
39     while (*adresa_glave != NULL) {
40         pomocni = (*adresa_glave)->sledeci;
41         free(*adresa_glave);
42         *adresa_glave = pomocni;
43     }
44     /* Nakon izlaska iz petlje pokazivac glava u main funkciji koji se
45        nalazi na adresi adresa_glave bice postavljen na NULL vrednost. */
46 }

47
48 /* Funkcija dodaje novi cvor na pocetak liste. Povratna vrednost je 1
49    ako je doslo do greske pri alokaciji memorije za nov cvor, odnosno
50    0 */
51 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
52                             char *etiketa)
53 {
54     /* Kreira se novi cvor i proverava se uspesnost alokacije */
55     Cvor *novi = napravi_cvor(br, etiketa);
56     if (novi == NULL)
57         return 1;

58     /* Dodaje se novi cvor na pocetak liste */
59     novi->sledeci = *adresa_glave;
60     *adresa_glave = novi;

61     /* Vraca se indikator uspesnog dodavanja */
62     return 0;
63 }

64
65 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
66    NULL ako takav cvor ne postoji u listi */
67 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
68 {
69     Cvor *tekuci;

70     /* Obilazi se cvor po cvor liste */
71     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
72         /* Ako tekuci cvor sadrzi trazenu etiketu, vracamo njegovu
73            vrednost */
74         if (strcmp(tekuci->etiketa, etiketa) == 0)
75             return tekuci;

76     /* Cvor nije pronadjen */
77     return NULL;
78 }

79
80 /* Funkcija ispisuje sadrzaj liste */
81 void ispisi_listu(Cvor * glava)
82 {
83     /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
84        broj njihovog pojavljivanja u HTML datoteci. */
85     for (; glava != NULL; glava = glava->sledeci)
86         printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
87 }

```

```

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         exit(EXIT_FAILURE);
103     }
104
105     /* Priprema datoteke za citanje */
106     FILE *in = NULL;
107     in = fopen(argv[1], "r");
108     if (in == NULL) {
109         fprintf(stderr,
110             "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
111         exit(EXIT_FAILURE);
112     }
113
114     char c;
115     int i = 0;
116     char procitana[MAX_DUZINA];
117     Cvor *glava = NULL;
118     Cvor *trazeni = NULL;
119
120     /* Cita se karakter po karakter datoteke sve dok se ne procita cela
121        datoteka */
122     while ((c = fgetc(in)) != EOF) {
123
124         /* Proverava se da li se pocinje sa citanjem nove etikete */
125         if (c == '<') {
126             /* Proverava se da li se cita zatvarajuca etiketa */
127             if ((c = fgetc(in)) == '/') {
128                 i = 0;
129                 while ((c = fgetc(in)) != '>')
130                     procitana[i++] = c;
131             }
132             /* Cita se otvarajuca etiketa */
133             else {
134                 i = 0;
135                 procitana[i++] = c;
136                 while ((c = fgetc(in)) != ' ' && c != '>')
137                     procitana[i++] = c;
138             }
139             procitana[i] = '\0';
140
141             /* Trazi se procitana etiketa medju postojećim cvorovima liste.
142                Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu sa
143                brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
144                etikete */
145             trazeni = pretrazi_listu(glava, procitana);
146             if (trazeni == NULL) {
147                 if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
148                     fprintf(stderr, "Neuspela alokacija za nov cvor\n");
149                     oslobodi_listu(&glava);
150                     exit(EXIT_FAILURE);
151                 }
152             } else
153                 trazeni->broj_pojavljivanja++;
154         }
155     }
156
157     /* Zatvaranje datoteke */
158     fclose(in);
159
160     /* Ispisuje se sadržaj cvorova liste */
161     ispisi_listu(glava);
162
163     /* Oslobadja se memorija zauzeta listom */
164     oslobodi_listu(&glava);
165
166     exit(EXIT_SUCCESS);
167 }

```

## Rešenje 4.4

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21
7
8 /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
   studentu */
9
10 typedef struct _Cvor {
11     char broj_indeksa[MAX_INDEKS];
12     char ime[MAX_IME_PREZIME];
13     char prezime[MAX_IME_PREZIME];
14     struct _Cvor *sledeci;
15 } Cvor;
16
17 /* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
   novi cvor ili NULL ukoliko je doslo do greske */
18
19 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
21     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
       alokacije */
22     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
23     if (novi == NULL)
24         return NULL;
25
26     /* Inicijalizacija polja strukture */
27     strcpy(novi->broj_indeksa, broj_indeksa);
28     strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
30     novi->sledeci = NULL;
31
32     /* Vraca se adresa novog cvora */
33     return novi;
34 }
35
36 /* Funkcija oslobadja memoriju zauzetu cvorovima liste */
37
38 void oslobodi_listu(Cvor ** adresa_glave)
39 {
40     /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
41     if (*adresa_glave == NULL)
42         return;
43
44     /* Rekurzivnim pozivom se oslobadja rep liste */
45     oslobodi_listu(&(*adresa_glave)->sledeci);
46
47     /* Potom se oslobadja i glava liste */
48     free(*adresa_glave);
49
50     /* Proglasava se lista praznom */
51     *adresa_glave = NULL;
52 }
53
54 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
   do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
55
56 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
57                             char *ime, char *prezime)
58 {
59     /* Kreira se novi cvor i proverava se uspesnost alokacije */
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
61     if (novi == NULL)
62         return 1;
63
64     /* Dodaje se novi cvor na pocetak liste */
65     novi->sledeci = *adresa_glave;
66     *adresa_glave = novi;
67
68     /* Vraca se indikator uspesnog dodavanja */
69     return 0;
70 }
```



```

72 /* Funkcija ispisuje sadrzaj cvorova liste. */
void ispisu_listu(Cvor * glava)
74 {
    /* Pocevsi od glave liste */
76     for (; glava != NULL; glava = glava->sledeci)
        printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
78             glava->prezime);
    }
80
/* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj indeksa.
U suprotnom funkcija vraca NULL */
82 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
84 {
    /* Ako je lista prazna, ne postoji trazeni cvor */
86     if (glava == NULL)
        return NULL;
88
    /* Poredi se trazeni broj indeksa sa brojem indeksa u glavi liste */
90     if (!strcmp(glava->broj_indeksa, broj_indeksa))
        return glava;
92
    /* Ukoliko u glavi liste nije trazeni indeks, pretraga se nastavlja
u repu liste */
94     return pretrazi_listu(glava->sledeci, broj_indeksa);
96 }

98 int main(int argc, char **argv)
99 {
100     /* Argumenti komandne linije su neophodni jer se iz komandne linije
dobija ime datoteke sa informacijama o studentima */
102     if (argc != 2) {
        fprintf(stderr,
104             "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
        exit(EXIT_FAILURE);
106     }

108     /* Priprema datoteke za citanje */
    FILE *in = NULL;
110     in = fopen(argv[1], "r");
    if (in == NULL) {
112         fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
114         exit(EXIT_FAILURE);
    }
116

    /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
listu */
118     char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
120     char broj_indeksa[MAX_INDEKS];
    Cvor *glava = NULL;
122     Cvor *trazeni = NULL;

124     /* Ucitavanje vrednosti u listu */
    while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
126         if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
            fprintf(stderr, "Neuspela alokacija za nov cvor\n");
128             oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
130         }

132     /* Datoteka vise nije potrebna i zatvara se. */
    fclose(in);
134

    /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
136     while (scanf("%s", broj_indeksa) != EOF) {
        trazeni = pretrazi_listu(glava, broj_indeksa);
138         if (trazeni == NULL)
            printf("ne\n");
        else
140             printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
142     }

144     /* Oslobadja se memorija zauzeta za cvorove liste. */

```

```

    oslobodi_listu(&glava);
146
    exit(EXIT_SUCCESS);
148 }

```

### Rešenje 4.6

```

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

5  /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
   na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7  pokazivaca postojećih cvorova listi */
Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9  {
   /* Pokazivaci na pocetne cvorove listi koje se prevezuju */
11  Cvor *lista1 = *adresa_glave_1;
   Cvor *lista2 = *adresa_glave_2;

13
   /* Pokazivac na pocetni cvor rezultujuće liste */
15  Cvor *rezultujuca = NULL;
   Cvor *tekuci = NULL;

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

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

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

29
   /* Odredjuje se prvi cvor rezultujuće liste - to je ili prvi cvor
   liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
   koji sadrzi manju vrednost */
31  if (lista1->vrednost < lista2->vrednost) {
       rezultujuca = lista1;
       lista1 = lista1->sledeci;
33  } else {
       rezultujuca = lista2;
       lista2 = lista2->sledeci;
35  }

37
   /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste, ne
   sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
   koji sadrzi adresu trenutnog cvora rezultujuće liste */
41  tekuci = rezultujuca;

43

45  /* U svakoj iteraciji petlje rezultujućoj listi se dodaje novi cvor
   tako da bude uređena neopadajuće. Pokazivac na listu iz koje se
   uzima cvor se azurira tako da pokazuje na sledeci cvor */
47  while (lista1 != NULL && lista2 != NULL) {
       if (lista1->vrednost < lista2->vrednost) {
49           tekuci->sledeci = lista1;
           lista1 = lista1->sledeci;
51       } else {
           tekuci->sledeci = lista2;
           lista2 = lista2->sledeci;
53       }
       tekuci = tekuci->sledeci;
55   }

57

59  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
   rezultujuću 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
69     pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
    prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
    se moze doci prateci pokazivace iz glave rezultujuce liste, tako
71     da stare pokazivace treba postaviti na NULL. */
    *adresa_glave_1 = NULL;
73     *adresa_glave_2 = NULL;

75     return rezultujuca;
    }

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     /* Otvaramo datoteke u kojima se nalaze elementi listi */
87     FILE *in1 = NULL;
88     in1 = fopen(argv[1], "r");
89     if (in1 == NULL) {
90         fprintf(stderr,
91             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
92         exit(EXIT_FAILURE);
93     }

94     FILE *in2 = NULL;
95     in2 = fopen(argv[2], "r");
96     if (in2 == NULL) {
97         fprintf(stderr,
98             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
99         exit(EXIT_FAILURE);
100    }

101    /* Liste su na pocetku prazne */
102    int broj;
103    Cvor *lista1 = NULL;
104    Cvor *lista2 = NULL;

105    /* Ucitavanje listi */
106    while (fscanf(in1, "%d", &broj) != EOF)
107        dodaj_na_kraj_liste(&lista1, broj);

108    while (fscanf(in2, "%d", &broj) != EOF)
109        dodaj_na_kraj_liste(&lista2, broj);

110    /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
111    fclose(in1);
112    fclose(in2);

113    /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
114       objedinjavanjem listi */
115    Cvor *rezultat = objedini(&lista1, &lista2);

116    /* Ispis rezultujuce liste. */
117    ispisi_listu(rezultat);

118    /* Lista rezultat dobijena je prevezivanjem cvorova polaznih listi.
119       Njenim oslobadjanjem bice oslobodjena sva zauzeta memorija. */
120    oslobodi_listu(&rezultat);

121    exit(EXIT_SUCCESS);
122 }

```

## Rešenje 4.8

```

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

```

```

3
4 /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
5 predstavljaja zagradu koja se koristi i pokazivac na sledeci cvor
6 liste */
7 typedef struct cvor {
8     char zagrada;
9     struct cvor *sledeci;
10 } Cvor;
11
12 /* Funkcija koja oslobadja memoriju zauzetu stekom */
13 void oslobodi_stek(Cvor ** stek)
14 {
15     Cvor *tekuci;
16     Cvor *pomocni;
17
18     /* Oslobadja se cvor po cvor steka */
19     tekuci = *stek;
20     while (tekuci != NULL) {
21         pomocni = tekuci->sledeci;
22         free(tekuci);
23         tekuci = pomocni;
24     }
25
26     /* Stek se proglašava praznim */
27     *stek = NULL;
28 }
29
30 int main()
31 {
32     /* Stek je na pocetku prazan */
33     Cvor *stek = NULL;
34     FILE *ulaz = NULL;
35     char c;
36     Cvor *pomocni = NULL;
37
38     /* Otvaranje datoteke za citanje izraza */
39     ulaz = fopen("izraz.txt", "r");
40     if (ulaz == NULL) {
41         fprintf(stderr,
42             "Greska prilikom otvaranja datoteke izraz.txt!\n");
43         exit(EXIT_FAILURE);
44     }
45
46     /* Cita se karakter po karakter iz datoteke dok se ne dodje do
47 kraja */
48     while ((c = fgetc(ulaz)) != EOF) {
49         /* Ako je učitana otvorena zagrada, stavlja se na stek */
50         if (c == '(' || c == '{' || c == '[') {
51             /* Alocira se memorija za novi cvor liste i proverava se
52 uspesnost alokacije */
53             pomocni = (Cvor *) malloc(sizeof(Cvor));
54             if (pomocni == NULL) {
55                 fprintf(stderr, "Greska prilikom alokacije memorije!\n");
56                 /* Oslobadja se memorija zauzeta stekom */
57                 oslobodi_stek(&stek);
58                 /* I prekida se sa izvršavanjem programa */
59                 exit(EXIT_FAILURE);
60             }
61
62             /* Inicijalizacija polja strukture */
63             pomocni->zagrada = c;
64
65             /* Promena vrha steka */
66             pomocni->sledeci = stek;
67             stek = pomocni;
68         }
69         /* Ako je učitana zatvorena zagrada, proverava se da li je stek
70 prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
71 otvorena zagrada */
72         else {
73             if (c == ')' || c == '}' || c == ']') {
74                 if (stek != NULL && ((stek->zagrada == '(' && c == ')')
75                     || (stek->zagrada == '{' && c == '}'))

```

```

77         || (stek->zagrada == '[' && c == ']')) {
78             /* Sa vrha steka se uklanja otvorena zagrada */
79             pomocni = stek->sledeci;
80             free(stek);
81             stek = pomocni;
82         } else {
83             /* Inace, zakljucujemo da zagrade u izrazu nisu ispravno
84                uparene */
85             break;
86         }
87     }
88 }
89
90 /* Procitana je cela datoteka i treba je zatvoriti */
91 fclose(ulaz);
92
93 /* Ako je stek prazan i procitana je cela datoteka, zagrade su
94    ispravno uparene */
95 if (stek == NULL && c == EOF)
96     printf("Zagrade su ispravno uparene.\n");
97 else {
98     /* U suprotnom se zakljucuje da zagrade nisu ispravno uparene */
99     printf("Zagrade nisu ispravno uparene.\n");
100    /* Oslobadja se memorija koja je ostala zauzeta stekom */
101    oslobodi_stek(&stek);
102 }
103
104 exit(EXIT_SUCCESS);
105 }

```

## Rešenje 4.9

Datoteka 4.8: *stek.h*

```

1 #ifndef _STEK_H_
2 #define _STEK_H_
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <ctype.h>
8
9 #define MAX 100
10
11 #define OTVORENA 1
12 #define ZATVORENA 2
13
14 #define VAN_ETIKETE 0
15 #define PROCITANO_MANJE 1
16 #define U_ETIKETI 2
17
18 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
19    pokazivac na sledeci cvor */
20 typedef struct cvor {
21     char etiketa[MAX];
22     struct cvor *sledeci;
23 } Cvor;
24
25 /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca 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 slucaju greske pri
33    alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0 */
34 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);
35
36 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument

```

```

38     pokazivac razlicit od NULL, tada u niz karaktera na koji on
    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 moguce 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
52 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
    etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
    Vraca EOF u slucaju da se dodje do kraja datoteke pre 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

```

Datoteka 4.9: *stek.c*

```

#include "stek.h"

2
Cvor *napravi_cvor(char *etiketa)
{
4
    /* Alocira se memorija za novi cvor liste i proverava se uspesnost
        alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;

10
    /* Inicijalizacija polja u novom cvoru */
    if (strlen(etiketa) >= MAX) {
12
        fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
        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
    Cvor *pomocni;

26
    /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
    while (*adresa_vrha != NULL) {
28
        /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
            osloboditi cvor koji predstavlja vrh steka */
        pomocni = *adresa_vrha;
32
        /* Sledeci cvor je novi vrh steka */
        *adresa_vrha = (*adresa_vrha)->sledeci;
        free(pomocni);
34
    }

36
    /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
        adresa_vrha ce pokazivati na NULL. */
38
}

40
int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
{
42
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(etiketa);
    if (novi == NULL)
44
        return 1;
46
}

```

```

48  /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
    novi->sledeci = *adresa_vrha;
50  *adresa_vrha = novi;
    return 0;
52 }

54 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
{
56     Cvor *pomocni;

58     /* Pokušaj skidanja vrednosti sa praznog steka rezultuje greskom i
        vraca se 0 */
60     if (*adresa_vrha == NULL)
        return 0;

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

66     /* Element sa vrha steka se uklanja */
    pomocni = *adresa_vrha;
    *adresa_vrha = (*adresa_vrha)->sledeci;
    free(pomocni);

72     /* Vraca se indikator uspesno izvršene radnje */
74     return 1;
}

76 char *vrh_steka(Cvor * vrh)
78 {
    /* Prazan stek nema cvor koji je vrh i vraca se NULL */
80     if (vrh == NULL)
        return NULL;

82     /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
        koji je na vrhu steka. */
84     return vrh->etiketa;
86 }

88 void prikazi_stek(Cvor * vrh)
{
90     /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
    for (; vrh != NULL; vrh = vrh->sledeci)
92         printf("< %s>\n", vrh->etiketa);
}

94 int uzmi_etiketu(FILE * f, char *etiketa)
96 {
    int c;
    int i = 0;
    /* Stanje predstavlja informaciju dokle se stalo sa citanjem
100     etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
        nije zapoceto citanje. */
102     /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
        OTVORENA ili ZATVORENA. */
104     int stanje = VAN_ETIKETE;
    int tip;

106     /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
        to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
        samo malim slovima. */
108     while ((c = fgetc(f)) != EOF) {
        switch (stanje) {
110             case VAN_ETIKETE:
                if (c == '<')
112                     stanje = PROCITANO_MANJE;
                break;
114             case PROCITANO_MANJE:
                if (c == '/') {
116                     /* Cita se zatvorena etiketa */
                    tip = ZATVORENA;
118                } else {
120

```

```

122         if (isalpha(c)) {
123             /* Cita se otvorena etiketa */
124             tip = OTVORENA;
125             etiketa[i++] = tolower(c);
126         }
127         /* Od sada se cita etiketa i zato se menja stanje */
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 završava
138              * niska koja sadrzi procitanu etiketu i vraća 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 vraća se EOF. */
147 return EOF;
148 }

```

Datoteka 4.10: *main.c*

```

#include "stek.h"
2
int main(int argc, char **argv)
4
{
    /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
6     * nije procitana. */
    Cvor *vrh = NULL;
    char etiketa[MAX];
    int tip;
    int uparene = 1;
    FILE *f = NULL;
12
    /* Ime datoteke se preuzima iz komandne linije. */
14    if (argc < 2) {
        fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
        exit(EXIT_FAILURE);
16    }
18
    /* Datoteka se otvara za citanje */
20    if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
22                argv[1]);
        exit(EXIT_FAILURE);
24    }
26
    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
    while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
28        /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
        * etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
        * potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
        * nemaju sadrzaj (npr link). Zbog jednostavnosti
        * pretpostavlja se da njih nema u HTML dokumentu. */
        if (tip == OTVORENA) {
34            if (strcmp(etiketa, "br") != 0
                && strcmp(etiketa, "hr") != 0
                && strcmp(etiketa, "meta") != 0)
36                if (potisni_na_stek(&vrh, etiketa) == 1) {
38                    fprintf(stderr, "Neuspela alokacija za nov cvor\n");
                    oslobodi_stek(&vrh);
                    exit(EXIT_FAILURE);
40                }
        }
    }
}

```



```

42     }
43     /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
44        u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
45        uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
46        je taj uslov ispunjen, skida se sa steka, jer je upravo
47        zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
48        nisu pravilno uparene. */
49     else if (tip == ZATVORENA) {
50         if (vrh_steka(vrh) != NULL
51             && strcmp(vrh_steka(vrh), etiketa) == 0)
52             skini_sa_steka(&vrh, NULL);
53         else {
54             printf("Etikete nisu pravilno uparene\n");
55             printf("(nadjena je etiketa </%s>", etiketa);
56             if (vrh_steka(vrh) != NULL)
57                 printf(", a poslednja otvorena je </%s>\n", vrh_steka(vrh));
58             else
59                 printf(" koja nije otvorena)\n");
60             uparene = 0;
61             break;
62         }
63     }
64 }
65 /* Završeno je citanje i datoteka se zatvara */
66 fclose(f);
67
68 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
69    da bude prazan. Ukoliko nije, tada postoje etikete koje su
70    ostale otvorene */
71 if (uparene) {
72     if (vrh_steka(vrh) == NULL)
73         printf("Etikete su pravilno uparene!\n");
74     else {
75         printf("Etikete nisu pravilno uparene\n");
76         printf("(etiketa </%s> nije zatvorena)\n", vrh_steka(vrh));
77         /* Oslobadja se memorija zauzeta stekom */
78         oslobodi_stek(&vrh);
79     }
80 }
81
82 exit(EXIT_SUCCESS);
83 }

```

### Rešenje 4.10

Datoteka 4.11: *red.h*

```

1  #ifndef _RED_H_
2  #define _RED_H_
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define MAX 1000
8  #define JMBG_DUZINA 14
9
10 /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11    opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
18    korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
20     Zahtev nalog;
21     struct cvor *sledeci;
22 } Cvor;
23
24 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa

```

```

25     poslate adrese i vraća adresu novog cvora ili NULL ako je došlo 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. */
Cvor *napravi_cvor(Zahtev * zahtev);

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

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

39
/* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
pokazivac razlicit od NULL, tada se u strukturu na koju on
41 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
43 suprotnom. */
45 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

47
/* Funkcija vraća pokazivac na strukturu koja sadrži zahtev korisnika
na pocetku reda. Ukoliko je red prazan funkcija vraća NULL. */
49 Zahtev *pocetak_reda(Cvor * pocetak);

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

55 #endif

```

Datoteka 4.12: *red.c*

```

1  #include "red.h"

3  Cvor *napravi_cvor(Zahtev * zahtev)
{
5      /* Alocira se memorija za novi cvor liste i proverava uspesnost
alokacije */
7      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
      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;
}

35
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                Zahtev * zahtev)
{

```

```

39  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(zahtev);
41  if (novi == NULL)
        return 1;
43
    /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
45     kraj, to je podjednako efikasno kao dodavanje na pocetak liste */
    if (*adresa_kraja != NULL) {
47         (*adresa_kraja)->sledeci = novi;
        *adresa_kraja = novi;
49     } else {
        /* Ako je red bio ranije prazan */
51         *adresa_pocetka = novi;
        *adresa_kraja = novi;
53     }

55     /* Vraca se indikator uspesnog dodavanja */
    return 0;
57 }

59 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                    Zahtev * zahtev)
61 {
    Cvor *pomocni = NULL;
63
    /* Ako je red prazan */
65     if (*adresa_pocetka == NULL)
        return 0;
67
    /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
69     zahtev koji je na pocetku reda. */
    if (zahtev != NULL)
71         *zahtev = (*adresa_pocetka)->nalog;

73     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac
        na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor
75     u redu. */
    pomocni = *adresa_pocetka;
77     *adresa_pocetka = (*adresa_pocetka)->sledeci;
    free(pomocni);
79
    /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,
81     potrebno je azurirati i vrednost pokazivaca na adresi
        adresa_kraja na NULL */
83     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");
    }

```

Datoteka 4.13: *main.c*

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

```
2 #include <stdlib.h>
3 #include <string.h>
4 #include "red.h"
5
6 #define VREME_ZA_PAUZU 5
7
8 int main(int argc, char **argv)
9 {
10     /* Red je prazan. */
11     Cvor *pocetak = NULL, *kraj = NULL;
12     Zahtev nov_zahtev;
13     Zahtev *sledeci = NULL;
14     char odgovor[3];
15     int broj_usluzenih = 0;
16
17     /* Sluzbenik evidentira korisnicke zahteve unosnjem njihovog JMBG
18        broja i opisa potrebne usluge. */
19     printf("Sluzbenik evidentira korisnicke zahteve:\n");
20     while (1) {
21
22         /* Ucitava se JMBG */
23         printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
24         if (scanf("%s", nov_zahtev.jmbg) == EOF)
25             break;
26
27         /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
28            JMBG broja procitao i da bi fgets nakon toga procitala
29            ispravan red sa opisom zahteva */
30         getchar();
31
32         /* Ucitava se opis problema */
33         printf("\tOpis problema: ");
34         fgets(nov_zahtev.opis, MAX - 1, stdin);
35         /* Ako je poslednji karakter nov red, eliminiše se */
36         if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
37             nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
38
39         /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
40         if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
41             fprintf(stderr, "Neuspela alokacija za nov cvor\n");
42             oslobodi_red(&pocetak, &kraj);
43             exit(EXIT_FAILURE);
44         }
45     }
46
47     /* Otvaranje datoteke za dopisivanje izvestaja */
48     FILE *izlaz = fopen("izvestaj.txt", "a");
49     if (izlaz == NULL) {
50         fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
51         exit(EXIT_FAILURE);
52     }
53
54     /* Dokle god ima korisnika u redu, treba ih usluziti */
55     while (1) {
56         sledeci = pocetak_reda(pocetak);
57         /* Ako nema nikog vise u redu, prekida se petlja */
58         if (sledeci == NULL)
59             break;
60
61         printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
62         printf("i zahtevom: %s\n", sledeci->opis);
63
64         skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
65
66         broj_usluzenih++;
67
68         printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
69         scanf("%s", odgovor);
70
71         if (strcmp(odgovor, "Da") == 0)
72             dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
73         else
74             fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
```

```

        nov_zahtev.opis);
76
    if (broj_usluzenih == VREME_ZA_PAUZU) {
78        printf("\nDa li je kraj smene? [Da/Ne] ");
        scanf("%s", odgovor);

80        if (strcmp(odgovor, "Da") == 0)
82            break;
        else
84            broj_usluzenih = 0;
    }
86 }

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

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

104        if (broj_usluzenih == VREME_ZA_PAUZU) {
106            printf("\nDa li je kraj smene? [Da/Ne] ");
            scanf("%s", odgovor);
108            if (strcmp(odgovor, "Da") == 0)
                break;
            else
110                broj_usluzenih = 0;
        }
112    }

114 *****/
    /* Datoteka vise nije potrebna i treba je zatvoriti. */
116    fclose(izlaz);

118    /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
        neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
120        koju zauzima red sa neobradjenim zahtevima korisnika. */
    oslobodi_red(&pocetak, &kraj);

122    exit(EXIT_SUCCESS);
124 }

```

## Rešenje 4.11

Datoteka 4.14: *dvostruko\_povezana\_lista.h*

```

1 #ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
2 #define _DVOSTRUKO_POVEZANA_LISTA_H_

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

12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14     na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
    Cvor *napravi_cvor(int broj);
16

```

```

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

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

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

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

36  /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
38  int dodaj_iza(Cvor * tekuci, int broj);

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

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
52  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na cinjenicu da je lista koja se pretražuje
    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
58  /* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
    pokazivac glava se nalazi na adresi adresa_glave. */
    void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
60      tekuci);

62  /* 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. */
64  void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
66      broj);

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

74
76  /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
    liste, razdvojene zapetama i uokvirene zagradama. */
    void ispisi_listu(Cvor * glava);

78
80  /* Funkcija prikazuje cvorove liste pocevsi od kraja ka glavi liste. */
    void ispisi_listu_unazad(Cvor * kraj);

82  #endif

```

Datoteka 4.15: dvostruko\_povezana\_lista.c

```

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

```

```

4  Cvor *napravi_cvor(int broj)
6  {
8      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
        alokacije */
10     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
12     if (novi == NULL)
14         return NULL;

16     /* Inicijalizacija polja strukture */
18     novi->vrednost = broj;
20     novi->sledeci = NULL;

22     /* Vraca se adresa novog cvora */
24     return novi;
26 }

28 void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
30 {
32     Cvor *pomocni = NULL;

34     /* Ako lista nije prazna, onda treba osloboditi memoriju */
36     while (*adresa_glave != NULL) {
38         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
            osloboditi memoriju cvora koji predstavlja glavu liste */
40         pomocni = (*adresa_glave)->sledeci;
42         free(*adresa_glave);
44         /* Sledeci cvor je nova glava liste */
46         *adresa_glave = pomocni;
48     }

50     /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
        treba postaviti na NULL */
52     *adresa_kraja = NULL;
54 }

56 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
58                             adresa_kraja, int broj)
60 {
62     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
64     Cvor *novi = napravi_cvor(broj);
66     if (novi == NULL)
68         return 1;

70     /* Sledbenik novog cvora je glava stare liste */
72     novi->sledeci = *adresa_glave;

74     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
        treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
        krajnji */
76     if (*adresa_glave != NULL)
78         (*adresa_glave)->prethodni = novi;
80     else
82         *adresa_kraja = novi;

84     /* Novi cvor je nova glava liste */
86     *adresa_glave = novi;

88     /* Vraca se indikator uspesnog dodavanja */
90     return 0;
92 }

94 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
96                         int broj)
98 {
100     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
102     Cvor *novi = napravi_cvor(broj);
104     if (novi == NULL)
106         return 1;

108     /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
        ujedno i cela lista. Azurira se vrednost na koju pokazuju
        adresa_glave i adresa_kraja */
110     if (*adresa_glave == NULL) {

```

```

78     *adresa_glave = novi;
    *adresa_kraja = novi;
} else {
80     /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
        sledbenik poslednjeg cvora i azurira se samo pokazivac na kraj
82     liste */
    (*adresa_kraja)->sledeci = novi;
84     novi->prethodni = (*adresa_kraja);
    *adresa_kraja = novi;
86 }

88 /* Vraca se indikator uspesnog dodavanja */
    return 0;
90 }

92 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
94     /* U praznoj listi nema takvog mesta i vraca se NULL */
    if (glava == NULL)
96         return NULL;

98     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
        pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
100     vrednost vecu ili jednaku od vrednosti novog cvora.

102     Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
        provera da li se doslo do poslednjeg cvora liste pre nego sto se
104     proverí vrednost u sledecem cvoru jer u slucaju poslednjeg,
        sledeci ne postoji pa ni njegova vrednost. */
106     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
        glava = glava->sledeci;

108     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
        poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
        vrednost vecu od broj */
110     return glava;
112 }

114 int dodaj_iza(Cvor * tekuci, int broj)
{
116     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
118     if (novi == NULL)
        return 1;

120     novi->sledeci = tekuci->sledeci;
    novi->prethodni = tekuci;

122     /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
        a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
        na ispravne adrese */
124     if (tekuci->sledeci != NULL)
        tekuci->sledeci->prethodni = novi;
126     tekuci->sledeci = novi;

128     /* Vraca se indikator uspesnog dodavanja */
    return 0;
130 }

132 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
    broj)
134 {
136     /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste */
    if (*adresa_glave == NULL) {
140         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
        Cvor *novi = napravi_cvor(broj);
142         if (novi == NULL)
            return 1;

144         /* Azuriraju se vrednosti pocetka i kraja liste */
        *adresa_glave = novi;
146         *adresa_kraja = novi;
148     }

```



```

150     /* Vraca se indikator uspesnog dodavanja */
151     return 0;
152 }

154 /* Ukoliko je vrednost glave liste veca ili jednaka od nove
155     vrednosti onda novi cvor treba staviti na pocetak liste */
156 if ((*adresa_glave)->vrednost >= broj) {
157     return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
158 }

160 /* Pronazi se cvor iza koga treba uvezati novi cvor */
161 Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
162 /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
163 if (dodaj_iza(pomocni, broj) == 1)
164     return 1;
165 /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
166     novi cvor poslednji u listi. */
167 if (pomocni == *adresa_kraja)
168     *adresa_kraja = novi;

169 return 0;
170 }

172 Cvor *pretrazi_listu(Cvor * glava, int broj)
173 {
174     /* Obilaze se cvorovi liste */
175     for (; glava != NULL; glava = glava->sledeci)
176         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
177             se obustavlja */
178             if (glava->vrednost == broj)
179                 return glava;

182     /* Nema trazenog broja u listi i vraca se NULL */
183     return NULL;
184 }

186 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
187 {
188     /* Obilaze se cvorovi liste */
189     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
190     for (; glava != NULL && glava->vrednost <= broj;
191           glava = glava->sledeci)
192         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
193             se obustavlja */
194             if (glava->vrednost == broj)
195                 return glava;

196     /* Nema trazenog broja u listi i bice vrateno NULL */
197     return NULL;
198 }

200 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
201     lako realizovati jer on sadrzi pokazivace na svog sledbenika i
202     prethodnika u listi. U funkciji se bise cvor zadat argumentom
203     tekuci */
204 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
205     tekuci)
206 {
207     /* Ako je tekuci NULL pokazivac, nema sta da se brise */
208     if (tekuci == NULL)
209         return;

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     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
217         treba da bude prethodnik tekuceg cvora */
218     if (tekuci->sledeci != NULL)
219         tekuci->sledeci->prethodni = tekuci->prethodni;

222     /* Ako je glava cvor koji se brise, nova glava liste ce biti

```

```

    sledbenik stare glave */
224 if (tekuci == *adresa_glave)
    *adresa_glave = tekuci->sledeci;
226
/* Ako je cvor koji se brise poslednji u listi, azurira se i
228 pokazivac na kraj liste */
if (tekuci == *adresa_kraja)
230 *adresa_kraja = tekuci->prethodni;

232 /* Oslobadja se dinamicki alocirani prostor za cvor tekuci */
free(tekuci);
234 }

236 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj)
{
238     Cvor *tekuci = *adresa_glave;

240     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broj, takvi
        cvorovi se brisu iz liste. */
242     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
244 }

246 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
        adresa_kraja, int broj)
248 {
    Cvor *tekuci = *adresa_glave;
250
    /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
252     takvi cvorovi se brisu iz liste. */
    while ((tekuci =
254         pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
256 }

258 void ispisi_listu(Cvor * glava)
{
260     putchar('[');
    /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
262     pocetka prema kraju liste */
    for (; glava != NULL; glava = glava->sledeci) {
264         printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
266             printf(", ");
    }

268     printf("]\n");
270 }

272 void ispisi_listu_unazad(Cvor * kraj)
{
274     putchar('[');
    /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
276     prema pocetku liste */
    for (; kraj != NULL; kraj = kraj->prethodni) {
278         printf("%d", kraj->vrednost);
        if (kraj->prethodni != NULL)
280             printf(", ");
    }
282     printf("]\n");
}

```

Datoteka 4.16: *main\_a.c*

```

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

5 /* 1) Glavni program */
int main()
7 {
    /* Lista je prazna na pocetku */

```

```

9  /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
11     da bi operacije poput dodavanja na kraj liste i ispisivanja
12     liste unazad bile efikasne poput dodavanja na pocetak liste i
13     ispisivanja liste od pocetnog do poslednjeg cvora. */
14
15  Cvor *glava = NULL;
16  Cvor *kraj = NULL;
17  Cvor *trazeni = NULL;
18  int broj;
19
20  /* Testira se funkcija za dodavanja novog broja na pocetak liste */
21  printf("Unesite brojeve: (za kraj CTRL+D)\n");
22  while (scanf("%d", &broj) > 0) {
23      /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
24         memorije za novi cvor. Memoriju alociranu za cvorove liste
25         treba osloboditi pre napustanja programa */
26      if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
27          fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
28          oslobodi_listu(&glava, &kraj);
29          exit(EXIT_FAILURE);
30      }
31      printf("\tLista: ");
32      ispisi_listu(glava);
33
34      /* Testira se funkcija za pretragu liste */
35      printf("\nUnesite broj koji se trazi u listi: ");
36      scanf("%d", &broj);
37
38      /* Pokazivac trazeni dobija vrednost rezultata pretrage */
39      trazeni = pretrazi_listu(glava, broj);
40      if (trazeni == NULL)
41          printf("Broj %d se ne nalazi u listi!\n", broj);
42      else
43          printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
44
45      /* Ispisuje se lista unazad */
46      printf("\nLista ispisana u nazad: ");
47      ispisi_listu_unazad(kraj);
48
49      /* Oslobadja se memorija zauzeta za cvorove liste */
50      oslobodi_listu(&glava, &kraj);
51
52      exit(EXIT_SUCCESS);
53  }

```

Datoteka 4.17: *main\_b.c*

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"
4
5  /* 2) Glavni program */
6  int main()
7  {
8      /* Lista je prazna na pocetku. */
9      Cvor *glava = NULL;
10     Cvor *kraj = NULL;
11     int broj;
12
13     /* Testira se funkcija za dodavanja novog broja na kraj liste */
14     printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
15     while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17            memorije za novi cvor. Memoriju alociranu za cvorove liste
18            treba osloboditi pre napustanja programa */
19         if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
20             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21             oslobodi_listu(&glava, &kraj);
22             exit(EXIT_FAILURE);
23         }
24         printf("\tLista: ");
25         ispisi_listu(glava);

```

```

26     }

28     /* Testira se funkcija za brisanje elemenata iz liste */
29     printf("\nUnesite broj koji se briše iz liste: ");
30     scanf("%d", &broj);

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

36     printf("Lista nakon brisanja: ");
37     ispisi_listu(glava);

38

40     /* Ispisuje se lista unazad */
41     printf("\nLista ispisana u nazad: ");
42     ispisi_listu_unazad(kraj);

44     /* Oslobadja se memorija zauzeta za cvorove liste */
45     oslobodi_listu(&glava, &kraj);

46     exit(EXIT_SUCCESS);
47 }

```

Datoteka 4.18: *main.c.c*

```

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

5  /* 3) Glavni program */
6  int main()
7  {
8      /* Lista je prazna na pocetku */
9      Cvor *glava = NULL;
10     Cvor *kraj = NULL;
11     Cvor *trazeni = NULL;
12     int broj;

13

14     /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
15        bude uredjena neopadajuće */
16     printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
17     while (scanf("%d", &broj) > 0) {
18         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
19            memorije za novi cvor. Memoriju alociranu za cvorove liste
20            treba osloboditi pre napustanja programa */
21         if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
22             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23             oslobodi_listu(&glava, &kraj);
24             exit(EXIT_FAILURE);
25         }
26         printf("\tLista: ");
27         ispisi_listu(glava);
28     }

29

30     /* Testira se funkcija za pretragu liste */
31     printf("\nUnesite broj koji se trazi u listi: ");
32     scanf("%d", &broj);

33

34     /* Pokazivac trazeni dobija vrednost rezultata pretrage */
35     trazeni = pretrazi_listu(glava, broj);
36     if (trazeni == NULL)
37         printf("Broj %d se ne nalazi u listi!\n", broj);
38     else
39         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

41     /* Testira se funkcija za brisanje elemenata iz liste */
42     printf("\nUnesite broj koji se briše iz liste: ");
43     scanf("%d", &broj);

44     /* Brisu se cvorovi iz liste čije polje vrednost je jednako broju
45        procitanom sa ulaza */
46     obrisi_cvor_sortirane_liste(&glava, &kraj, broj);

```

```

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

51
   /* Ispisuje se lista unazad */
53 printf("\nLista ispisana u nazad: ");
   ispisi_listu_unazad(kraj);

55
   /* Oslobadja se memorija zauzeta za cvorove liste */
57 oslobodi_listu(&glava, &kraj);

59 exit(EXIT_SUCCESS);
   }

```

### Rešenje 4.14

Datoteka 4.19: *stabla.h*

```

1  #ifndef _STABLA_H_
2  #define _STABLA_H_ 1

4  /* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
6  typedef struct cvor {
   int broj;
8   struct cvor *levo;
   struct cvor *desno;
10 } Cvor;

12 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
   funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
   do greske */
18 int dodaj_u_stablo(Cvor ** adresa_korena, int broj);

20
22 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
   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
36 /* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
   postablo - Koren - Desno podstablo ) */
   void ispisi_stablo_infiksno(Cvor * koren);

38
40 /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
   Levo podstablo - Desno podstablo ) */
   void ispisi_stablo_prefiksno(Cvor * koren);

42
44 /* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
   podstablo - Desno postablo - Koren ) */
   void ispisi_stablo_postfiksno(Cvor * koren);

46
48 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
   void oslobodi_stablo(Cvor ** adresa_korena);

50 #endif

```

Datoteka 4.20: *stabla.c*

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

Cvor *napravi_cvor(int broj)
{
    /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;

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

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

int dodaj_u_stablo(Cvor ** adresa_korena, int broj)
{
    /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {

        /* Kreira se novi cvor */
        Cvor *novi_cvor = napravi_cvor(broj);

        /* Proverava se uspesnost kreiranja */
        if (novi_cvor == NULL) {

            /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
               */
            return 1;
        }
        /* Inace ... */

        /* Novi cvor se proglašava korenom stabla */
        *adresa_korena = novi_cvor;

        /* I vraca se indikator uspesnosti kreiranja */
        return 0;
    }

    /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati
       broj */

    /* Ako je zadata vrednost manja od vrednosti korena */
    if (broj < (*adresa_korena)->broj)

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

Cvor *pretrazi_stablo(Cvor * koren, int broj)
{
    /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
    if (koren == NULL)
        return NULL;

    /* Ako je trazena vrednost sadrzana u korenu */
    if (koren->broj == broj) {

        /* Prekidamo pretragu */
        return koren;
    }
}
```

```

74  /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
75  if (broj < koren->broj)
76
77      /* Pretraga se nastavlja u levom podstablu */
78      return pretrazi_stablo(koren->levo, broj);
79
80  else
81
82      /* U suprotnom, pretraga se nastavlja u desnom podstablu */
83      return pretrazi_stablo(koren->desno, broj);
84  }
85
86  Cvor *pronadji_najmanji(Cvor * koren)
87  {
88
89      /* Ako je stablo prazno, prekida se pretraga */
90      if (koren == NULL)
91          return NULL;
92
93      /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
94         levo od njega */
95
96      /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
97         najmanju vrednost */
98      if (koren->levo == NULL)
99          return koren;
100
101      /* Inace, pretragu treba nastaviti u levom podstablu */
102      return pronadji_najmanji(koren->levo);
103  }
104
105  Cvor *pronadji_najveci(Cvor * koren)
106  {
107
108      /* Ako je stablo prazno, prekida se pretraga */
109      if (koren == NULL)
110          return NULL;
111
112      /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
113         desno od njega */
114
115      /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
116         najveću vrednost */
117      if (koren->desno == NULL)
118          return koren;
119
120      /* Inace, pretragu treba nastaviti u desnom podstablu */
121      return pronadji_najveci(koren->desno);
122  }
123
124  void obrisi_element(Cvor ** adresa_korena, int broj)
125  {
126      Cvor *pomocni_cvor = NULL;
127
128      /* Ako je stablo prazno, brisanje nije primenljivo */
129      if (*adresa_korena == NULL)
130          return;
131
132      /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
133         stabla, ona se eventualno nalazi u levom podstablu, pa treba
134         rekursivno primeniti postupak na levo podstablo. Koren ovako
135         modifikovanog stabla je nepromenjen. */
136      if (broj < (*adresa_korena)->broj) {
137          obrisi_element(&(*adresa_korena)->levo, broj);
138          return;
139      }
140
141      /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
142         stabla, ona se eventualno nalazi u desnom podstablu pa treba
143         rekursivno primeniti postupak na desno podstablo. Koren ovako
144         modifikovanog stabla je nepromenjen. */
145      if ((*adresa_korena)->broj < broj) {
146          obrisi_element(&(*adresa_korena)->desno, broj);
147          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 vrsi 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 vrsi 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
    /* Slucaj 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 moze pronaci 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 brise). 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 najvise jednog
190    potomka, jasno je da ce njegovo brisanje biti obavljeno na jedan
        od jednostavnijih nacina koji su gore opisani. */
192    pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
    (*adresa_korena)->broj = pomocni_cvor->broj;
194    pomocni_cvor->broj = broj;
    obrisi_element(&(*adresa_korena)->desno, broj);
196 }

198 void ispisi_stablo_infiksno(Cvor * koren)
    {
199     /* Ako stablo nije prazno */
    if (koren != NULL) {
200
201         /* Prvo se ispisuju svi cvorovi levo od korena */
202         ispisi_stablo_infiksno(koren->levo);
203
204         /* Zatim se ispisuje vrednost u korenu */
205         printf("%d ", koren->broj);
206
207         /* Na kraju se ispisuju cvorovi desno od korena */
208         ispisi_stablo_infiksno(koren->desno);
209     }
210 }

212

214 void ispisi_stablo_prefiksno(Cvor * koren)
    {
215     /* Ako stablo nije prazno */
    if (koren != NULL) {
216
217         /* Prvo se ispisuje vrednost u korenu */
218

```



```

220     printf("%d ", koren->broj);

222     /* Zatim se ispisuju svi cvorovi levo od korena */
    ispisi_stablo_prefiksno(koren->levo);

224     /* Na kraju se ispisuju svi cvorovi desno od korena */
    ispisi_stablo_prefiksno(koren->desno);
226 }
228 }

230 void ispisi_stablo_postfiksno(Cvor * koren)
{
232     /* Ako stablo nije prazno */
234     if (koren != NULL) {

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

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

240         /* Na kraju se ispisuje vrednost u korenu */
        printf("%d ", koren->broj);
242     }
244 }
}

246 void oslobodi_stablo(Cvor ** adresa_korena)
{
248     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
250     if (*adresa_korena == NULL)
        return;

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

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

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

260     /* Proglasava se stablo praznim */
    *adresa_korena = NULL;
262 }
264 }

```

Datoteka 4.21: *main.c*

```

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

5  int main()
{
7     Cvor *koren;
   int n;
9     Cvor *trazeni_cvor;

11     /* Proglasava se stablo praznim */
    koren = NULL;

13     /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
       dodavanja */
15     printf("Unesite brojeve (CTRL+D za kraj unosa): ");
    while (scanf("%d", &n) != EOF) {
17         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 ispisi: ");
27  ispisi_stablo_infiksno(koren);
    printf("\nPrefiksni ispisi: ");
29  ispisi_stablo_prefiksno(koren);
    printf("\nPostfiksni ispisi: ");
31  ispisi_stablo_postfiksno(koren);

33  /* Demonstrira se rad funkcije za pretragu */
    printf("\nTrazi se broj: ");
35  scanf("%d", &n);
    trazen_i_cvor = pretrazi_stablo(koren, n);
37  if (trazen_i_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

```

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

6  #define MAX 50

8  /* Struktura kojom se opisuje cvor stabla: sadrži rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
    char *rec;
12    int broj;
    struct cvor *levo;
14    struct cvor *desno;
} Cvor;

16

18 /* Funkcija koja kreira novi cvor stabla */
Cvor *napravi_cvor(char *rec)
{
20     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
22     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
    if (novi_cvor == NULL)
24         return NULL;

26     /* Alocira se memorija za zadatu rec: potrebno je rezervirati
       memoriju za svaki karakter reci ukljucujuci i terminirajucu nulu
       */
28     novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
    if (novi_cvor->rec == NULL) {
30         free(novi_cvor);
32         return NULL;
    }

34

36     /* Inicijalizuju se polja u novom cvoru */
    strcpy(novi_cvor->rec, rec);
    novi_cvor->broj = 1;

```

```

38     novi_cvor->levo = NULL;
    novi_cvor->desno = NULL;
40
    /* Vraca se adresa novog cvora */
42     return novi_cvor;
    }
44
    /* Funkcija koja dodaje novu rec u stablo - ukoliko je dodavanje
    uspesno povratna vrednost je 0, u suprotnom povratna vrednost je 1
    */
46
48     int dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
    {
49         /* Ako je stablo prazno */
        if (*adresa_korena == NULL) {
50             /* Kreira se cvor koji sadrzi zadatu rec */
            Cvor *novi_cvor = napravi_cvor(rec);
52             /* Proverava se uspesnost kreiranja novog cvora */
            if (novi_cvor == NULL) {
53                 /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
                */
54                 return 1;
            }
55             /* Inace... */
            /* Novi cvor se proglašava korenom stabla */
56             *adresa_korena = novi_cvor;
57
            /* I vraca se indikator uspesnog dodavanja */
58             return 0;
59         }
60
        /* Ako stablo nije prazno, trazi odgovarajuca pozicija za novu rec */
61
62         /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
        levo podstablo */
63         if (strcmp(rec, (*adresa_korena)->rec) < 0)
64             return dodaj_u_stablo(&(*adresa_korena)->levo, rec);
65
66         else
67             /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
            desno podstablo */
68             if (strcmp(rec, (*adresa_korena)->rec) > 0)
69                 return dodaj_u_stablo(&(*adresa_korena)->desno, rec);
70
71         else
72             /* Ako je rec jednaka reci u korenu, uvecava se njen broj
            pojavljivanja */
73             (*adresa_korena)->brojac++;
74     }
75
76     /* Funkcija koja oslobadja memoriju zauzetu stablom */
77     void oslobodi_stablo(Cvor ** adresa_korena)
78     {
79         /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
80         if (*adresa_korena == NULL)
81             return;
82
83         /* Inace ... */
84         /* Oslobadja se memorija zauzeta levim podstablom */
85         oslobodi_stablo(&(*adresa_korena)->levo);
86
87         /* Oslobadja se memorija zauzeta desnim podstablom */
88         oslobodi_stablo(&(*adresa_korena)->desno);
89
90         /* Oslobadja se memorija zauzeta korenom */
91         free((*adresa_korena)->rec);
92         free(*adresa_korena);
93
94         /* Stablo se proglašava praznim */
95         *adresa_korena = NULL;
96     }
97
98     /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
    sa najvećim brojem pojavljivanja) */
99

```

```

Cvor *nadj_i_najfrekventniju_rec(Cvor * koren)
112 {
    Cvor *max, *max_levo, *max_desno;

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

118     /* Pronalazi se najfrekventnija rec u levom podstablu */
120     max_levo = nadj_i_najfrekventniju_rec(koren->levo);

122     /* Pronalazi se najfrekventnija rec u desnom podstablu */
124     max_desno = nadj_i_najfrekventniju_rec(koren->desno);

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

132     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
134     return max;
}

136 /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
138     pracen brojem pojavljivanja */
void prikazi_stablo(Cvor * koren)
140 {
    /* Ako je stablo prazno, završava se sa ispisom */
142     if (koren == NULL)
        return;

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

148     /* Zatim rec iz korena */
150     printf("%s: %d\n", koren->rec, koren->brojac);

152     /* I nastavlja se sa ispisom reci iz desnog podstabla */
    prikazi_stablo(koren->desno);
154 }

156 /* Funkcija učitava sledeću rec iz zadate datoteke f i upisuje je u
    niz rec. Maksimalna dužina reci je određena argumentom max.
158     Funkcija vraća EOF ako u datoteci nema više reci ili 0 u
    suprotnom. Rec je niz malih ili velikih slova. */
160 int procitaj_rec(FILE * f, char rec[], int max)
{
    /* Karakter koji se cita */
162     int c;

164     /* Indeks pozicije na koju se smesta procitani karakter */
166     int i = 0;

168     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
        nije stiglo do kraja datoteke... */
170     while (i < max - 1 && (c = fgetc(f)) != EOF) {
        /* Proverava se da li je procitani karakter slovo */
172         if (isalpha(c))
            /* Ako jeste, smesta se u niz - pritom se vrsi konverzija u
174             mala slova jer program treba da bude neosetljiv na razliku
            izmedju malih i velikih slova */
176             rec[i++] = tolower(c);

178         else
            /* Ako nije, proverava se da li je procitano barem jedno slovo
180             nove reci */
            /* Ako jeste, prekida se sa citanjem */
182             if (i > 0)
                break;
    }
}

```

```

184     /* U suprotnom se ide na sledecu iteraciju */
185 }
186
187 /* Dodaje se na rec terminirajuca nula */
188 rec[i] = '\0';
189
190 /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
191 return i > 0 ? 0 : EOF;
192 }
193
194 int main(int argc, char **argv)
195 {
196     Cvor *koren = NULL, *max;
197     FILE *f;
198     char rec[MAX];
199
200     /* Provera da li je navedeno ime datoteke prilikom pokretanja
201        programa */
202     if (argc < 2) {
203         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
204         exit(EXIT_FAILURE);
205     }
206
207     /* Priprema datoteke za citanje */
208     if ((f = fopen(argv[1], "r")) == NULL) {
209         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
210             argv[1]);
211         exit(EXIT_FAILURE);
212     }
213
214     /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
215        pretrage uz proveru uspesnosti dodavanja */
216     while (procitaj_rec(f, rec, MAX) != EOF) {
217         if (dodaj_u_stablo(&koren, rec) == 1) {
218             fprintf(stderr, "Neuspelo dodavanje reci %s\n", rec);
219             oslobodi_stablo(&koren);
220             exit(EXIT_FAILURE);
221         }
222     }
223
224     /* Posto je citanjem reci zavrшено, zatvara se datoteka */
225     fclose(f);
226
227     /* Prikazuju se sve reci iz teksta i brojevi njihovih
228        pojavljivanja. */
229     prikazi_stablo(koren);
230
231     /* Pronalazi se najfrekventnija rec */
232     max = najdi_najfrekventniju_rec(koren);
233
234     /* Ako takve reci nema... */
235     if (max == NULL)
236
237         /* Ispisuje se odgovarajuće obaveštenje */
238         printf("U tekstu nema reci!\n");
239
240     else
241         /* Inace, ispisuje se broj pojavljivanja reci */
242         printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
243             max->rec, max->brojac);
244
245     /* Oslobadja se dinamički alociran prostor za stablo */
246     oslobodi_stablo(&koren);
247
248     exit(EXIT_SUCCESS);
249 }
250

```

## Rešenje 4.16

```

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

```

```

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

#define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
#define MAX_IME_I_PREZIME 100

/* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime, broj
   telefona i redom pokazivace na levo i desno podstablo */
typedef struct cvor {
    char ime_i_prezime[MAX_IME_I_PREZIME];
    char telefon[MAX_CIFARA];
    struct cvor *levo;
    struct cvor *desno;
} Cvor;

/* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
    /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
    Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
    if (novi_cvor == NULL)
        return NULL;

    /* Inicijalizuju se polja novog cvora */
    strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
    strcpy(novi_cvor->telefon, telefon);
    novi_cvor->levo = NULL;
    novi_cvor->desno = NULL;

    /* Vraca se adresa novog cvora */
    return novi_cvor;
}

/* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
   ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
   povratna vrednost je 1 */
int
dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
               char *telefon)
{
    /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
        /* Kreira se novi cvor */
        Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
        /* Proverava se uspesnost kreiranja novog cvora */
        if (novi_cvor == NULL) {
            /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
               */
            return 1;
        }
        /* Inace... */
        /* Novi cvor se proglašava korenom stabla */
        *adresa_korena = novi_cvor;

        /* I vraca se indikator uspesnog dodavanja */
        return 0;
    }

    /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
       unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
       treba da bude pretraživacko po ovom polju */

    /* Ako je zadato ime i prezime leksikografski manje od imena i
       prezimena sadržanog u korenu, podaci se dodaju u levo podstablo */
    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
        < 0)
        return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
                              telefon);
    else
        /* Ako je zadato ime i prezime leksikografski veće od imena i
           prezimena sadržanog u korenu, podaci se dodaju u desno podstablo */
        return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
                              telefon);
}

```

```

76     prezimena sadrzanog u korenu, podaci se dodaju u desno
       podstablo */
78     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
       return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
80                             telefon);
}
82
/* Funkcija koja oslobadja memoriju zauzetu stablom */
84 void oslobodi_stablo(Cvor ** adresa_korena)
{
86     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
     if (*adresa_korena == NULL)
88         return;

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

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

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

100    /* Stablo se proglašava praznim */
    *adresa_korena = NULL;
102 }

104 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
/* Napomena: ova funkcija nije trazena u zadatku ali se može
106    koristiti za proveru da li je stablo lepo kreirano ili ne */
void prikazi_stablo(Cvor * koren)
108 {
    /* Ako je stablo prazno, završava se sa ispisom */
110    if (koren == NULL)
        return;

112    /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
       podstabla */
114    prikazi_stablo(koren->levo);

116    /* Zatim se ispisuju podaci iz korena */
118    printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

120    /* I nastavlja se sa ispisom podataka iz desnog podstabla */
    prikazi_stablo(koren->desno);
122 }

124 /* Funkcija učitava sledeći kontakt iz zadate datoteke i upisuje ime
       i prezime i broj telefona u odgovarajuće nizove. Maksimalna dužina
126    imena i prezimena određena je konstantom MAX_IME_PREZIME, a
       maksimalna dužina broja telefona konstantom MAX_CIFARA. Funkcija
128    vraća EOF ako nema više kontakata ili 0 u suprotnom. */
int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
130 {
    /* Karakter koji se čita */
132    int c;

134    /* Indeks pozicije na koju se smesta procitani karakter */
    int i = 0;

136    /* Linije datoteke koje se obrađuju su formata Ime Prezime
       BrojTelefona */
138

140    /* Preskaku se eventualne praznine sa početka linije datoteke */
    while ((c = fgetc(f)) != EOF && isspace(c));

142

144    /* Prvo procitano slovo upisuje se u ime i prezime */
    if (!feof(f))
        ime_i_prezime[i++] = c;

146

148    /* Naznaka kraja citanja imena i prezimena će biti pojava prve
       cifre tako da se citanje vrši sve dok se ne nađe na cifru.

```

```

150     Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
        procitanog karaktera i da se slucajno ne dodje do kraja datoteke
151     */
152     while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
        if (!isdigit(c))
154         ime_i_prezime[i++] = c;

156         else if (i > 0)
            break;
158     }

160     /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
        blanko karaktera */
162     ime_i_prezime[--i] = '\0';

164     /* I pocinje se sa citanjem broja telefona */
        i = 0;

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

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

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

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

184     /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
        */
186     Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
188     {
        /* Ako je imenik prazan, 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();
223 if ((f = fopen(ime_datoteke, "r")) == NULL) {
224     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
                ime_datoteke);
225     exit(EXIT_FAILURE);
226 }
227
228 /* Citaju se podaci iz datoteke i smestanju u binarno stablo
229 pretrage uz proveru uspesnosti dodavanja */
230 while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
231     if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
232         fprintf(stderr, "Neuspelo dodavanje podataka za osobu %s\n",
                    ime_i_prezime);
233         oslobodi_stablo(&koren);
234         exit(EXIT_FAILURE);
235     }
236
237 /* Zatvara se datoteka */
238 fclose(f);
239
240 /* Omogucava se pretraga imenika */
241 while (1) {
242     /* Ucitavaja se ime i prezime */
243     printf("Unesite ime i prezime: ");
244     i = 0;
245     while ((c = getchar()) != '\n')
246         ime_i_prezime[i++] = c;
247     ime_i_prezime[i] = '\0';
248
249     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
250 funkcionalnost */
251     if (strcmp(ime_i_prezime, "KRAJ") == 0)
252         break;
253
254     /* Inace se ispisuje rezultat pretrage */
255     trazeni = pretrazi_imenik(koren, ime_i_prezime);
256     if (trazeni == NULL)
257         printf("Broj nije u imeniku!\n");
258     else
259         printf("Broj je: %s \n", trazeni->telefon);
260 }
261
262 /* Oslobadja se memorija zauzeta imenikom */
263 oslobodi_stablo(&koren);
264
265 exit(EXIT_SUCCESS);
266 }

```

### Rešenje 4.17

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX 51
6
7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8     studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9     jezika i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX];
12     char prezime[MAX];
13     double uspeh;
14     double matematika;
15     double jezik;
16     struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
18 } Cvor;
19
20 /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
                    double matematika, double jezik)

```

```

23 {
24     /* Alocira se memorija za novi cvor i proverava se uspesnost
25        alokacije. */
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;
29
30     /* Inicijalizuju se polja strukture */
31     strcpy(novi->ime, ime);
32     strcpy(novi->prezime, prezime);
33     novi->uspeh = uspeh;
34     novi->matematika = matematika;
35     novi->jezik = jezik;
36     novi->levo = NULL;
37     novi->desno = NULL;
38
39     /* Vraca se adresa kreiranog cvora */
40     return novi;
41 }
42
43 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo -
44    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
45    povratna vrednost je 1 */
46 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
47                   double uspeh, double matematika, double jezik)
48 {
49     /* Ako je stablo prazno */
50     if (*koren == NULL) {
51         /* Kreira se novi cvor */
52         Cvor *novi_cvor =
53             napravi_cvor(ime, prezime, uspeh, matematika, jezik);
54         /* Proverava se uspesnost kreiranja novog cvora */
55         if (novi_cvor == NULL) {
56             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
57                */
58             return 1;
59         }
60         /* Inace... */
61         /* Novi cvor se proglašava korenom stabla */
62         *koren = novi_cvor;
63
64         /* I vraca se indikator uspesnog dodavanja */
65         return 0;
66     }
67
68     /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
69        sortirano po ukupnom broju poena */
70     if (uspeh + matematika + jezik >
71         (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
72         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
73                               matematika, jezik);
74     else
75         return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
76                               matematika, jezik);
77 }
78
79 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
80 void oslobodi_stablo(Cvor ** koren)
81 {
82     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
83     if (*koren == NULL)
84         return;
85
86     /* Inace ... */
87     /* Oslobadja se memorija zauzeta levim podstablom */
88     oslobodi_stablo(&(*koren)->levo);
89
90     /* Oslobadja se memorija zauzeta desnim podstablom */
91     oslobodi_stablo(&(*koren)->desno);
92
93     /* Oslobadja se memorija zauzeta korenom */
94     free(*koren);
95 }

```

```

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
105 položili prijemni, a ako je vrednost argumenta različita od nule,
    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);

    /* Stampaju se informacije iz korenog cvora */
115     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)
    {
151         FILE *in;
        Cvor *koren;
153         char ime[MAX], prezime[MAX];
        double uspeh, matematika, jezik;
155
        /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
157         in = fopen("prijemni.txt", "r");
        if (in == NULL) {
159             fprintf(stderr,
                "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
161             exit(EXIT_FAILURE);
        }
163
        /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
            dodavanja */
165         koren = NULL;
167         while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
            &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
```

```

vrednost je 1 */
43 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                     int dan, int mesec)
44 {
45     /* Ako je stablo prazno */
46     if (*koren == NULL) {
47
48         /* Kreira se novi cvor */
49         Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
50         /* Proverava se uspesnost kreiranja novog cvora */
51         if (novi_cvor == NULL) {
52             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
53              */
54             return 1;
55         }
56         /* Inace... */
57         /* Novi cvor se proglašava korenom stabla */
58         *koren = novi_cvor;
59
60         /* I vraca se indikator uspesnog dodavanja */
61         return 0;
62     }
63
64     /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
65      meseca */
66     if (mesec < (*koren)->mesec)
67         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
68     else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
69         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
70     else
71         return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec);
72 }
73
74 /* Funkcija vrsi pretragu stabla i vraca cvor sa trazanim datumom */
75 Cvor *pretrazi(Cvor * koren, int dan, int mesec)
76 {
77     /* Stablo je prazno, obustavlja se pretraga */
78     if (koren == NULL)
79         return NULL;
80
81     /* Ako je trazeni datum u korenu */
82     if (koren->dan == dan && koren->mesec == mesec)
83         return koren;
84
85     /* Ako je mesec trazenog datuma manji od meseca sadržanog u korenu
86      ili ako su meseci isti ali je dan trazenog datuma manji od
87      aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
88      svakako proverava da li leva grana postoji - ako ne postoji
89      treba vratiti prvi sledeci, a to je bas vrednost uocenog korena */
90     if (mesec < koren->mesec
91         || (mesec == koren->mesec && dan < koren->dan)) {
92         if (koren->levo == NULL)
93             return koren;
94         else
95             return pretrazi(koren->levo, dan, mesec);
96     }
97
98     /* Inace se nastavlja pretraga u desnom delu */
99     return pretrazi(koren->desno, dan, mesec);
100 }
101
102 /* Funkcija koja pronalazi najmanji datum u stablu */
103 Cvor *pronadji_najmanji_datum(Cvor * koren)
104 {
105     /* Stablo je prazno, obustavlja se pretraga */
106     if (koren == NULL)
107         return NULL;
108
109     /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
110      sadrzi najmanji datum */
111     if (koren->levo == NULL)
112         return koren;
113     else

```

```

115     /* Inace, trazimo manji datum u levom podstablu */
116     return pronadji_najmanji_datum(koren->levo);
117 }

119 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM. */
120 void datum_u_nisku(int dan, int mesec, char datum[])
121 {
122     if (dan < 10) {
123         datum[0] = '0';
124         datum[1] = dan + '0';
125     } else {
126         datum[0] = dan / 10 + '0';
127         datum[1] = dan % 10 + '0';
128     }
129     datum[2] = '.';

131     if (mesec < 10) {
132         datum[3] = '0';
133         datum[4] = mesec + '0';
134     } else {
135         datum[3] = mesec / 10 + '0';
136         datum[4] = mesec % 10 + '0';
137     }
138     datum[5] = '.';
139     datum[6] = '\\0';
140 }

141 /* Funkcija koja oslobadja memoriju zauzetu stablom */
142 void oslobodi_stablo(Cvor ** adresa_korena)
143 {
144     /* Stablo je prazno */
145     if (*adresa_korena == NULL)
146         return;

147     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
148     if ((*adresa_korena)->levo)
149         oslobodi_stablo(&(*adresa_korena)->levo);

151     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
152     if ((*adresa_korena)->desno)
153         oslobodi_stablo(&(*adresa_korena)->desno);

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

157     /* Proglasava se stablo praznim */
158     *adresa_korena = NULL;
159 }

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     }
187 }

```

```

189  /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja */
koren = NULL;
191  while (fscanf
        (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
193      if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
          fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
195                  prezime);
          oslobodi_stablo(&koren);
197          exit(EXIT_FAILURE);
      }

199  /* Datoteka se zatvara */
201  fclose(in);

203  /* Omogucuje se pretraga podataka */
while (1) {
205
    /* Ucitava se novi datum */
207    printf("Unesite datum: ");
    if (scanf("%d.%d.", &dan, &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: */
    /* 1. drvo je prazno */
217    if (slavljenik == NULL && koren == NULL) {
        printf("Nema podataka o ovom ni o sledecem rođjendanu.\n");
219        continue;
    }
221    /* 2. posle datuma koji je unesen, nema podataka u stablu - u
        ovom slucaju se pretraga vrši pocevši od naredne godine i
        ispisuje se najmanji datum */
223    if (slavljenik == NULL) {
        slavljenik = pronadji_najmanji_datum(koren);
225        datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
        printf("Slavljenik: %s %s %s\n", slavljenik->ime,
227                slavljenik->prezime, datum);
        continue;
229    }
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 rođjendanu */
    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

```

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

4 /* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

```

```

6
8  /* Funkcija koja proverava da li su dva stabla koja sadrže cele
   brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
10 int identitet(Cvor * koren1, Cvor * koren2)
12 {
14     /* Ako su oba stabla prazna, jednaka su */
16     if (koren1 == NULL && koren2 == NULL)
18         return 1;

20     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
22     if (koren1 == NULL || koren2 == NULL)
24         return 0;

26     /* Ako su oba stabla neprazna i u korenu se nalaze različite
       vrednosti, može se zaključiti da se razlikuju */
28     if (koren1->broj != koren2->broj)
30         return 0;

32     /* Inače, proverava se da li vazi i jednakost levih i desnih
       podstabala */
34     return (identitet(koren1->levo, koren2->levo)
36             && identitet(koren1->desno, koren2->desno));
38 }

40 int main()
42 {
44     int broj;
46     Cvor *koren1, *koren2;

48     /* Učitavaju se elementi prvog stabla */
50     koren1 = NULL;
52     printf("Prvo stablo: ");
54     scanf("%d", &broj);
56     while (broj != 0) {
58         if (dodaj_u_stablo(&koren1, broj) == 1) {
60             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
62             oslobodi_stablo(&koren1);
64             return 0;
66         }
68         scanf("%d", &broj);
70     }

72     /* Učitavaju se elementi drugog stabla */
74     koren2 = NULL;
76     printf("Drugo stablo: ");
78     scanf("%d", &broj);
80     while (broj != 0) {
82         if (dodaj_u_stablo(&koren2, broj) == 1) {
84             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
86             oslobodi_stablo(&koren2);
88             return 0;
90         }
92         scanf("%d", &broj);
94     }

96     /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
       njen rezultat. */
98     if (identitet(koren1, koren2))
100         printf("Stabla jesu identična.\n");
102     else
104         printf("Stabla nisu identična.\n");

106     /* Oslobadja se memorija zauzeta stablima */
108     oslobodi_stablo(&koren1);
110     oslobodi_stablo(&koren2);

112     return 0;
114 }

```

## Rešenje 4.20



```

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

4  /* Uključuje se biblioteka za rad sa stablima */
   #include "stabla.h"

6

8  /* Funkcija kreira novo stablo identično stablu koje je dato korenom.
   Povratna vrednost funkcije je 0 ukoliko je kopiranje uspesno,
   odnosno 1 ukoliko je doslo do greske */
10 int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
   {
12     /* Izlaz iz rekurzije */
13     if (koren == NULL) {
14         *duplikat = NULL;
15         return 0;
16     }

18     /* Duplira se koren stabla i postavlja da bude koren novog stabla */
19     *duplikat = napravi_cvor(koren->broj);
20     if (*duplikat == NULL) {
21         return 1;
22     }

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

32     /* Ako je uspesno duplirano i levo i desno podstablo */
33     if (kopija_levo == 0 && kopija_desno == 0)
34         /* Uspesno je duplirano i celo stablo */
35         return 0;
36     /* Inace, prijavljuje se da je doslo do greske */
37     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 */
44 int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
   {
46     /* Ako drugo stablo nije prazno */
47     if (koren2 != NULL) {
48         /* 1. Dodaje se njegov koren u prvo stablo */
49         if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
50             return 1;
51         }

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);

58         /* Ako je unija podstabala uspesno kreirana */
59         if (unija_levo == 0 && unija_desno == 0)
60             /* Uspesno je kreirana i unija stabala */
61             return 0;

62         /* U suprotnom se prijavljuje da je doslo do greske */
63         return 1;
64     }

66     /* Ako je drugo stablo prazno, nista se ne preduzima */
67     return 0;
68 }

70

72 /* Funkcija izracunava presek dva skupa predstavljena stablima -
   rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
   Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka

```

```

74     uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
76 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
78     if (*adresa_korena1 == NULL)
        return 0;
80
    /* 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 rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
    Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike
104 uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
106 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
108     if (*adresa_korena1 == NULL)
        return 0;
110
    /* Inace... */
112     /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
        tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
114     elementi koji postoje i u drugom stablu */
    int razlika_levo =
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 */
126     return 0;
    }
128
    /* Inece, prijavljuje se da je doslo do greske */
130     return 1;
}

132 int main()
134 {
    Cvor *skup1;
136     Cvor *skup2;
    Cvor *pomocni_skup = NULL;
138     int n;

140     /* Ucitavaju se elementi prvog skupa */
    skup1 = NULL;
142     printf("Prvi skup: ");
    while (scanf("%d", &n) != EOF) {
144         if (dodaj_u_stablo(&skup1, n) == 1) {
            fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
146             oslobodi_stablo(&skup1);

```

```

    return 0;
}
}

/* Ucitavaju se elementi drugog skupa */
skup2 = NULL;
printf("Drugi skup: ");
while (scanf("%d", &n) != EOF) {
    if (dodaj_u_stablo(&skup2, n) == 1) {
        fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
        oslobodi_stablo(&skup2);
        return 0;
    }
}

/* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
   bi se isti mogao iskoristiti i za preostale operacije */
if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
    oslobodi_stablo(&skup1);
    oslobodi_stablo(&pomocni_skup);
    return 0;
}
if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
    oslobodi_stablo(&pomocni_skup);
    oslobodi_stablo(&skup2);
    return 0;
}
printf("Unija: ");
ispisi_stablo_infiksno(pomocni_skup);
putchar('\n');

/* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
   operacije */
oslobodi_stablo(&pomocni_skup);

/* Kreira se presek skupova: prvo se napravi kopija prvog skupa
   kako bi se isti mogao iskoristiti i za preostale operacije */
if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
    oslobodi_stablo(&skup1);
    oslobodi_stablo(&pomocni_skup);
    return 0;
}
if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
    oslobodi_stablo(&pomocni_skup);
    oslobodi_stablo(&skup2);
    return 0;
}
printf("Presek: ");
ispisi_stablo_infiksno(pomocni_skup);
putchar('\n');

/* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
   operacije */
oslobodi_stablo(&pomocni_skup);

/* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
   kako bi se isti mogao iskoristiti i za preostale operacije */
if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
    oslobodi_stablo(&skup1);
    oslobodi_stablo(&pomocni_skup);
    return 0;
}
if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
    oslobodi_stablo(&pomocni_skup);
    oslobodi_stablo(&skup2);
    return 0;
}
printf("Razlika: ");
ispisi_stablo_infiksno(pomocni_skup);
putchar('\n');

/* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
   operacije */

```

```
220   oslobodi_stablo(&pomocni_skup);

222   /* Oslobadja se memorija zauzeta polaznim skupovima */
   oslobodi_stablo(&skup1);
224   oslobodi_stablo(&skup2);

226   return 0;
}
```

### Rešenje 4.21

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

3

   /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

7  #define MAX 50

9  /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
   cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11  su smestene u niz. */
   int kreiraj_niz(Cvor * koren, int a[])
13  {
       int r, s;

15       /* Stablo je prazno - u niz je smesteno 0 elemenata */
       if (koren == NULL)
17         return 0;

19       /* Dodaju se u niz elementi iz levog podstabla */
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
37   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
   desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
39   kreiran i sortiran, a 1 ukoliko je doslo do greske.

41   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)
   {
       int i;
       Cvor *koren;

47       /* Kreira se stablo smestanjem elemenata iz niza u stablo */
       koren = NULL;
       for (i = 0; i < n; i++) {
51         if (dodaj_u_stablo(&koren, a[i]) == 1) {
             oslobodi_stablo(&koren);
53             return 1;
55         }
       }
57       /* Infiksni obilaskom stabla elementi iz stabla se prepisuju u niz
       a */
59       kreiraj_niz(koren, a);

61   }
```

```

63  /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima */
    oslobodi_stablo(&koren);

65  /* Vraca se indikator uspesnog sortiranja */
    return 0;
67 }

69 int main()
{
71     int a[MAX];
    int n, i;

73     /* Ucitavaju se dimenzija i elementi niza */
75     printf("n: ");
    scanf("%d", &n);
77     if (n < 0 || n > MAX) {
        printf("Greska: pogresna dimenzija niza!\n");
79         return 0;
    }

81     printf("a: ");
83     for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

85     /* Poziva se funkcija za sortiranje */
87     if (sortiraj(a, n) == 0) {
        /* Ako je niz uspesno sortirano, ispisuje se rezultujuci niz */
89         for (i = 0; i < n; i++)
            printf("%d ", a[i]);
91         printf("\n");
    } else {
93         /* Inace, obavestava se korisnik da je doslo do greske */
        printf("Greska: problem prilikom sortiranja niza!\n");
95     }

97     return 0;
}

```

### Rešenje 4.22

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
    #include "stabla.h"

6  /* a) Funkcija koja izracunava broj cvorova stabla */
8  int broj_cvorova(Cvor * koren)
{
10     /* Ako je stablo prazno, broj cvorova je nula */
    if (koren == NULL)
12         return 0;

14     /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
        levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
        zato sto treba racunati i koren */
16     return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
18 }

20 /* b) Funkcija koja izracunava broj listova stabla */
    int broj_listova(Cvor * koren)
22 {
    /* Ako je stablo prazno, broj listova je nula */
24     if (koren == NULL)
        return 0;

26     /* Proverava se da li je tekuci cvor list */
28     if (koren->levo == NULL && koren->desno == NULL)
        /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
        uvecati broj listova za 1 */
30         return 1;
32 }

```

```
34  /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
35  */
36  return broj_listova(koren->levo) + broj_listova(koren->desno);
37  }

38  /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
39  void pozitivni_listovi(Cvor * koren)
40  {
41      /* Slučaj kada je stablo prazno */
42      if (koren == NULL)
43          return;
44
45      /* Ako je cvor list i sadrži pozitivnu vrednost */
46      if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
47          /* Stampa se */
48          printf("%d ", koren->broj);
49
50      /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
51      pozitivni_listovi(koren->levo);
52      pozitivni_listovi(koren->desno);
53  }

54
55  /* d) Funkcija koja izracunava zbir cvorova stabla */
56  int zbir_svih_cvorova(Cvor * koren)
57  {
58      /* Ako je stablo prazno, zbir cvorova je 0 */
59      if (koren == NULL)
60          return 0;
61
62      /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
63      elemenata u podstablima */
64      return koren->broj + zbir_svih_cvorova(koren->levo) +
65             zbir_svih_cvorova(koren->desno);
66  }

67
68  /* e) Funkcija koja izracunava najveći element stabla */
69  Cvor *najveci_element(Cvor * koren)
70  {
71      /* Ako je stablo prazno, obustavlja se pretraga */
72      if (koren == NULL)
73          return NULL;
74
75      /* Zbog prirode pretrazivackog stabla, vrednosti veće od korena se
76      nalaze u desnom podstablu */
77
78      /* Ako desnog podstabla nema */
79      if (koren->desno == NULL)
80          /* Najveća vrednost je koren */
81          return koren;
82
83      /* Inace, najveća vrednost se traži desno */
84      return najveci_element(koren->desno);
85  }

86
87  /* f) Funkcija koja izracunava dubinu stabla */
88  int dubina_stabla(Cvor * koren)
89  {
90      /* Dubina praznog stabla je 0 */
91      if (koren == NULL)
92          return 0;
93
94      /* Izracunava se dubina levog podstabla */
95      int dubina_levo = dubina_stabla(koren->levo);
96
97      /* Izracunava se dubina desnog podstabla */
98      int dubina_desno = dubina_stabla(koren->desno);
99
100     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
101     jer se racuna i koren */
102     return dubina_levo >
103            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 trazenog
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 trazenog nivoa, vraca se 1 - to ce kasnije zbog
117        rekurzivnih poziva uvecati broj cvorova za 1 */
118     if (i == 0)
119         return 1;
120
121     /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu */
122     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
123         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
124 }
125
126 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
127 void ispis_nivo(Cvor * koren, int i)
128 {
129     /* Ideja je slicna ideji iz prethodne funkcije */
130
131     /* Nema vise cvorova, nema spustanja kroz stablo */
132     if (koren == NULL)
133         return;
134
135     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
136     if (i == 0) {
137         printf("%d ", koren->broj);
138         return;
139     }
140     /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
141        desnom podstablu */
142     ispis_nivo(koren->levo, i - 1);
143     ispis_nivo(koren->desno, i - 1);
144 }
145
146 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
147        stabla */
148 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
149 {
150     /* Ako je stablo prazno, obustavlja se pretraga */
151     if (koren == NULL)
152         return NULL;
153
154     /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
155     if (i == 0)
156         return koren;
157
158     /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
159     Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
160
161     /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
162     Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);
163
164     /* Trazi se i vraca maksimum izracunatih vrednosti */
165     if (a == NULL && b == NULL)
166         return NULL;
167     if (a == NULL)
168         return b;
169     if (b == NULL)
170         return a;
171     return a->broj > b->broj ? a : b;
172 }
173
174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
175 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
177     /* Ako je stablo prazno, zbir je nula */
178     if (koren == NULL)

```

```

    return 0;
180
/* Ako se stiglo do trazenog nivoa, vraca se vrednost */
182 if (i == 0)
    return koren->broj;
184
/* Inace, spustanje se nastavlja za jedan nivo nize i traze 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

/* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
192 manje ili jednake od date vrednosti x */
int zbir_manjih_od_x(Cvor * koren, int x)
194 {
    /* Ako je stablo prazno, zbir je nula */
196 if (koren == NULL)
    return 0;
198

    /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
200 prirode pretrazivackog stabla treba obici i levo i desno
202 podstablo */
    if (koren->broj <= x)
204 return koren->broj + zbir_manjih_od_x(koren->levo, x) +
        zbir_manjih_od_x(koren->desno, x);
206

    /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
208 medju njima jedino moze biti onih koje zadovoljavaju uslov */
    return zbir_manjih_od_x(koren->levo, x);
210 }

int main(int argc, char **argv)
212 {
    /* Analiza argumenata komandne linije */
214 if (argc != 3) {
        fprintf(stderr,
216             "Greska! Program se poziva sa: ./a.out nivo broj_za_pretragu\n");
        return 1;
218     }

    int i = atoi(argv[1]);
220 int x = atoi(argv[2]);
222

    /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
224 */
    Cvor *koren = NULL;
    int broj;
226 while (scanf("%d", &broj) != EOF) {
        if (dodaj_u_stablo(&koren, broj) == 1) {
228             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
                oslobodi_stablo(&koren);
230             return 0;
        }
232     }
}
234

/* ispisuju se rezultati rada funkcija */
236 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,
250     broj_cvorova_na_itom_nivou(koren, i));
printf("Elementi na %d. nivou: ", i);

```



```

252   ispis_nivo(koren, i);
      printf("\n");
254   if (najveci_element_na_itom_nivou(koren, i) == NULL)
      printf("Nema elemenata na %d. nivou!\n", i);
256   else
      printf("Maksimalni element na %d. nivou: %d\n", i,
258           najveci_element_na_itom_nivou(koren, i)->broj);

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

8  /* Funkcija koja izracunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
   {
10     /* Dubina praznog stabla je 0 */
       if (koren == NULL)
12         return 0;

14     /* Izracunava se dubina levog podstabla */
       int dubina_levo = dubina_stabla(koren->levo);

16     /* Izracunava se dubina desnog podstabla */
       int dubina_desno = dubina_stabla(koren->desno);

18     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
       jer se racuna i koren */
20     return dubina_levo >
           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
22 }

24

26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
   void ispisi_nivo(Cvor * koren, int i)
   {
28     /* Ideja je slicna ideji iz prethodne funkcije */
       /* Nema vise cvorova, nema spustanja niz stablo */
       if (koren == NULL)
30         return;

32     /* Ako se stiglo do trazene nivoa - ispisuje se vrednost */
       if (i == 0) {
34         printf("%d ", koren->broj);
           return;
36     }

38     /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
       podstablu */
40     ispisi_nivo(koren->levo, i - 1);
       ispisi_nivo(koren->desno, i - 1);
42 }

44

46 /* Funkcija koja ispisuje stablo po nivoima */
   void ispisi_stablo_po_nivoima(Cvor * koren)
   {
48     int i;

50     /* Prvo se izracunava dubina stabla */
       int dubina;

```

```

52 dubina = dubina_stabla(koren);

54 /* Ispisuje se nivo po nivo stabla */
55 for (i = 0; i < dubina; i++) {
56     printf("%d. nivo: ", i);
57     ispisi_nivo(koren, i);
58     printf("\n");
59 }
60 }

62 int main(int argc, char **argv)
63 {
64     Cvor *koren;
65     int broj;

66     /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
67     uspesnosti dodavanja */
68     koren = NULL;
69     while (scanf("%d", &broj) != EOF) {
70         if (dodaj_u_stablo(&koren, broj) == 1) {
71             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
72             oslobodi_stablo(&koren);
73             return 0;
74         }
75     }

76     /* Ispisuje se stablo po nivoima */
77     ispisi_stablo_po_nivoima(koren);

78     /* Oslobadja se memorija zauzeta stablom */
79     oslobodi_stablo(&koren);

80     return 0;
81 }

```

## Rešenje 4.25

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija koja izracunava dubinu stabla */
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     }

```

```

37  /* Izracunava se dubina levog podstabla korena */
    dubina_levo = dubina_stabla(koren->levo);

39

    /* Izracunava se dubina desnog podstabla korena */
41  dubina_desno = dubina_stabla(koren->desno);

    /* Ako je uslov za AVL stablo ispunjen */
43  if (abs(dubina_desno - dubina_levo) <= 1) {
45      /* Racuna se broj AVL cvorova u levom i desnom podstablu i
        uvecava za jedan iz razloga sto koren ispunjava uslov */
47      return 1 + avl(koren->levo) + avl(koren->desno);
    } else {
49      /* Inace, racuna se samo broj AVL cvorova u podstablama */
        return avl(koren->levo) + avl(koren->desno);
51    }
}

53
int main(int argc, char **argv)
55 {
    Cvor *koren;
57    int broj;

    /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
        uspesnosti dodavanja */
59    koren = NULL;
    while (scanf("%d", &broj) != EOF) {
61        if (dodaj_u_stablo(&koren, broj) == 1) {
63            fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
65            oslobodi_stablo(&koren);
            return 0;
67        }
    }

69    /* Racuna se i ispisuje broj AVL cvorova */
    printf("%d\n", avl(koren));
71

    /* Oslobadja se memorija zauzeta stablom */
73    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

    /* 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
        ce funkcija vratiti maksimalne vrednosti iz uocenog podstabala ili
12     vrednost -1 ukoliko se zakljuci da stablo nije hip. */
int heap(Cvor * koren)
14 {
16     int max_levo, max_desno;

    /* Prazno sablo je hip - kao rezultat se vraca 0 kao najmanji
        pozitivan broj */
18     if (koren == NULL) {
20         return 0;
    }

22     /* Ukoliko je stablo list... */
    if (koren->levo == NULL && koren->desno == NULL) {
24         /* Vraca se njegova vrednost */
        return koren->broj;
26     }
}

```

```
28  /* Inace... */
30  /* Proverava se svojstvo za levo podstablo */
   max_levo = heap(koren->levo);
32
   /* Proverava se svojstvo za desno podstablo */
34  max_desno = heap(koren->desno);
36
   /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
      ni celo stablo */
38  if (max_levo == -1 || max_desno == -1) {
   return -1;
40 }
42
   /* U suprotnom proverava se da li svojstvo vazi za uoceni cvor */
   if (koren->broj > max_levo && koren->broj > max_desno) {
44     /* Ako vazi, vraca se vrednost korena */
   return koren->broj;
46 }
48
   /* U suprotnom zakljucuje se da stablo nije hip */
   return -1;
50 }

52 int main(int argc, char **argv)
   {
54     Cvor *koren;
   int hip_indikator;
56
   /* Kreira se stablo prema zadatoj slici */
58   koren = NULL;
   koren = napravi_cvor(100);
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");
   } else {
76     printf("Zadato stablo je hip!\n");
   }
78
   /* Oslobadja se memorija zauzeta stablom */
80   oslobodi_stablo(&koren);
82
   return 0;
   }
```

## Glava 5

# Ispitni rokovi

### 5.1 Programiranje 2, praktični deo ispita, jun 2015.

**Zadatak 5.1** Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu za grešku ispisati vrednost `-1` i prekinuti izvršavanje programa.

<p><i>Test 1</i></p> <pre>Poziv: ./a.out ulaz.txt ULAZ.TXT 5 Programiranje Matematika 12345 dInAmiCnArEc Ispit IZLAZ: Ispit Matematika Programiranje</pre>	<p><i>Test 2</i></p> <pre>Poziv: ./a.out ulaz.txt ULAZ.TXT 2 maksimalano poena IZLAZ:</pre>
<p><i>Test 3</i></p> <pre>Poziv: ./a.out ulaz.txt DATOTEKA ULAZ.TXT NE POSTOJI IZLAZ: -1</pre>	<p><i>Test 4</i></p> <pre>Poziv: ./a.out IZLAZ: -1</pre>

[Rešenje 5.1]

**Zadatak 5.2** Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na  $n$ -tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj  $n$ , a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

```

Test 1
|| ULAZ:
|| 2 8 10 3 6 14 13 7 4 0
|| IZLAZ:
|| 20

```

```

Test 2
|| ULAZ:
|| 0 50 14 5 2 4 56 8 52 7 1 0
|| IZLAZ:
|| 50

```

[Rešenje 5.2]

**Zadatak 5.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost -1 na standardni izlaz za greške.

```

Test 1
|| ULAZ:
|| 4 5
|| 1 2 3 4 5
|| -1 2 -3 4 -5
|| -5 -4 -3 -2 1
|| -1 0 0 0 0
|| IZLAZ:
|| 0

```

```

Test 2
|| ULAZ:
|| 2 3
|| 0 0 -5
|| 1 2 -4
|| IZLAZ:
|| 2

```

```

Test 3
|| ULAZ:
|| -2
|| IZLAZ:
|| -1

```

[Rešenje 5.3]

## 5.2 Programiranje 2, praktični deo ispita, jul 2015.

**Zadatak 5.4** Napisati program koji kao prvi argument komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili NULL ako podniska nije pronađena.

```

Test 1
|| Poziv: ./a.out ulaz.txt test
||
|| ULAZ.TXT
|| Ovo je test primer.
|| U njemu se rec test javlja
|| vise puta. testtesttest
||
|| IZLAZ:
|| 5

```

```

Test 2
|| Poziv: ./a.out
||
|| IZLAZ:
|| -1

```

```

Test 3
|| Poziv: ./a.out ulaz.txt foo
||
|| DATOTEKA ULAZ.TXT NE POSTOJI
||
|| IZLAZ:
|| -1

```

```

Test 4
|| Poziv: ./a.out ulaz.txt .
||
|| DATOTEKA ULAZ.TXT JE PRAZNA
||
|| IZLAZ:
|| 0

```

[Rešenje 5.4]

**Zadatak 5.5** Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitava trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P =$



<p><i>Test 4</i></p> <pre> ULAZ: 4 0 IZLAZ: 4 </pre>	<p><i>Test 5</i></p> <pre> ULAZ: 0 5 IZLAZ: 0 </pre>
--	--

[Rešenje 5.7]

**Zadatak 5.8** Napisati funkciju `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

```

<p><i>Test 2</i></p> <pre> ULAZ: 12 IZLAZ: 12.00 </pre>	<p><i>Test 3</i></p> <pre> ULAZ: prazna lista IZLAZ: </pre>	<p><i>Test 4</i></p> <pre> ULAZ: 13.3 15.8 IZLAZ: 13.30 14.55 </pre>
---	---	--

[Rešenje 5.8]

**Zadatak 5.9** Sa standardnog ulaza se učitava dimenzija  $n$  kvadratne celobrojne matrice  $A$  ( $n > 0$ ), a zatim i elementi matrice  $A$ . Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje je suma brojeva u svakom redu i svakoj koloni jednaka). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati `-`. Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati `-1` na standardni izlaz za grešku. NAPOMENA: *Rešenje koristi biblioteku za rad sa matricama iz zadatka 2.19.*

<p><i>Test 1</i></p> <pre> ULAZ: 4 1 2 3 4 2 1 4 3 3 4 2 1 4 3 1 2 IZLAZ: 10 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 3 1 1 1 1 1 1 1 1 1 IZLAZ: 3 </pre>	<p><i>Test 3</i></p> <pre> ULAZ: 2 1 1 2 2 IZLAZ: - </pre>
<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: -1 </pre>

[Rešenje 5.9]

## 5.4 Rešenja

Rešenje 5.1



```

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX 50

/* Funkcija vrši dinamičku alokaciju memorije potrebne n linija tj.
   n niski od kojih nijedna nije duža od MAX karaktera. */
char **alociranje_memorije(int n)
{
    char **linije = NULL;
    int i, j;
    /* Alocira se prostor za niz vrsti matrice */
    linije = (char **) malloc(n * sizeof(char *));
    /* U slučaju neuspješnog otvaranja ispisuje se -1 na stderr i
       program završava. */
    if (linije == NULL)
        return NULL;
    /* Alocira se prostor za svaku vrstu matrice */
    for (i = 0; i < n; i++) {
        linije[i] = malloc(MAX * sizeof(char));
        /* Ako alokacija nije prošla uspješno, oslobadja se svi
           prethodno alocirani resursi, i povratna vrednost je NULL */
        if (linije[i] == NULL) {
            for (j = 0; j < i; j++) {
                free(linije[j]);
            }
            free(linije);
            return NULL;
        }
    }
    return linije;
}

char **oslobadjanje_memorije(char **linije, int n)
{
    int i;
    /* Oslobadja se prostor rezervisan za svaku vrstu */
    for (i = 0; i < n; i++) {
        free(linije[i]);
    }
    /* Oslobadja se memorija za niz pokazivaca na vrste */
    free(linije);

    /* Matrica postaje prazna, tj. nealocirana */
    return NULL;
}

int main(int argc, char *argv[])
{
    FILE *ulaz;
    char **linije;
    int i, j, n;

    /* Proverava argumenata komandne linije. */
    if (argc != 2) {
        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
    }

    /* Otvaranje datoteke cije ime je navedeno kao argument komandne
       linije neposredno nakon imena programa koji se poziva. U slučaju
       neuspješnog otvaranja ispisuje se -1 na stderr i program završava. */
    ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
    }

    /* Ucitavanje broja linija. */
    fscanf(ulaz, "%d", &n);

    /* Alociranje memorije na osnovu ucitanog broja linija. */
    linije = alociranje_memorije(n);

```

```

74  /* U slucaju neuspesne alokacije ispisuje se -1 na stderr i program
    zavrsava. */
76  if (ulaz == NULL) {
    fprintf(stderr, "-1\n");
78  exit(EXIT_FAILURE);
    }

80

    /* Ucitavanje svih n linija iz datoteke. */
82  for (i = 0; i < n; i++) {
    fscanf(ulaz, "%s", linije[i]);
84  }

    /* Ispisivanje u odgovarajucem poretku ucitane linije koje
    zadovoljavaju kriterijum. */
86  for (i = n - 1; i >= 0; i--) {
    if (isupper(linije[i][0])) {
88      printf("%s\n", linije[i]);
90    }
    }
92  }
    /* Oslobadjanje memorije koja je dinamicki alocirana. */
94  linije = oslobadjanje_memorije(linije, n);

96  /* Zatvaranje datoteku. */
    fclose(ulaz);
98  exit(EXIT_SUCCESS);
}

```

## Rešenje 5.2

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Datoteka 5.1: *main.c*

```

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

4

int sumirajN(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;
    /* 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 sumirajN(koren->levo, n - 1) + sumirajN(koren->desno, n - 1);
}

20

int main()
22  {
    Cvor *koren = NULL;
24  int n;
    int nivo;

26

    /* Ucitava se vrednost nivoa */
28  scanf("%d", &nivo);
    while (1) {
30      scanf("%d", &n);
        /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
32      if (n == 0)
        break;
        /* Ako nije, dodaje se procitani broj u stablo. */
34      if (dodaj_u_stablo(&koren, n) == 1) {
36          fprintf(stderr, "-1\n", n);
            oslobodi_stablo(&koren);
            exit(EXIT_FAILURE);
38      }
    }
}

```

```

40 }

42 /* Ispisuje se rezultat rada trazene funkcije */
printf("%d\n", sumirajN(koren, nivo));

44 /* Oslobadja se memorija */
oslobodi_stablo(&koren);

46

48 exit(EXIT_SUCCESS);
}

```

### Rešenje 5.3

```

#include <stdio.h>
2 #include <stdlib.h>
#define MAX 50

4 /* Funkcija ucitava elemente matrice sa standardnog ulaza */
6 void ucitaj_matricu(int m[][MAX], int v, int k)
{
8     int i, j;
    for (i = 0; i < v; i++) {
10         for (j = 0; j < k; j++) {
            scanf("%d", &m[i][j]);
12         }
    }
14 }

16 /* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
    koja ima v vrsta */
18 int broj_negativnih_u_koloni(int m[][MAX], int v, int k)
{
20     int broj_negativnih = 0;
    int i;
22     for (i = 0; i < v; i++) {
        if (m[i][k] < 0)
24             broj_negativnih++;
    }
26     return broj_negativnih;
}

28 int max_indeks(int m[][MAX], int v, int k)
{
30     int i, j;
    int broj_negativnih;
32     /* Inicijalizujemo na nulu indeks kolone sa maksimalnim brojem
        negativnih (max_indeks_kolone), kao i maksimalni broj negativnih
34         elemenata u nekoj koloni (max_broj_negativnih) */
    int max_indeks_kolone = 0;
    int max_broj_negativnih = 0;
36

38     for (j = 0; j < k; j++) {
        /* Racunamo broj negativnih u j-oj koloni */
        broj_negativnih = broj_negativnih_u_koloni(m, v, j);
42         /* Ukoliko broj negativnih u j-toj koloni veci od trenutnog
            maksimalnog, azuriramo trenutni maksimalni broj negativnih
            kao i indeks kolone sa maksimalnim brojem negativnih */
44         if (max_broj_negativnih < broj_negativnih) {
            max_indeks_kolone = j;
            max_broj_negativnih = broj_negativnih;
46         }
    }
48     return max_indeks_kolone;
50 }

52 int main()
{
54     int m[MAX][MAX];
    int v, k;
56

58     /* Ucitavanje dimenzije matrice */
    scanf("%d", &v);

```

```

60     if (v < 0 || v > MAX) {
61         fprintf(stderr, "-1\n");
62         exit(EXIT_FAILURE);
63     }
64     scanf("%d", &k);
65     if (k < 0 || k > MAX) {
66         fprintf(stderr, "-1\n");
67         exit(EXIT_FAILURE);
68     }
69     /* Ucitavanje elemenata matrice */
70     ucitaj_matricu(m, v, k);
71
72     /* Pronalazenje kolone koja sadrzi najveći broj negativnih
73        elemenata i ispisivanje trazenog rezultata */
74     printf("%d\n", max_indeks(m, v, k));
75     exit(EXIT_SUCCESS);
76 }

```

## Rešenje 5.4

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #define MAX 128

6 int main(int argc, char **argv)
{
8     FILE *f;
    int brojac = 0;
10    char linija[MAX], *p;

12    /* Provera da li je broj argumenata komandne linije 3 */
    if (argc != 3) {
14        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
16    }
    /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18       linije */
    if ((f = fopen(argv[1], "r")) == NULL) {
20        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
22    }
    /* Ucitavanje iz otvorene datoteke - liniju po liniju */
24    while (fgets(linija, MAX, f) != NULL) {
        p = linija;
26        while (1) {
            p = strstr(p, argv[2]);

28            /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
30            if (p == NULL)
                break;
            /* 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    }
    /* Zatvaranje datoteke */
40    fclose(f);
    /* Ispisivanje vrednosti brojac */
42    printf("%d\n", brojac);

44    exit(EXIT_SUCCESS);
}

```

## Rešenje 5.5

```

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

```

```

3 #include <math.h>

5 /* Struktura trougao */
6 typedef struct _trougao {
7     double xa, ya, xb, yb, xc, yc;
8 } trougao;

9 /* Funkcija racuna duzinu duzi */
11 double duzina(double x1, double y1, double x2, double y2)
12 {
13     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
14 }

15 /* Funkcija racuna povrsinu trougla koristeći Heronov obrazac */
17 double povrsina(trougao t)
18 {
19     /* Racunanje duzina stranica trougla */
20     double a = duzina(t.xb, t.yb, t.xc, t.yc);
21     double b = duzina(t.xa, t.ya, t.xc, t.yc);
22     double c = duzina(t.xa, t.ya, t.xb, t.yb);
23     /* Poluobim */
24     double s = (a + b + c) / 2;
25     /* Primena Heronovog obrasca */
26     return sqrt(s * (s - a) * (s - b) * (s - c));
27 }

29 /* Funkcija poredi dva trougla: ukoliko je povrsina trougla koji je
30 prvi argument funkcije manja od povrsine trougla koji je drugi
31 element funkcije funkcija vraca 1, ukoliko je veca -1, a ukoliko
32 su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
33 napisana tako da se moze proslediti funkciji qsort da se niz
34 trouglova sortira po povrsini opadajuće. */
35 int poredi(const void *a, const void *b)
36 {
37     trougao x = *(trougao *) a;
38     trougao y = *(trougao *) b;
39     double xp = povrsina(x);
40     double yp = povrsina(y);
41     if (xp < yp)
42         return 1;
43     if (xp > yp)
44         return -1;
45     return 0;
46 }

47 int main()
48 {
49     FILE *f;
50     int n, i;
51     trougao *niz;

52     /* Otvaranje datoteke ciji je naziv trouglovi.txt */
53     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
54         fprintf(stderr, "-1\n");
55         exit(EXIT_FAILURE);
56     }

57     /* Ucitavanje podataka o broju trouglova iz datoteke */
58     if (fscanf(f, "%d", &n) != 1) {
59         fprintf(stderr, "-1\n");
60         exit(EXIT_FAILURE);
61     }

62     /* Dinamicka alokacija memorije za niz trouglova duzine n */
63     if ((niz = malloc(n * sizeof(trougao))) == NULL) {
64         fprintf(stderr, "-1\n");
65         exit(EXIT_FAILURE);
66     }

67     /* Ucitavanje podataka u niz iz otvorene datoteke */
68     for (i = 0; i < n; i++) {
69         if (fscanf(f, "%lf%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
70                 &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6) {

```

```

77     fprintf(stderr, "-1\n");
       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);
}

```

### Rešenje 5.6

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Datoteka 5.2: *main.c*

```

#include <stdio.h>
2  #include <stdlib.h>
   #include "stabla.h"
4
   /* Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
6   stablo. Funkcija vraća 1 u slučaju neuspješnog dodavanja elementa u
   stablo, a inače 0. */
8  int ucitaj_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
14     je i povratna vrednost iz funkcije ucitaj_stablo 1. */
       while (scanf("%d", &x) == 1)
16         if (dodaj_u_stablo(koren, x) == 1)
           return 1;
18     return 0;
   }
20

   /* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
22  int prebrojN(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) {
       if (koren->levo == NULL && koren->desno != NULL)
32         return 1;
       if (koren->levo != NULL && koren->desno == NULL)
34         return 1;
       return 0;
36     }
       /* 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
       levog podstabla */
40     return prebrojN(koren->levo, n - 1) + prebrojN(koren->desno, n - 1);
   }
42

   int main()
44  {

```

```

Cvor *koren;
46 int n;
scanf("%d", &n);

48
/* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
50 oslobodja se alocirana memorija i izlazi se iz programa. */
if (ucitaj_stablo(&koren) == 1) {
52     fprintf(stderr, "-1\n");
    oslobodi_stablo(&koren);
54     exit(EXIT_FAILURE);
}

56
/* Ispisivanje rezultata */
58 printf("%d\n", prebrojN(koren, n));

60
/* Oslobadjanje dinamički alociranog stabla */
oslobodi_stablo(&koren);

62
exit(EXIT_SUCCESS);
64 }

```

### Rešenje 5.7

```

#include <stdio.h>

2
/* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
4 zapis argumenta x rotira za n mesta udesno */
unsigned int Rotiraj(unsigned int x, unsigned int n)
6 {
    int i;
    unsigned int maska = 1;
    /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
10 izgledala: 000...00001111 */
    for (i = 1; i < n; i++)
12     maska = (maska << 1) | 1;
    /* Kada se x poremeri za n mesta udesno x >> n, poslednjih n bitova
14 binarne reprezentacije broja x ce "ispasti". Za rotaciju je
    potrebno da se tih n bitova postavi na pocetak broja. Kreirana
16 maska omogucava da se tih n bitova izdvoji sa (maska & x), a
    zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta ulevo tih
18 n bitova postavlja na pocetak. Primenom logicke disjunkcije
    dobija se rotirani broj. */
20     return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
}

22
int main()
24 {
    unsigned int x, n;

26
    /* Ucitavanje brojeva sa standardnog ulaza */
    scanf("%u%u", &x, &n);

28
    /* Ispisivanje rezultata */
    printf("%u\n", Rotiraj(x, n));
30
    return 0;
32
}

```

### Rešenje 5.8

Datoteka 5.3: *liste.h*

```

#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.
11                                     */
12     Cvor * napravi_cvor(double broj);
13
14                                     /* Funkcija oslobadja dinamicu
15                                     memoriju zauzetu za elemente
16                                     liste
17
18     ciji se pocetni cvor nalazi
19                                     na adresi adresa_glave. */
20 void oslobodi_listu(Cvor ** adresa_glave);
21
22                                     /* Funkcija pronalazi i vraca
23                                     pokazivac na poslednji element
24                                     liste,
25
26     ili NULL kao je lista
27                                     prazna */
28     Cvor * pronadji_poslednji(Cvor * glava);
29
30                                     /* Funkcija dodaje novi cvor na kraj
31                                     liste. Vraca 1 ukoliko je bilo
32                                     greske pri alokaciji memorije,
33                                     inace vraca 0. */
34 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);
35
36 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste. */
37 void ispisi_listu(Cvor * glava);
38 #endif

```

Datoteka 5.4: *liste.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "liste.h"
4
5 /* Pomocna funkcija koja kreira cvor. */
6 Cvor *napravi_cvor(double broj)
7 {
8     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9     if (novi == NULL)
10         return NULL;
11     /* inicijalizacija polja u novom cvoru */
12     novi->vrednost = broj;
13     novi->sledeci = NULL;
14     return novi;
15 }
16
17 /* Funkcija oslobadja dinamicu memoriju zauzetu za elemente liste
18     ciji se pocetni cvor nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave)
20 {
21     Cvor *pomocni = NULL;
22     while (*adresa_glave != NULL) {
23         pomocni = (*adresa_glave)->sledeci;
24         free(*adresa_glave);
25         *adresa_glave = pomocni;
26     }
27 }
28
29 }
30
31 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
32     ili NULL kao je lista prazna */
33 Cvor *pronadji_poslednji(Cvor * glava)
34 {
35     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
36         funkcija vraca NULL. */
37     if (glava == NULL)
38         return NULL;
39     while (glava->sledeci != NULL)
40

```



```

    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)
    {
49     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 }

64 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste. */
void ispisi_listu(Cvor * glava)
    {
66     for (; glava != NULL; glava = glava->sledeci)
        printf("%.21f ", glava->vrednost);
68     putchar('\n');
70 }

```

Datoteka 5.5: *main.c*

```

#include <stdio.h>
2  #include <stdlib.h>
#include "liste.h"
4

/* Funkcija umece novi cvor iza tekuceg u listi */
6 void dodaj_iza(Cvor * tekuci, Cvor * novi)
    {
8     /* Novi cvor se dodaje iza 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 {
    Cvor *tekuci;
    Cvor *novi;
20     double aritmeticka_sredina;
22     /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
    if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
24         return 1;
    /* Promenljiva tekuci se inicijalizuje da pokazuje na pocetni
26     cvor */
    tekuci = *adresa_glave;
28     /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
        vrednosti u susednim cvorovima i kreira cvor sa tom vrednoscu. U
30     slucaju neupele alokacije novog cvora, funkcija vraca 1. Inace,
        novi cvor se umece izmedju dva cvora za koje racunata
32     aritmeticka sredina */
    while (tekuci->sledeci != NULL) {
34         aritmeticka_sredina =
            ((tekuci)->vrednost + ((tekuci)->sledeci->vrednost) / 2;
36         novi = napravi_cvor(aritmeticka_sredina);
            if (novi == NULL)
38             return 1;
        /* Poziva se funkcija koja umece novi cvor iza tekuceg 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 }

50 int main()
51 {
52     Cvor *glava = NULL;
53     double broj;
54
55     /* Ucitavanje se vrši 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 5.9

```

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

/* Funkcija racuna zbir elemenata v-te vrste */
int zbir_vrste(int **M, int n, int v)
{
    int j, zbir = 0;
    for (j = 0; j < n; j++)
        zbir += M[v][j];
    return zbir;
}

/* Funkcija racuna zbir elemenata k-te kolone */
int zbir_kolone(int **M, int n, int k)
{
    int i, zbir = 0;
    for (i = 0; i < n; i++)
        zbir += M[i][k];
    return zbir;
}

/* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao

```

```

24     argument magican. Ukoliko jeste magican funkcija vraca 1, inace 0.
    Argument funkcije zbir ce sadrzati zbir elemenata neke vrste ili
26     kolone ukoliko je kvadrat magican. */
int magicni_kvadrat(int **M, int n, int *zbirmagicnog)
28 {
    int i, j;
30     int zbir = 0, zbir_pom;
    /* Promenljivu zbir inicijalizujemo na zbir 0-te vreste */
32     zbir = zbir_vrste(M, n, 0);

34     /* Racunaju se zbirovi 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 zbirovi u svim kolonama i ako neki razlikuje od
42     vrednosti promeljive zbir funkcija vraca 1 */
    for (j = 0; j < n; j++) {
44         zbir_pom = zbir_kolone(M, n, j);
        if (zbir_pom != zbir)
46             return 0;
    }
48     /* Inace su zbirovi svih vrsta i kolona jednaki, postavlja se
    vresnost u zbirmagicnog i funkcija vraca 1 */
50     *zbirmagicnog = zbir;
    return 1;
52 }

54 int main()
{
56     int n, i, j;
    int **matrica = NULL;
58     int zbir = 0, zbirmagicnog = 0;
    scanf("%d", &n);

60
    /* Provera da li je n strogo pozitivan */
62     if (n <= 0) {
        printf("-1\n");
64         exit(EXIT_FAILURE);
    }

66
    /* Dinamicka alokacija matrice dimenzije nxn */
68     matrica = alociraj_matricu(n);
    if (matrica == NULL) {
70         printf("-1\n");
        exit(EXIT_FAILURE);
72     }

74     /* Ucitavanje elemenata matrice sa standardnog ulaza */
    ucitaj_matricu(matrica, n);

76
    /* Ispisivanje rezultata na osnovu fukcije magicni_kvadrat */
78     if (magicni_kvadrat(matrica, n, &zbirmagicnog)) {
        printf("%d\n", zbirmagicnog);
80     } else
        printf("-\n");
82
    /* Oslobadjanje dinamicki alocirane memorije */
84     matrica = dealociraj_matricu(matrica, n);

86     exit(EXIT_SUCCESS);
}

```