

Univerzitet u Beogradu  
Matematički fakultet

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

## Zbirka programa

Beograd, 2015.



# 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.

Zahvaljujemo se recenzentima na ..., kao i studentima koji su svojim aktivnim učešćem u nastavi pomogli i doprineli u obličavanju ovog materijala.

*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 . . . . .	12
1.4	Rešenja . . . . .	19
<b>2</b>	<b>Pokazivači</b>	<b>61</b>
2.1	Pokazivačka aritmetika . . . . .	61
2.2	Višedimenzioni nizovi . . . . .	65
2.3	Dinamička alokacija memorije . . . . .	69
2.4	Pokazivači na funkcije . . . . .	72
2.5	Rešenja . . . . .	74
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>113</b>
3.1	Pretraživanje . . . . .	113
3.2	Sortiranje . . . . .	118
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	127
3.4	Rešenja . . . . .	132
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>207</b>
4.1	Liste . . . . .	207
4.2	Stabla . . . . .	219
4.3	Rešenja . . . . .	228
<b>5</b>	<b>Ispitni rokovi</b>	<b>325</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	325
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	327
5.3	Rešenja . . . . .	329



# 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 predstavlja kompleksan broj i sadrži realan i imaginaran deo kompleksnog broja.
- (b) Napisati funkciju `ucitaj_kompleksan_broj` koja učitava kompleksan broj sa standardnog ulaza.
- (c) Napisati funkciju `ispisi_kompleksan_broj` 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 `realan_deo` koja računa vrednosti realnog dela broja.
- (e) Napisati funkciju `imaginaran_deo` koja računa vrednosti imaginarnog dela broja.
- (f) Napisati funkciju `modulo` koja računa modulo kompleksnog broja.
- (g) Napisati funkciju `konjugovan` koja računa konjugovano-kompleksni broj svog argumenta.
- (h) Napisati funkciju `saberi` koja sabira dva kompleksna broja.
- (i) Napisati funkciju `oduzmi` koja oduzima dva kompleksna broja.
- (j) Napisati funkciju `mnozi` koja množi dva kompleksna broja.

## 1 Uvodni zadaci

---

(k) Napisati funkciju `argument` koja računa argument kompleksnog broja.

Napisati program koji testira prethodno napisane funkcije za dva kompleksna broja  $z_1$  i  $z_2$  koja se unose sa standardnog ulaza i ispisuje:

- (a) realni deo, imaginarni deo i moduo kompleksnog broja  $z_1$ ,
- (b) konjugovano kompleksan broj i argument broja  $z_2$ ,
- (c) zbir, razliku i proizvod brojeva  $z_1$  i  $z_2$ .

*Test 1*

```
Ulaz:      Unesite realan i imaginaran deo kompleksnog broja: 1 -3
           Unesite realan i imaginaran deo kompleksnog broja: -1 4
Izlaz:      (1.00 - 3.00 i)
           realan_deo: 1
           imaginaran_deo: -3.000000
           moduo 3.162278

           (-1.00 + 4.00 i)
           Njegov konjugovano kompleksan broj: (-1.00 - 4.00 i)
           Argument kompleksnog broja: 1.815775

           (1.00 - 3.00 i) + (-1.00 + 4.00 i) = (1.00 i)
           (1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)
           (1.00 - 3.00 i) * (-1.00 + 4.00 i) = (11.00 + 7.00 i)
```

[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.

*Test 1*

```
Ulaz:      Unesite realan i imaginaran deo kompleksnog broja: -5 2
Izlaz:      Polarni oblik kompleksnog broja je 5.39 * e^i * 2.76
```

[Rešenje 1.2]

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

- (a) Definisati strukturu `Polinom` koja predstavlja polinom (stepena najviše 20). 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 koja ispisuje polinom na standardni izlaz u što lepšem obliku.
- (c) Napisati funkciju koja učitava polinom sa standardnog ulaza.
- (d) Napisati funkciju za izračunavanje vrednosti polinoma u datoj tački koristeći Hornerov algoritam.
- (e) Napisati funkciju koja sabira dva polinoma.
- (f) Napisati funkciju koja množi dva polinoma.

Napisati program koji testira prethodno napisane funkcije tako što se najpre unosi polinom  $p$  (stepen polinoma, a zatim i koeficijenti) i ispisuje na standardan izlaz u odgovarajućem obliku. Nakon toga se od korisnika traži da unese tačku u kojoj se računa vrednost tog polinoma a zatim se ispisuje izračunata vrednost zaokružena na dve decimale. Nakon toga se unosi polinom  $q$ , a potom se ispisuju zbir i proizvod polinoma  $p$  i  $q$ . Na kraju se sa standardnog ulaza unosi broj  $n$ , a potom se ispisuje  $n$ -ti izvod polinoma  $p$ .

### Upotreba programa 1

```
Unesite polinom (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
Unesite tačku u kojoj racunate vrednost polinoma
5
Vrednost polinoma u tacki je 194.00
Unesite drugi polinom (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
2 1 0 1
Zbir polinoma je: 1.00x3+3.00x2+3.00x+5.00
Prozvod polinoma je: 1.00x5+2.00x4+4.00x3+6.00x2+3.00x+4.00
Unesite izvod polinoma koji zelite:
2
2. izvod prvog polinoma je: 6.00x+4.00
```

[Rešenje 1.3]

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

- (a) Definisati strukturu za reprezentovanje razlomaka.
- (b) Napisati funkcije za učitavanje i ispis razlomaka.
- (c) Napisati funkcije koje vraćaju brojilac i imenilac.
- (d) Napisati funkciju koja vraća vrednost razlomka kao `double` vrednost.
- (e) Napisati funkciju koja izračunava recipročnu vrednost razlomka.
- (f) Napisati funkciju koja skraćuje dati razlomak.

## 1 Uvodni zadaci

(g) Napisati funkcije koje sabiraju, oduzimaju, množe i dele dva razlomka.

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 koji su dobijeni kao zbir, razlika, proizvod i količnik razlomka **r1** i recipročne vrednosti razlomka **r2**.

### Test 1

```

Ulaz:  Unesite imenilac i brojilac prvog razlomka: 1 2
        Unesite imenilac i brojilac drugog razlomka: 3 1
Izlaz:  Zbir je 5/6
        Razlika je 1/6
        Proizvod je 1/6
        Kolicnik je 3/2

```

## 1.2 Algoritmi za rad sa bitovima

**Zadatak 1.5** Napisati funkciju `print_bits` koja štampa bitove u binarnom zapisu neoznačenog celog broja  $x$ . Napisati program koja testira funkciju `print_bits` za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

*Test 1*

```
|| Ulaz:    0x7F  
| Izlaz:   00000000000000000000000000001111111
```

### Test 2

```
Ulaz:    0x80  
Izlaz:   000000000000000000000000000000000000000000000000000
```

### Test 3

```
Ulaz: 0x00FF00FF
Izlaz: 00000000111111111000000001111111
```

*Test 4*

```
Ulaz: 0xFFFFFFFF
Izlaz: 111111111111111111111111111111111111
```

*Test*

```
Ulaz: 0xABCDE123
Izlaz: 10101011110011011110000100100011
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkciju koja broji bitove postavljene na 1 u zapisu celog broja  $x$ , tako što se pomeranje vrši nad

- (a) maskom,
- (b) brojem  $x$ .

Napisati program koji testira tu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

*Test 1*

```

|| Ulaz: 0x7F
|| Izlaz:
|| Broj jedinica u zapisu je 7
    
```

*Test 2*

```

|| Ulaz: 0x80
|| Izlaz:
|| Broj jedinica u zapisu je 1
    
```

*Test 3*

```

|| Ulaz: 0x00FF00FF
|| Izlaz:
|| Broj jedinica u zapisu je 16
    
```

*Test 4*

```

|| Ulaz: 0xFFFFFFFF
|| Izlaz:
|| Broj jedinica u zapisu je 32
    
```

*Test 4*

```

|| Ulaz: 0xABCDE123
|| Izlaz:
|| Broj jedinica u zapisu je 17
    
```

[Rešenje 1.6]

**Zadatak 1.7** Napisati funkciju **najveci** koja određuje najveći broj koji se može zapisati istim binarnim ciframa kao dati broj i funkciju **najmanji** koja određuje najmanji broj koji se može zapisati istim binarnim ciframa kao dati broj.

Napisati program koji testira prethodno napisane funkcije tako što prikazuje binarnu reprezentaciju brojeva koji se dobijaju nakon poziva funkcije **najveci**, odnosno **najmanji** za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

*Test 1*

```

|| Ulaz: 0x7F
|| Izlaz:
|| Najveci:
|| 11111110000000000000000000000000
|| Najmanji:
|| 0000000000000000000000000000111111
    
```

*Test 2*

```

|| Ulaz: 0x80
|| Izlaz:
|| Najveci:
|| 10000000000000000000000000000000
|| Najmanji:
|| 00000000000000000000000000000001
    
```

## 1 Uvodni zadaci

---

### Test 3

```
|| Ulaz:  0x00FF00FF
|| Izlaz:
|| Najveci:
|| 11111111111111111000000000000000
|| Najmanji:
|| 00000000000000000111111111111111
```

### Test 4

```
|| Ulaz:  0xFFFFFFFF
|| Izlaz:
|| Najveci:
|| 11111111111111111111111111111111
|| Najmanji:
|| 11111111111111111111111111111111
```

### Test 4

```
|| Ulaz:  0xABCDE123
|| Izlaz:
|| Najveci:
|| 11111111111111111000000000000000
|| Najmanji:
|| 00000000000000000111111111111111
```

[Rešenje 1.7]

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

- (a) Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja, počevši od pozicije  $p$  postave na 0.
- (b) Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja, počevši od pozicije  $p$  postave na 1.
- (c) Napisati funkciju koja određuje broj koji se dobija od  $n$  bitova datog broja, počevši od pozicije  $p$  i vraća ih kao bitove najmanje težine rezultata.
- (d) Napisati funkciju koja vraća broj koji se dobija upisivanjem poslednjih  $n$  bitova broja  $y$  u broj  $x$ , počevši od pozicije  $p$ .
- (e) Napisati funkciju koja vraća broj koji se dobija invertovanjem  $n$  bitova broja  $x$  počevši od pozicije  $p$ .
- (f) Napisati program koji testira prethodno napisane funkcije.

Program treba da testira prethodno napisane funkcije nad neoznačenim celim brojem koji se unosi sa standardnog ulaza. *Napomena: pozicije se broje počev od pozicije bita najmanje težine, pri čemu je pozicija bita najmanje težine nula.*

*Test 1*

```
Ulaz:      235 5 10 127
Izlaz:
  Broj      235          = 00000000000000000000000011101011
  reset(    235,      5,    10) = 000000000000000000000000000101011
  set(      235,      5,    10) = 0000000000000000000000001111101011
  get_bits( 235,      5,    10) = 000000000000000000000000000000011
  y =                                127 = 00000000000000000000000001111111
  set_n_bits( 235,      5,    10, 127) = 0000000000000000000000001111101011
  invert(    235,      5,    10) = 0000000000000000000000001110010111
```

[Rešenje 1.8]

**Zadatak 1.9** Rotiranje ulevo podrazumeva 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 u levo datog celog broja `x`.
- Napisati funkciju `rotate_right` koja određuje broj koji se dobija rotiranjem `k` puta u desno datog celog neoznačenog broja `x`.
- Napisati funkciju `rotate_right_signed` koja određuje broj koji se dobija rotiranjem `k` puta u desno datog celog broja `x`.

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

### Test 1

```

Ulaz:    B10011A7 5
Izlaz:
  x      = 10110001000000000001000110100111
  rotate_left(2969571751,    5)  = 001000000000000100011010011110110
  rotate_right(2969571751,    5)  = 001111010000100000000000010001101
  rotate_right_signed(2969571751, 5) = 001111010000100000000000010001101

```

[Rešenje 1.9]

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

## 1 Uvodni zadaci

---

a potom i binarnu reprezentaciju broja dobijenog nakon poziva funkcije `mirror` za uneti broj.

*Test 1*

```
|| Ulaz: 255
|| Izlaz:
||      0000000000000000000000001001010101
||      10101010010000000000000000000000
```

[Rešenje 1.10]

**Zadatak 1.11** Napisati funkciju `int Broj01(unsigned int n)` koja za dati broj `n` vraća 1 ako u njegovom binarnom zapisu ima više jedinica nego nula, a inače vraća 0. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

*Test 1*

```
|| Ulaz: 10
|| Izlaz: 0
```

*Test 2*

```
|| Ulaz: 1024
|| Izlaz: 0
```

*Test 3*

```
|| Ulaz: 2147377146
|| Izlaz: 1
```

*Test 4*

```
|| Ulaz: 1111111115
|| Izlaz: 0
```

[Rešenje 1.11]

**Zadatak 1.12** Napisati funkciju koja broji koliko se puta kombinacija 11 (dve uzastopne jedinice) pojavljuje u binarnom zapisu celog neoznačenog broja `x`. Tri uzastopne jedinice se broje dva puta. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

*Test 1*

```
|| Ulaz: 11
|| Izlaz: 1
```

*Test 2*

```
|| Ulaz: 1024
|| Izlaz: 0
```

*Test 3*

```
|| Ulaz: 2147377146
|| Izlaz: 22
```

*Test 4*

```
|| Ulaz: 1111111115
|| Izlaz: 6
```

[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  $i$ ,  $j$ . Pozicije  $i$ ,  $j$  se učitavaju kao parametri komandne linije. Smatrati da je krajnji desni bit binarne reprezentacije 0-ti bit. Pri rešavanju nije dozvoljeno koristiti pomoćni niz niti aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ .

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
Poziv: ./a.out 1 2	Poziv: ./a.out 1 2	Poziv: ./a.out 12 12
Ulaz: 11	Ulaz: 1024	Ulaz: 12345
Izlaz: 13	Izlaz: 1024	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.

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

[Rešenje 1.14]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
Ulaz: 123 10	Ulaz: 3251 0	Ulaz: 12541 1024
Izlaz: 4294967285	Izlaz: 4294967295	Izlaz: 4294966271

**Zadatak 1.16** ++ Napisati funkciju koja računa koliko petica bi imao ceo neoznačen broj  $x$  u oktalnom zapisu. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>   Ulaz: 123    Izlaz: 0</pre>	<pre>   Ulaz: 3245    Izlaz: 2</pre>	<pre>   Ulaz: 100328    Izlaz: 1</pre>

### 1.3 Rekurzija

**Zadatak 1.17** Napisati rekurzivnu funkciju koja izračunava  $x^k$ , za dati ceo broj  $x$  i prirodan broj  $k$ . Napisati program koji testira napisanu funkciju za vrednosti koje se unose sa standardnog ulaza.

*Test 1*

```
|| Ulaz: 2 10
|| Izlaz: 1024
```

[Rešenje 1.17]

**Zadatak 1.18** Koristeći uzajamnu (posrednu) rekurziju napisati naredne dve funkcije:

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

Napisati program koji testira napisanu funkciju tako što se za heksadekadnu vrednost koja se unosi sa standardnog ulaza ispisuje da li je paran ili neparan.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz: 11    Izlaz: Uneti broj ima paran broj cifara</pre>	<pre>   Ulaz: 123    Izlaz: Uneti broj ima neparan             broj cifara</pre>

[Rešenje 1.18]

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



*Test 1*

```

|| Ulaz:   Unesite n (<= 12): 5
|| Izlaz:  5! = 120

```

[Rešenje 1.19]

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

$$F(0) = 0$$

$$F(1) = 1$$

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

Napisati rekurzivnu funkciju koja računa  $n$ -ti element u nizu  $F$  ali tako da se problemi manje dimenzije rešavaju samo jedan put. Napisati program koji testira napisane funkcije za poizvoljan broj  $n$  ( $n \in \mathbb{N}$ ) unet sa standardnog ulaza.

*Test 1*

```

|| Ulaz:   Unesi koeficijente
||         2 3
||         Unesi koji clan niza racunamo
||         5
|| Izlaz:  F(5) = 61

```

[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]

## 1 Uvodni zadaci

---

**Zadatak 1.22** Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva. Napisati program koji testira napisane funkcije. 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.

*Test 1*

```
|| Ulaz:  5 1 2 3 4 5
|| Izlaz: Suma elemenata je 15
```

[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, i njegovi elementi se unose sve do kraja ulaza.

*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
```

*Test 4*

```
|| Ulaz:  5
|| Izlaz: 5
```

[Rešenje 1.23]

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

*Test 1*

```
|| Ulaz:  3 1 2 3 1 2 3
|| Izlaz: 14
```

*Test 2*

```
|| Ulaz:  2 3 5 2 6
|| Izlaz: 36
```

*Test 3*

```
|| Ulaz:  0
|| Izlaz: 0
```

[Rešenje 1.24]

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

*Test 1*

```
|| Ulaz: 4 1 2 3 4
|| Izlaz: 1
```

*Test 2*

```
|| Ulaz: 11 3 2 11 14 11 43 1
|| Izlaz: 2
```

*Test 3*

```
|| Ulaz: 1 3 21 5 6
|| Izlaz: 0
```

[Rešenje 1.25]

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

*Test 1*

```
|| Ulaz: 1 2 3 4 1 2 3 4 5
|| Izlaz: da
```

*Test 2*

```
|| Ulaz: 1 2 3 11 1 2 4 3 6
|| Izlaz: ne
```

*Test 3*

```
|| Ulaz: 1 2 3 1 2
|| Izlaz: ne
```

[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: 0x80
|| Izlaz: 1
```

*Test 3*

```
|| Ulaz: 0x00FF00FF
|| Izlaz: 16
```

## 1 Uvodni zadaci

*Test 4*

```
Ulaz: 0xFFFFFFFF
Izlaz: 32
```

[Rešenje 1.27]

**Zadatak 1.28** Napisati rekurzivnu funkciju koja štampa bitovsku reprezentaciju neoznačenog celog broja, i program koji je testira za vrednost koja se zadaje sa standardnog ulaza.

### Test 1

```
Ulaz:      10
Izlaz:     000000000000000000000000000000001010
```

**Zadatak 1.29** Napisati rekurzivnu funkciju za određivanje najveće cifre u oktalnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. *Uputstvo: binarne cifre grupisati u podgrupe od po tri cifre, počev od bitova najmanje težine.*

### Test 1

```
Ulaz: 5
Izlaz: 5
```

### Test 2

```
|| Ulaz: 125
|| Izlaz: 7
```

### Test 3

```
Ulaz: 8
Izlaz: 1
```

*Test 4*

```
Ulaz: 10
Izlaz: 2
```

[Rešenje 1.29]

**Zadatak 1.30** Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. *Uputstvo: binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.*

*Test 1*

```
Ulaz: 5
Izlaz: 5
```

### Test 2

```
Ulaz: 16
Izlaz: 1
```

### Test 3

```
Ulaz: 18
Izlaz: 2
```

*Test 4*

```
|| Ulaz: 165
|| Izlaz: 10
```

[Rešenje 1.30]

**Zadatak 1.31** Napisati rekurzivnu funkciju **palindrom** koja ispituje da li je data niska palindrom. Napisati program koji testira ovu funkciju. Pretpostaviti da niska neće imati više od 31 karaktera, i da se unosi sa standardnog ulaza.

*Test 1*

```
|| Ulaz:  programiranje
|| Izlaz:  ne
```

*Test 2*

```
|| Ulaz:  anavolimilovana
|| Izlaz:  da
```

*Test 3*

```
|| Ulaz:  a
|| Izlaz:  da
```

*Test 4*

```
|| Ulaz:  aba
|| Izlaz:  da
```

*Test*

```
|| Ulaz:  aa
|| Izlaz:  da
```

[Rešenje 1.31]

\* **Zadatak 1.32** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za poizvoljan prirodan broj  $n$  ( $n \leq 50$ ) unet sa standardnog ulaza.

*Test 1*

```
|| Ulaz:  Unesite duzinu permutacije: 3
|| Izlaz:  1 2 3
||         1 3 2
||         2 1 3
||         2 3 1
||         3 1 2
||         3 2 1
```

[Rešenje 1.32]

\* **Zadatak 1.33** Paskalov trougao se dobija tako što mu je svako polje (izuzev jedinica po krajevima) zbir jednog polja levo i jednog polja iznad.

```
      1
     1 1
    1 2 1
   1 3 3 1
```

## 1 Uvodni zadaci

```
    1   4   6   4   1
   1   5  10  10   5   1
```

- (a) Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta  $\binom{n}{k}$ , tj. vrednost polja  $(n, k)$ , gde je  $n$  redni broj hipotenuze, a  $k$  redni broj elementa u tom redu (na toj hipotenuzi). Brojanje počinje od nule. Na primer vrednost polja  $(4, 2)$  je 6.
- (b) Napisati rekurzivnu funkciju koja izračunava  $d_n$  kao sumu elemenata  $n$ -te hipotenuze Paskalovog trougla.

Napisati program koji za unetu veličinu Paskalovog trougla i hipotenuzu najpre iscrtava Paskalov trougao a zatim sumu elemenata hipotenuze.

*Test 1*

```
Ulaz:  5 3
Izlaz:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
8
```

*Test 2*

```
Ulaz:  6 5
Izlaz:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
32
```

[Rešenje 1.33]

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

*Test 1*

```
Ulaz:  3
Izlaz:
a a a
a a b
a b a
a b b
b a a
b a b
b b a
b b b
```

**Zadatak 1.35** *Hanojske kule*: Data su tri vertikalna štapa, na jednom se nalazi  $n$  diskova poluprečnika 1,2,3,... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je premestiti diskove na

drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, a preostali štap se koristi kao pomoćni štap prilikom premeštanja.

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

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

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

## 1.4 Rešenja

### Rešenje 1.1

```

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

4  /* Struktura kojom predstavljamo kompleksan broj, cuvajuci
     njegov realan i imaginaran deo */
6  typedef struct {
7      float real;
8      float imag;
9  } KompleksanBroj;

10
11 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
12 kompleksnog broja i smesta ih u strukturu cija adresa je
13 argument funkcije */
14 void ucitaj_kompleksan_broj(KompleksanBroj * z)
15 {
16     printf("Unesite realan i imaginaran deo kompleksnog broja: ");
17     scanf("%f", &z->real);
18     scanf("%f", &z->imag);
19 }

20
21 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji
22 joj se salje kao argument u obliku (x + y i) Ovoj funkciji se
23 kompleksan broj prenosi po vrednosti, jer za ispis nam nije
24 neophodno da imamo adresu */
25 void ispisi_kompleksan_broj(KompleksanBroj z)
26 {

```

```
printf("(");
28 if (z.real != 0) {
    printf("%.2f", z.real);
30
    if (z.imag > 0)
32         printf(" + %.2f i", z.imag);
    else if (z.imag < 0)
34         printf(" - %.2f i", -z.imag);
    } else
36         printf("%.2f i", z.imag);

38 if (z.imag == 0 && z.real == 0)
    printf("0");
40
42 printf(")");
}

44 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
float realan_deo(KompleksanBroj z)
46 {
    return z.real;
48 }

50 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
float imaginaran_deo(KompleksanBroj z)
52 {
    return z.imag;
54 }

56 /* Funkcija vraca vrednost modula kompleksnog broja koji joj se
    salje kao argument */
58 float moduo(KompleksanBroj z)
{
60     return sqrt(z.real * z.real + z.imag * z.imag);
}

62
64 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
    odgovara kompleksnom broju poslatom kao argument */
KompleksanBroj konjugovan(KompleksanBroj z)
66 {
    KompleksanBroj z1 = z;
68     z1.imag *= -1;
    return z1;
70 }

72 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
    argumenata funkcije */
74 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
{
76     KompleksanBroj z = z1;

78     z.real += z2.real;
```



```
    z.imag += z2.imag;
80
    return z;
82 }

84 /* Funkcija vraca kompleksan broj cija vrednost je jednaka
    razlici argumenata funkcije */
86 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
88 {
    KompleksanBroj z = z1;

90    z.real -= z2.real;
    z.imag -= z2.imag;

92    return z;
94 }

96 /* Funkcija vraca kompleksan broj cija vrednost je jednaka
    proizvodu argumenata funkcije */
98 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
100 {
    KompleksanBroj z;

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

104    return z;
106 }

108 /* Funkcija vraca argument kompleksnog broja koji je funkciji
    poslat kao argument */
110 float argument(KompleksanBroj z)
112 {
    return atan2(z.imag, z.real);
114 }

116 /* U main() funckiji testiramo sve funckije koje smo definisali */
int main()
118 {
    /* deklariseemo promenljive tipa KompleksanBroj */
120    KompleksanBroj z1, z2;

122    /* Ucitavamo prvi kompleksan broj */
    ucitaj_kompleksan_broj(&z1);

124    /* Ucitavamo i drugi kompleksan broj */
126    ucitaj_kompleksan_broj(&z2);

128    /* Ispisujemo prvi kompleksan broj, a zatim i njegov realan i
    imaginaran deo, kao i moduo kompleksnog broja z1 */
130    ispisi_kompleksan_broj(z1);
```

```

132 printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
    realan_deo(z1), imaginaran_deo(z1), moduo(z1));
134 printf("\n");
136 /* Ispisujemo drugi kompleksan broj, a zatim i racunamo i
    ispisujemo konjugovano kompleksan broj od z2 */
138 ispisi_kompleksan_broj(z2);
139 printf("\nNjegov konjugovano kompleksan broj: ");
140 ispisi_kompleksan_broj(konjugovan(z2));
142 /* Testiramo funkciju koja racuna argument kompleksnih brojeva
    */
143 printf("\nArgument kompleksnog broja: %.f\n", argument(z2));
144 printf("\n");
146 /* Testiramo sabiranje kompleksnih brojeva */
147 ispisi_kompleksan_broj(z1);
148 printf(" + ");
149 ispisi_kompleksan_broj(z2);
150 printf(" = ");
151 ispisi_kompleksan_broj(saberi(z1, z2));
152 printf("\n");
154 /* Testiramo oduzimanje kompleksnih brojeva */
155 printf("\n");
156 ispisi_kompleksan_broj(z1);
157 printf(" - ");
158 ispisi_kompleksan_broj(z2);
159 printf(" = ");
160 ispisi_kompleksan_broj(oduzmi(z1, z2));
161 printf("\n");
162 /* Testiramo mnozenje kompleksnih brojeva */
163 printf("\n");
164 ispisi_kompleksan_broj(z1);
165 printf(" * ");
166 ispisi_kompleksan_broj(z2);
167 printf(" = ");
168 ispisi_kompleksan_broj(mnozi(z1, z2));
169 printf("\n");
170 /* Program se zavrшава uspesno, tj, bez greske */
172 return 0;
}

```

### Rešenje 1.2

```

2 /* Uključujemo zaglavlje neophodno za rad sa kompleksnim brojevima
   * Ovdje je to neophodno jer nam je neophodno da bude poznata
     definicija tipa KompleksanBroj
   * i da budu uključena zaglavlja standardne biblioteke koja smo već
     naveli u complex.h

```

```

4  */
   #include "complex.h"
6
   /* Funkcija ucitava sa standardnog ulaza realan i imaginaran deo
      kompleksnog broja i smesta ih u strukturu cija adresa je argument
      funkcije */
8  void ucitaj_kompleksan_broj(KompleksanBroj* z) {
   printf("Unesite realan i imaginaran deo kompleksnog broja: ");
10  scanf("%f", &z->real);
   scanf("%f", &z->imag);
12  }

14  /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
      se salje kao argument u obliku (x + y i)
      Ovoj funkciji se kompleksan broj prenosi po vrednosti, jer za
      ispis nam nije neophodno da imamo adresu
16  */
   void ispisi_kompleksan_broj(KompleksanBroj z) {
18     printf("(");
     if(z.real != 0) {
20         printf("%.2f", z.real);

22         if(z.imag > 0)
             printf(" + %.2f i", z.imag);
24         else if(z.imag < 0)
             printf(" - %.2f i", -z.imag);
26         }
     else
28         printf("%.2f i", z.imag);

30     if(z.imag == 0 && z.real == 0 )
         printf("0");
32
     printf(")");
34  }

36  /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
   float realan_deo(KompleksanBroj z) {
38     return z.real;
   }
40
   /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
42  float imaginaran_deo(KompleksanBroj z) {
     return z.imag;
44  }

46  /* Funkcija vraca vrednost modula kompleksnog broja koji joj se salje
      kao argument */
   float moduo(KompleksanBroj z) {
48     return sqrt(z.real* z.real + z.imag* z.imag);
   }
50

```

## 1 Uvodni zadaci

---

```
/* Funkcija vraća vrednost konjugovano kompleksnog broja koji
   odgovara kompleksnom broju poslatom kao argument */
52 KompleksanBroj konjugovan(KompleksanBroj z) {
    KompleksanBroj z1 = z;
54     z1.imag *= -1;
    return z1;
56 }

58 /* Funkcija vraća kompleksan broj čija vrednost je jednaka zbiru
   argumenata funkcije */
KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2) {
60     KompleksanBroj z = z1;

62     z.real += z2.real;
    z.imag += z2.imag;
64
    return z;
66 }

68 /* Funkcija vraća kompleksan broj čija vrednost je jednaka razlici
   argumenata funkcije */
KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2) {
70     KompleksanBroj z = z1;

72     z.real -= z2.real;
    z.imag -= z2.imag;
74
    return z;
76 }

78 /* Funkcija vraća kompleksan broj čija vrednost je jednaka proizvodu
   argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2) {
80     KompleksanBroj z;

82     z.real = z1.real * z2.real - z1.imag * z2.imag;
    z.imag = z1.real * z2.imag + z1.imag * z2.real;
84
    return z;
86 }

88 /* Funkcija vraća argument kompleksnog broja koji je funkciji poslat
   kao argument */
float argument(KompleksanBroj z) {
90     return atan2(z.imag, z.real);
}

1 /*
   Zaglavlje complex.h sadrži definiciju tipa KompleksanBroj i
   deklaracije funkcija za rad sa kompleksnim brojevima.
3   Zaglavlje nikada ne treba da sadrži definicije funkcija.
```

```

    Bilo koji program koji bi hteo da koristi ove brojeve i funkcije iz
    ove biblioteke, neophodno je da ukljuci ovo zaglavlje
5  */

7  /* Ovim pretprocesorskim direktivama zakljucavamo zaglavlje i time
    onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci.
    Niska posle kljucne reci ifndef je proizvoljna ali treba da se
    ponovi u narednoj pretprocesorskoj define direktivi
9  */
11 #ifndef _COMPLEX_H
13 #define _COMPLEX_H

13 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
    koje se koriste u definicijama funkcija koje smo naveli u complex
    .c */
15 #include <stdio.h>
15 #include <math.h>

17 /* struktura kojom predstavljamo kompleksan broj, cuvajuci njegov
    realan i imaginaran deo */
19 typedef struct {
19     float real;
19     float imag;
21 } KompleksanBroj;

23 /* Deklaracije funkcija za rad sa kompleksnim brojevima.
    Sve one su definisane u complex.c */
25 void ucitaj_kompleksan_broj(KompleksanBroj* z) ;

27 void ispisi_kompleksan_broj(KompleksanBroj z) ;

29 float realan_deo(KompleksanBroj z) ;

31 float imaginaran_deo(KompleksanBroj z);

33 float moduo(KompleksanBroj z);

35 KompleksanBroj konjugovan(KompleksanBroj z) ;

37 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) ;

39 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) ;

41 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) ;

43 float argument(KompleksanBroj z) ;

45 /* Kraj zakljucanog dela */
    #endif

2  /*
    * Koristimo korektno definisanu biblioteku kompleksnih brojeva.

```

## 1 Uvodni zadaci

```

2  * U zaglavlju complex.h nalazi se definicija kompleksnog broja i
3  * popis deklaracija podrzanih funkcija
4  * a u complex.c se nalaze njihove definicije.
5  *
6  * Ovde pisemo i main() funkciju drugaciju od prethodnog zadatka.
7  *
8  * I dalje kompilacija programa se najjednostavnije postize naredbom
9  * gcc -Wall -lm -o izvrsni complex.c main.c
10
11  Kompilaciju mozemo uraditi i na sledeci nacin:
12  gcc -Wall -c -o complex.o complex.c
13  gcc -Wall -c -o main.o main.c
14  gcc -lm -o complex complex.o main.o
15  */
16
17
18  #include <stdio.h>
19  /* Ukljuujemo zaglavlje neophodno za rad sa kompleksnim brojevima */
20  #include "complex.h"
21
22  /* U main funkciji za uneti kompleksan broj ispisujemo njegov polarni
23  * oblik */
24  int main() {
25      KompleksanBroj z;
26
27      /* Ucitavamo kompleksan broj */
28      ucitaj_kompleksan_broj(&z);
29
30      printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f\n",
31             moduo(z), argument(z));
32
33      return 0;
34  }
```

### Rešenje 1.3

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "polinom.h"
4
5
6  /* Funkcija koja ispisuje polinom na standardan izlaz u citljivom
7  * obliku.
8  * Kako bi uštedeli kopiranje cele strukture, polinom prenosimo po
9  * adresi */
10 void ispisi(const Polinom * p)
11 {
12     int i;
13     for (i = p->stepen; i >= 0; i--) {
14         if (p->koeff[i]) {
15             if (p->koeff[i] >= 0 && i != p->stepen)
16                 printf("%d", p->koeff[i]);
17             else
18                 printf("%d", -p->koeff[i]);
19             printf("x^%d", i);
20             if (i > 0)
21                 printf(" + ");
22         }
23     }
24     printf("\n");
25 }
```

```

15     putchar('+');
16     if (i > 1)
17         printf("%.2fx^%d", p->koef[i], i);
18     else if (i == 1)
19         printf("%.2fx", p->koef[i]);
20     else
21         printf("%.2f", p->koef[i]);
22 }
23 }
24 putchar('\n');
25 }
26
27 /* Funkcija koja ucitava polinom sa tastature */
28 Polinom ucitaj()
29 {
30     int i;
31     Polinom p;
32
33     /* Ucitavamo stepen polinoma */
34     scanf("%d", &p.stepen);
35
36     /* Ponavljamo ucitavanje stepena sve dok ne unesemo stepen iz
37     dozvoljenog opsega */
38     while (p.stepen > MAX_STEPEN || p.stepen < 0) {
39         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
40         scanf("%d", &p.stepen);
41     }
42
43     /* Unosimo koeficijente polinoma */
44     for (i = p.stepen; i >= 0; i--)
45         scanf("%lf", &p.koef[i]);
46     return p;
47 }
48
49 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
50 algoritmom */
51 /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
52 double izracunaj(const Polinom * p, double x)
53 {
54     double rezultat = 0;
55     int i = p->stepen;
56     for (; i >= 0; i--)
57         rezultat = rezultat * x + p->koef[i];
58     return rezultat;
59 }
60
61 /* Funkcija koja sabira dva polinoma */
62 Polinom saberi(const Polinom * p, const Polinom * q)
63 {
64     Polinom rez;
65     int i;

```

```
        rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
65
        for (i = 0; i <= rez.stepen; i++)
67            rez.koef[i] =
                (i > p->stepen ? 0 : p->koef[i]) + (i >
69                    q->stepen ? 0 : q->
                        koef[i]);
71        return rez;
73    }

75    /* Funkcija mnozi dva polinoma p i q */
    Polinom pomnozi(const Polinom * p, const Polinom * q)
77    {
        int i, j;
79        Polinom r;

81        r.stepen = p->stepen + q->stepen;
        if (r.stepen > MAX_STEPEN) {
83            fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
            exit(EXIT_FAILURE);
85        }

87        for (i = 0; i <= r.stepen; i++)
            r.koef[i] = 0;

89        for (i = 0; i <= p->stepen; i++)
91            for (j = 0; j <= q->stepen; j++)
                r.koef[i + j] += p->koef[i] * q->koef[j];

93        return r;
95    }

97    /* Funkcija racuna izvod polinoma p */
    Polinom izvod(const Polinom * p)
99    {
        int i;
101        Polinom r;

103        if (p->stepen > 0) {
            r.stepen = p->stepen - 1;
105
            for (i = 0; i <= r.stepen; i++)
107                r.koef[i] = (i + 1) * p->koef[i + 1];
            } else
109            r.koef[0] = r.stepen = 0;

111        return r;
    }

113
115    /* Funkcija racuna n-ti izvod polinoma p */
    Polinom nIzvod(const Polinom * p, int n)
```



```

{
117     int i;
        Polinom r;

119     if (n < 0) {
121         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
        exit(EXIT_FAILURE);
123     }

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

127     r = izvod(p);
129     for (i = 1; i < n; i++)
        r = izvod(&r);

131     return r;
133 }

```

```

1
/* Ovim preprocesorskim direktivama zakljucavamo zaglavlje i time
   onemogucujemo
3   da se sadrzaj zaglavlja vise puta ukljuci
*/
5 #ifndef _POLINOM_H
   #define _POLINOM_H
7
   #include <stdio.h>
9   #include <stdlib.h>

11  /* Maksimalni stepen polinoma */
   #define MAX_STEPEN 20
13

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

23 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku
   Polinom prenosimo po adresi, da bi ustedeli kopiranje cele
   strukture,
25   vec samo prenosimo adresu na kojoj se nalazi polinom kog
   ispisujemo */
   void ispisi(const Polinom * p);
27

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

```

## 1 Uvodni zadaci

---

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

35 /* Funkcija koja sabira dva polinoma */
    Polinom saberi(const Polinom * p, const Polinom * q);
37
    /* Funkcija mnozi dva polinoma p i q */
39 Polinom pomnozi(const Polinom * p, const Polinom * q);

41 /* Funkcija racuna izvod polinoma p */
    Polinom izvod(const Polinom * p);
43
    /* Funkcija racuna n-ti izvod polinoma p */
45 Polinom nIzvod(const Polinom * p, int n);
    #endif

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

4 /*
    Prevodjenje:
6 gcc -o test-polinom polinom.c main.c

8 ili:
    gcc -c polinom.c
10 gcc -c main.c
    gcc -o test-polinom polinom.o main.o
12 */

14 int main(int argc, char **argv)
    {
16         Polinom p, q, r;
            double x;
18         int n;

20         /* Unos polinoma */
            printf
22         ("Unesite polinom (prvo stepen, pa zatim koeficijente od najveceg
            stepena do nultog):\n");
            p = učitaj();
24
            /* Ispis polinoma */
26         ispisi(&p);

28         printf("Unesite tacku u kojoj racunate vrednost polinoma\n");
            scanf("%lf", &x);
30
            /* Ispisujemo vrednost polinoma u toj tacki */
32         printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&p, x));
```

```

34     /* Unesimo drugi polinom */
    printf
36 ("Unesite drugi polinom (prvo stepen, pa zatim koeficijente od
    najveceg stepena do nultog):\n");
    q = ucitaj();

38     /* Sabiramno polinome i ispisujemo zbir ta dva polinoma */
    r = saberi(&p, &q);
    printf("Zbir polinoma je: ");
    ispisi(&r);

42     /* Mnozimo polinome i ispisujemo proizvod ta dva polinoma */
    r = pomnozi(&p, &q);
    printf("Prozvod polinoma je: ");
    ispisi(&r);

44     /* Izvod polinoma */
    printf("Unesite izvod polinoma koji zelite:\n");
    scanf("%d", &n);
    r = nIzvod(&p, n);
    printf("%d. izvod prvog polinoma je: ", n);
    ispisi(&r);

46     /* Uspesno završavamo program */
    return 0;

58 }

```

### Rešenje 1.5

```

#include <stdio.h>

2
/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
4  celog broja u memoriji. Bitove u zapisu broja
    ispisujemo sa leva na desno, tj. od bita najvece tezine ka
6  bitu najmanje tezine. */
void print_bits(unsigned x)
8 {

10     /* broj bitova celog broja */
    unsigned velicina = sizeof(unsigned) * 8;
12     /* maska koja se koristi za "ocitavanje" bitova */
    unsigned maska;

14     /* Pocetna vrednost maske se postavlja na broj ciji binarni
        zapis na mestu bita najvece tezine sadrzi jedinicu, a na
16     svim ostalim mestima sadrzi nulu. U svakoj iteraciji ova
        jedinica se pomera u desno, kako bi se ocitao naredni bit.
        Odgovarajuci
18     karakter, ('0' ili '1'), ispisuje se na standardnom izlazu. Zbog
        siftovanja maske u desno, koja na pocetku ima najvisi bit
20     postavljen na 1, neophodno je da maska bude neoznaceni ceo

```

## 1 Uvodni zadaci

---

```
22     broj kako bi se siftovanjem u desno vrsilo logicko siftovanje
    (popunjavanje nulama) a ne aritmeticko siftovanje (popunjavanje
24     znakom broja). */
    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;
        scanf("%x", &broj);
36     print_bits(broj);

38     return 0;
    }
```

### Rešenje 1.6

```
1  #include <stdio.h>

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

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

13     putchar('\n');
    }

15

17 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
    x pomeranjem broja x */
17 int count_bits(int x)
    {
19     int br = 0;
21     unsigned wl = sizeof(unsigned) * 8 - 1;

23     /* Formiramo masku 100000...0000000, koja služi za ocitavanje
        bita najveće težine. U svakoj iteraciji maska se pomera u
25     desno za 1 mesto, i ocitavamo sledeći bit. Petlja se
        završava kada više nema jedinica tj. kada maska postane
27     nula. */
        unsigned maska = 1 << wl;
29     for (; maska != 0; maska >>= 1)
        {
            x & maska ? br++ : 1;
        }
    }
```

```
31     return br;
33 }

35
37 int main()
38 {
39     int x;
40     scanf("%x", &x);
41     printf("Broj jedinica u zapisu je %d.\n", count_bits(x));
42
43     return 0;
44 }
```

```
1  #include <stdio.h>

3  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji */
5  void print_bits(int x)
6  {
7      unsigned velicina = sizeof(int) * 8;
8      unsigned maska;

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

12     putchar('\n');
13 }

15 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
   x pomeranjem broja x */
17 int count_bits1(int x)
18 {
19     int br = 0;
20     unsigned wl = sizeof(int) * 8 - 1;

22     /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1
   vrsila
   bi aritmeticko pomeranje u desno, tj. popunjavanje bita najvece
   24     tezine bitom znaka. U tom slucaju nikad ne bi bio ispunjen
   uslov x!=0 i program bi bio zarobljen u beskonacnoj petlji.
   Zbog toga se koristi pomeranj broja x ulevo i maska koja
   26     ocitava bit najvece tezine. */

28     unsigned maska = 1 << wl;
29     for (; x != 0; x <<= 1)
30         x & maska ? br++ : 1;

32     return br;
33 }

35
37
```

## 1 Uvodni zadaci

---

```
int main()
39 {
    int x;
41     scanf("%x", &x);
    printf("Broj jedinica u zapisu je %d.\n", count_bits1(x));
43
    return 0;
45 }
```

### Rešenje 1.7

```
#include <stdio.h>

2
/* Funkcija vraca najveći neoznačeni broj sastavljen iz istih
4   bitova kao i x */
unsigned najveći(unsigned x)
6 {
    unsigned velicina = sizeof(unsigned) * 8;
8
    /* Formiramo masku 100000...00000000 */
10    unsigned maska = 1 << (velicina - 1);

12    /* Inicijalizujemo rezultat na 0 */
    unsigned rezultat = 0;
14

16    /* Dokle god postoje jedinice u binarnoj reprezentaciji broja
    x (tj. dokle god je x različit od nule) pomeramo ga ulevo. */
    for (; x != 0; x <= 1) {
18        /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
        leva u rezultat */
20        if (x & maska) {
            rezultat >>= 1;
22            rezultat |= maska;
        }
24    }

26    return rezultat;
}

28
/* Funkcija vraca najmanji neoznačen broj sa istim binarnim
30   ciframa kao i x */
unsigned najmanji(unsigned x)
32 {
    /* Inicijalizujemo rezultat na 0 */
34    unsigned rezultat = 0;

36    /* Dokle god imamo jedinice u broju x, pomeramo ga udesno. */
    for (; x != 0; x >= 1) {
38        /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
        desna u rezultat */
40        if (x & 1) {
```

```

    rezultat <<= 1;
    rezultat |= 1;
}
}

return rezultat;
}

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

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

    putchar('\n');
}

int main()
{
    int broj;
    scanf("%x", &broj);

    printf("Najveci:\n");
    print_bits(najveci(broj));

    printf("Najmanji:\n");
    print_bits(najmanji(broj));

    return 0;
}

```

### Rešenje 1.8

```

1  #include <stdio.h>

3  /* Funkcija postavlja na nulu n bitova pocev od pozicije p.
   Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
   se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
   1010 1110 0111 1010 1011 1100 1101 1110 1000 0010 0111 */
7  unsigned reset(unsigned x, unsigned n, unsigned p)
   {
9      /* Cilj nam je da samo zeljene bitove anuliramo, a da ostali
      ostanu nepromenjeni. Formiramo masku koja ima n bitova
11     postavljenih na 0 pocev od pozicije p, dok su svi ostali
      postavljeni na 1.

13     Na primer, za n=5 i p=10 formiramo masku oblika 1111 1111

```

## 1 Uvodni zadaci

```
15      1111 1111 1111 1000 0011 1111 To postizemo na sledeci
      nacin: ~0 1111 1111 1111 1111 1111 1111 1111 (~0 << n)
17      1111 1111 1111 1111 1111 1111 1110 0000 ~(~0 << n) 0000
      0000 0000 0000 0000 0000 0001 1111 (~(~0 << n) << ( p-n+1))
19      0000 0000 0000 0000 0000 0111 1100 0000 ~(~(~0 << n) << (
      p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111 */
21      unsigned maska = ~(~(~0 << n) << (p - n + 1));

23      return x & maska;
25  }

27  /* Funkcija postavlja na 1 n bitova pocev od pozicije p.
      Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
      se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
29      1010 1110 0111 1010 1011 1100 1101 1110 1111 1110 0111 */
      unsigned set(unsigned x, unsigned n, unsigned p)
31  {
      /* Kako zelimo da samo odredjenih n bitova postavimo na 1, dok
33      ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
      formiramo masku oblika 0000 0000 0000 0000 0000 0111 1100
35      0000 prateci vrlo slican postupak kao za prethodnu funkciju
      */
37      unsigned maska = ~(~0 << n) << (p - n + 1);

39      return x | maska;
41  }

43  /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
      predstavlja n bitova pocev od pozicije p u binarnoj
      reprezentaciji broja x, pri cemu se pozicija broji sa desna
45      ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
      i broj 1010 1011 1100 1101 1110 1010 1110 0111 0000 0000 0000
47      0000 0000 0000 0000 1011 */
      unsigned get_bits(unsigned x, unsigned n, unsigned p)
49  {
      /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a
51      ostali su 0. Na primer za n=5 0000 0000 0000 0000 0000 0000
      0001 1111 */
53      unsigned maska = ~(~0 << n);

55      /* Pomeramo sadrzaj u desno tako da trazeno polje bude uz
      desni kraj. Zatim maskiramo ostale bitove, sem zeljenih n i
57      vracamo vrednost */
      return maska & (x >> (p - n + 1));
59  }

61  /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
      postavljeni na vrednosti n bitova najnize tezine binarne
      reprezentacije broja y */
63      unsigned set_n_bits(unsigned x, unsigned n, unsigned p,
65                          unsigned y)
```



```

67 {
68     /* Kreiramo masku kod koje su poslednjih n bitova 1, a
69        ostali su 0. Na primer za n=5 0000 0000 0000 0000 0000 0000
        0001 1111 */
70     unsigned last_n_1 = ~(~0 << n);
71
72     /* Kao ranije u funkciji reset, kreiramo masku koja ima n
73        bitova postavljenih na 0 pocevsi od pozicije p, dok su
74        ostali bitovi 1. Na primer za n=5 i p =10 1111 1111 1111
75        1111 1111 1000 0011 1111 */
76     unsigned middle_n_0 = ~(~0 << n) << (p - n + 1);
77
78     /* x sa resetovanih n bita na pozicijama pocev od p */
79     unsigned x_reset = x & middle_n_0;
80
81     /* y cijih je n bitova najnize tezine pomereni tako da stoje
82        pocev od pozicije p. Ostali bitovi su nule. (y & last_n_1)
83        resetuje sve bitove osim najnižih n */
84     unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);
85
86     return x_reset ^ y_shift_middle;
87 }
88
89
90 /* Funkcija invertuje bitove u zapisu broja x pocevsi od
91    pozicije p njih n */
92 unsigned invert(unsigned x, unsigned n, unsigned p)
93 {
94     /* Formiramo masku sa n jedinica pocev od pozicije p Na primer
95        za n=5 i p=10 0000 0000 0000 0000 0000 0111 1100 0000 */
96     unsigned maska = ~(~0 << n) << (p - n + 1);
97
98     /* Operator ekskluzivno ili invertuje sve bitove gde je
99        odgovarajuci bit maske 1. Ostali bitovi ostaju
100        nepromenjeni. */
101     return maska ^ x;
102 }
103
104
105 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
106    celog broja u memoriji */
107 void print_bits(int x)
108 {
109     unsigned velicina = sizeof(int) * 8;
110     unsigned maska;
111
112     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
113         putchar(x & maska ? '1' : '0');
114
115     putchar('\n');
116 }
117

```

## 1 Uvodni zadaci

```
119
121 int main()
122 {
123     unsigned broj, p, n, y;
124     scanf("%u%u%u%u", &broj, &n, &p, &y);
125     printf("Broj %5u %25s= ", broj, "");
126     print_bits(broj);
127
128     printf("reset(%5u,%5u,%5u)%11s = ", broj, n, p, "");
129     print_bits(reset(broj, n, p));
130
131     printf("set(%5u,%5u,%5u)%13s = ", broj, n, p, "");
132     print_bits(set(broj, n, p));
133
134     printf("get_bits(%5u,%5u,%5u)%8s = ", broj, n, p, "");
135     print_bits(get_bits(broj, n, p));
136
137     printf("y = %31u = ", y);
138     print_bits(y);
139     printf("set_n_bits(%5u,%5u,%5u,%5u) = ", broj, n, p, y);
140     print_bits(set_n_bits(broj, n, p, y));
141
142     printf("invert(%5u,%5u,%5u)%10s = ", broj, n, p, "");
143     print_bits(invert(broj, n, p));
144
145     return 0;
146 }
147
```

### Rešenje 1.9

```
#include <stdio.h>

2
/* Funkcija broj x rotira u levo za n mesta Na primer za n =5 i
4   x cija je interna reprezentacija 1010 1011 1100 1101 1110
   0001 0010 0011 0111 1001 1011 1100 0010 0100 0111 0101 */
6 unsigned rotate_left(int x, unsigned n)
7 {
8     unsigned first_bit;
9     /* Maska koja ima samo najvisi bit postavljen na 1 neophodna
10      da bismo pre siftovanja u levo za 1 sacuvali najvisi bit. */
11     unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
12     int i;

13
14     /* n puta vrsimo rotaciju za jedan bit u levo. U svakoj
15      iteraciji odredimo prvi bit, a potom pomeramo sadrzaj broja
16      x u levo za 1 i potom najnizi bit postavljamo na vrednost
17      koju je imao prvi bit koji smo istisnuli siftovanjem */
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 return x;
23 }
24
25 /* Funkcija neoznaceni broj x rotira u desno za n Na primer za n
26    =5 i x cija je interna reprezentacija 1010 1011 1100 1101
27    1110 0001 0010 0011 0001 1101 0101 1110 0110 1111 0000 1001 */
28 unsigned rotate_right(unsigned x, unsigned n)
29 {
30     unsigned last_bit;
31     int i;
32
33     /* n puta ponavljamo rotaciju u desno za jedan bit. U svakoj
34        iteraciji odredjujemo bit najmanje tezine broja x, zatm
35        tako odredjeni bit siftujemo u levo tako da najnizi bit
36        dođe do pozicije najviseg bita i nakon siftovanja x za 1 u
37        desno postavljamo x-ov najvisi bit na vrednost najnižeg
38        bita. */
39     for (i = 0; i < n; i++) {
40         last_bit = x & 1;
41         x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
42     }
43
44     return x;
45 }
46
47 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde
48    je x oznaceni broj */
49 int rotate_right_signed(int x, unsigned n)
50 {
51     unsigned last_bit;
52     int i;
53
54     /* U svakoj iteraciji odredjujemo bit najmanje tezine tj.
55        last_bit. Kako je x oznacen ceo broj, tada se prilikom
56        siftovanja u desno vrši aritmeticki sift i cuva se znak
57        broja. Iza tog razloga imamo dva slucaja u zavisnosti od
58        znaka od x. Nije dovoljno da se ova provera izvrši pre
59        petlje, jer rotiranjem u desno na mesto najviseg bita može
60        doći i 0 i 1, nezavisno od pocetnog znaka x. */
61     for (i = 0; i < n; i++) {
62         last_bit = x & 1;
63
64         if (x < 0)
65             /* Siftovanjem u desno broja koji je negativan dobijamo 1
66                na najvisoj poziciji. Na primer ako je x 1010 1011
67                1100 1101 1110 0001 0010 001b (sa b oznacavamo u
68                primeru 1 ili 0 na najnižoj poziciji) last_bit je 0000
69                0000 0000 0000 0000 0000 000b nakon Siftovanja za
70                1 u desno 1101 0101 1110 0110 1111 0000 1001 0001 da

```

## 1 Uvodni zadaci

---

```
72         bismo najvisu 1 u x postavili na b nije dovoljno da ga
73         siftujemo na najvisu poziciju jer bi se time dobile 0,
74         a nama su potrebne 1 zbog bitovskog & zato prvo
75         komplementiramo, pa tek onda siftujemo ~last_bit <<
76         (sizeof(int)*8 -1) B000 0000 0000 0000 0000 0000 0000
77         0000 (B oznacava ~b ) i ponovo komplementiramo da bismo
78         imali b na najvisoj poziciji i sve 1 na ostalim
79         pozicijama ~(~last_bit << (sizeof(int)*8 -1)) b111 1111
80         1111 1111 1111 1111 1111 1111 */
81         x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
82     else
83         x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
84     }
85
86     return x;
87 }
88
89 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
90    celog broja u memoriji */
91 void print_bits(int x)
92 {
93     unsigned velicina = sizeof(int) * 8;
94     unsigned maska;
95     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
96         putchar(x & maska ? '1' : '0');
97
98     putchar('\n');
99 }
100
101 int main()
102 {
103     unsigned x, k;
104     scanf("%x%x", &x, &k);
105     printf("x %36s = ", "");
106     print_bits(x);
107     printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
108     print_bits(rotate_left(x, k));
109
110     printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
111     print_bits(rotate_right(x, k));
112
113     printf("rotate_right_signed(%7u,%6u) = ", x, k);
114     print_bits(rotate_right_signed(x, k));
115
116     return 0;
117 }
```

### Rešenje 1.10

---

```

1  #include <stdio.h>
2
3  /* Funkcija vraća vrednost čija je binarna reprezentacija slika
4     u ogledalu binarne reprezentacije broja x. Na primer za x
5     čija binarna reprezentacija izgleda ovako
6     101010111100110111100100100100011 funkcija treba da vrati
7     broj čija binarna reprezentacija izgleda:
8     11000100100001111011001111010101 */
9  unsigned mirror(unsigned x)
10 {
11     unsigned najnizi_bit;
12     unsigned rezultat = 0;
13
14     int i;
15     /* Krecemo od najnižeg bita u zapisu broja x i dodajemo ga u
16        rezultat */
17     for (i = 0; i < sizeof(x) * 8; i++) {
18         najnizi_bit = x & 1;
19         x >>= 1;
20         /* Potiskujemo trenutni rezultat ka levom kraju. Tako svi
21            prethodno postavljeni bitovi dobijaju veću poziciju. Novi
22            bit postavljamo na najnižu poziciju */
23         rezultat <<= 1;
24         rezultat |= najnizi_bit;
25     }
26     return rezultat;
27 }
28
29 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
30    celog broja u memoriji */
31 void print_bits(int x)
32 {
33     unsigned velicina = sizeof(int) * 8;
34     unsigned maska;
35     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
36         putchar(x & maska ? '1' : '0');
37
38     putchar('\n');
39 }
40
41 int main()
42 {
43     int broj;
44     scanf("%x", &broj);
45
46     /* Ispisujemo binarnu reprezentaciju unetog broja */
47     print_bits(broj);
48
49     /* Ispisujemo binarnu reprezentaciju broja dobijenog pozivom
50        funkcije mirror */
51     print_bits(mirror(broj));
52 }

```

```
54     return 0;
    }
```

### Rešenje 1.11

```
#include <stdio.h>

2
/* Funkcija vraća 1 ukoliko je u binarnoj reprezentaciji broja n
4   broj jedinica veći od broja nula. U suprotnom funkcija vraća
   0 */
6 int Broj01(unsigned int n)
{
8
   int broj_nula, broj_jedinica;
10  unsigned int maska;

12  broj_nula = 0;
   broj_jedinica = 0;
14
   /* Postavljamo masku tako da počinjemo sa analiziranjem bita
16     najveće težine */
   maska = 1 << (sizeof(unsigned int) * 4 - 1);
18
   /* Dok ne obidjemo sve bitove u zapisu broj n */
20  while (maska != 0) {

22     /* Proveravamo da li se na poziciji koju određuje maska
       nalazi 0 ili 1 i uvećavamo odgovarajući brojac */
24     if (n & maska) {
       broj_jedinica++;
26     } else {
       broj_nula++;
28     }

30     /* Pomeramo masku u desnu stranu tako da možemo da očitamo
       vrednost narednog bita */
32     maska = maska >> 1;
   }

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

40
int main()
42 {
   unsigned int n;
44
   /* Učitavamo broj */
```

```
46 scanf("%u", &n);
48 /* Ispisujemo vrednost funkcije */
   printf("%d\n", Broj01(n));
50
   return 0;
52 }
```

### Rešenje 1.12

```
#include <stdio.h>
2
int broj_parova(unsigned int x)
4 {
6     int broj_parova;
   unsigned int maska;
8
   /* Postavljamo broj parova na 0 */
10   broj_parova = 0;
12
   /* Postavljamo masku tako da mozemo da procitamo da li su dva
      najmanja bita u zapisu broja x 11 */
14   /* broj 3 je binarno 000....00011 */
   maska = 3;
16
   /* Dok ne obidjemo sve parove bitova u zapisu broja x */
18   while (x != 0) {
20       /* Proveravamo da li se na najmanjim pozicijama broj x
          nalazi 11 par */
22       if ((x & maska) == maska) {
           broj_parova++;
24       }
26
       /* Pomeramo broj u desnu stranu tako da mozemo da očitamo
          vrednost sledeceg para bitova. Pomeranjem u desno
          bit najvece tezine se popunjava nulom jer je x
          neoznacen broj. */
30       x = x >> 1;
   }
32
   return broj_parova;
34 }
36
int main()
38 {
   unsigned int x;
40
   /* Ucitavamo broj */
```

## 1 Uvodni zadaci

---

```
42 | scanf("%u", &x);  
  
44 | /* Ispisujemo vrednost funkcije */  
    | printf("%d\n", broj_parova(x));  
  
46 | return 0;  
48 | }
```

### Rešenje 1.13

### Rešenje 1.14

```
#include <stdio.h>  
  
2 |  
/*  
4 | Niska koju formiramo je duzine (sizeof(unsigned int)*8)/4 +1  
   | jer za svaku heksadekadnu cifru nam trebaju 4 binarne cifre i  
6 | jedna dodatna pozicija nam treba za terminirajucu nulu.  
  
8 | Prethodni izraz je identican sa sizeof(unsigned int)*2+1.  
  
10 | Na primer, ako je duzina unsigned int 4 bajta onda je  
    | MAX_DUZINA 9 */  
  
12 | #define MAX_DUZINA sizeof(unsigned int)*2 +1  
14 |  
  
16 | void prevod(unsigned int x, char s[])  
    | {  
18 |     int i;  
20 |     unsigned int maska;  
    |     int vrednost;  
22 |  
    | /* Heksadekadni zapis broja 15 je 000...0001111 - ovo nam  
24 |    odgovara ako hocemo da citamo 4 uzastopne cifre */  
    | maska = 15;  
26 |  
    | /*  
28 |    Broj cemo citati od pozicije najmanje tezine ka poziciji  
    |    najvece tezine; npr. za broj  
30 |    0000000001101000100001111010101 u prvom koraku cemo  
    |    procitati bitove: 000000000110100010000111101<0101>  
32 |    (bitove izdvojene sa <...>) u drugom koraku cemo procitati:  
    |    00000000011010001000011<1101>0101 u trecem koraku cemo  
34 |    procitati: 0000000001101000100<0011>11010101 i tako redom  
  
36 |    indeks i oznacava poziciju na koju smestamo vrednost  
  
38 |    */
```



```

40  for (i = MAX_DUZINA - 2; i >= 0; i--) {
41      /* Vrednost izdvojene cifre */
42      vrednost = x & maska;
43
44      /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter
45      dobijamo dodavanjem ASCII koda '0' Ako je vrednost iz
46      opsega od 10 do 15 odgovarajuci karakter dobijamo tako
47      sto prvo oduzmemo 10 (dobijamo vrednosti od 0 do 5) pa
48      dodamo ASCII kod 'A' (time dobijamo slova 'A', 'B', ...
49      'F') */
50      if (vrednost < 10) {
51          s[i] = vrednost + '0';
52      } else {
53          s[i] = vrednost - 10 + 'A';
54      }
55
56      /* Broj pomeramo za 4 bita u desnu stranu tako da mozemo da
57      procitamo sledecu cifru */
58      x = x >> 4;
59  }
60
61  s[MAX_DUZINA - 1] = '\0';
62  }
63
64  int main()
65  {
66      unsigned int x;
67      char s[MAX_DUZINA];
68
69      /* Ucitavamo broj */
70      scanf("%u", &x);
71
72      /* Pozivamo funkciju */
73      prevod(x, s);
74
75      /* Ispisujemo dobijenu nisku */
76      printf("%s\n", s);
77
78      return 0;
79  }

```

### Rešenje 1.17

```

1  #include <stdio.h>
2
3  /* Iskomentarisan je deo koji se ispisuje svaki put kad se udje
4  u funkciju. Odkomentarisati pozive printf funkcije u obe
5  funkcije da uocite razliku u broju rekursivnih poziva obe
6  verzije. */

```

## 1 Uvodni zadaci

---

```
8  /* Linearno resenje se zasniva na cinjenici:  $x^0 = 1$   $x^k = x * x^{(k-1)}$  */
10 int stepen(int x, int k)
11 {
12     // printf("Racunam stepen (%d, %d)\n", x, k);
13     if (k == 0)
14         return 1;
15
16     return x * stepen(x, k - 1);
17
18     /* Celo telo funkcije se moze ovako kratko zapisati return k
19        == 0 ? 1 : x * stepen(x,k-1); */
20 }
21
22 /* Druga verzija prethodne funkcije. Obratiti paznju na
23    efikasnost u odnosu na prvu verziju! */
24
25 /* Logaritamsko resenje je zasnovano na cinjenicama: -  $x^0 = 1$ ; -
26     $x^k = x * (x^2)^{(k/2)}$ , za neparno k -  $x^k = (x^2)^{(k/2)}$ ,
27    za parno k
28
29    Ovom resenju ce biti potrebno manje rekursivnih poziva da bi
30    doslo do rezultata, i stoga je efikasnije. */
31 int stepen2(int x, int k)
32 {
33     // printf("Racunam stepen2 (%d, %d)\n",x,k);
34     if (k == 0)
35         return 1;
36
37     /* Ako je stepen paran */
38     if ((k % 2) == 0)
39         return stepen2(x * x, k / 2);
40     /* Inace (ukoliko je stepen neparan) */
41     return x * stepen2(x * x, k / 2);
42 }
43
44 int main()
45 {
46     int x, k;
47     scanf("%d%d", &x, &k);
48
49     printf("%d", stepen(2, 10));
50     // printf("\n-----\n");
51     // printf("%d\n",stepen2(2,10));
52     return 0;
53 }
54 }
```

### Rešenje 1.18

---

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 100
5
   /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
7    (posrednu) rekurziju. */

9   /* Deklaracija funkcije neparan mora da bude navedena jer se ta
   funkcija koristi u telu funkcije paran, tj. koristi se pre
11  svoje definicije. Funkcija je mogla biti deklarirana i u telu
   funkcije paran. */
13
   unsigned neparan(unsigned n);
15
   /* Funkcija vraca 1 ako broj n ima paran broj cifara inace
17    vraca 0. */
   unsigned paran(unsigned n)
19 {
   if (n >= 0 && n <= 9)
21     return 0;
   else
23     return neparan(n / 10);
   }
25
   /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace
27    vraca 0. */
   unsigned neparan(unsigned n)
29 {
   if (n >= 0 && n <= 9)
31     return 1;
   else
33     return paran(n / 10);
   }
35
   /* Glavna funkcija za testiranje */
37 int main()
   {
39     int n;
     scanf("%d", &n);
41     printf("Uneti broj ima %s paran broj cifara\n",
           (paran(n) == 1 ? "" : "ne"));
43     return 0;
   }

```

### Rešenje 1.19

```

   #include <stdio.h>
2   /* Repno-rekurzivna (eng. tail recursive) je ona funkcija
   Cije se telo završava rekurzivnim pozivom, pri čemu taj
4   rekurzivni poziv ne učestvuje u nekom izrazu.

```

```
6      Kod ovih funkcija se po završetku za tekuci rekurzivni
8      poziv umesto skoka na adresu povratka skace na adresu
      povratka za prethodni poziv, odnosno za poziv na manjoj
      dubini. Time se stedi i prostor i vreme.

10     Ovakve funkcije se mogu lako zameniti odgovarajucom
12     iterativnom funkcijom, cime se smanjuje prostorna
      slozenost algoritma. */
14     /* Pomoćna funkcija koja izracunava n! * result. Koristi
      repnu rekurziju. */
16     /* Result je argument u kom cemo akumulirati do tada
      izracunatu vrednost faktoriijela. Kada završimo, tj. kada
18     dodjemo do izlaza iz rekurzije potrebno je da vratimo
      result. */
20     int faktoriijelRepna(int n, int result)
      {
22         if (n == 0)
            return result;

24         return faktoriijelRepna(n - 1, n * result);
26     }

28     /* Sada zelimo da se oslobodimo repne rekurzije koja postoji u
      funkciji faktoriijelRepna, koristeći algoritam sa predavanja.

30     Najpre cemo vrednost argumenta funkcije postaviti na vrednost
32     koja bi se prosledjivala rekurzivnom pozivu i pomocu goto
      naredbe vratiti se na pocetak tela funkcije. */

34     int faktoriijelRepna_v1(int n, int result)
      {
36         pocetak:
38         if (n == 0)
            return result;

40         result = n * result;
42         n = n - 1;
            goto pocetak;
44     }

46     /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra
      programerska praksa. Iskoristicemo prethodni medjukorak da
48     bismo dobili iterativno resenje bez bezuslovnih skokova. */
      int faktoriijelRepna_v2(int n, int result)
      {
50         while (n != 0) {
52             result = n * result;
            n = n - 1;
54         }

56         return result;
```

```

58 }
/* Nasim gore navedenim funkcijama pored n, mora da se salje i 1
60 za vrednost drugog argumenta u kome ce se akumulirati
rezultat. Funkcija faktorijel(n) je ovde radi udobnosti
62 korisnika, jer je sasvim prirodno da za faktorijel zahteva
samo 1 parametar. Funkcija faktorijel izracunava n!, tako Sto
64 odgovarajucoj gore navedenoj funkciji koja zaista racuna
faktorijel, salje ispravne argumente i vraca rezultat koju
66 joj ta funkcija vrati. Za testiranje, zameniti u telu
funkcije faktorijel poziv faktorijelRepna sa pozivom
68 faktorijelRepna_v1, a zatim sa pozivom funkcije
faktorijelRepna_v2. */
70 int faktorijel(int n)
{
72     return faktorijelRepna(n, 1);
}
74
/* Test program */
76 int main()
{
78     int n;

80     printf("Unesite n (<= 12): ");
scanf("%d", &n);
82     printf("%d! = %d\n", n, faktorijel(n));

84     return 0;
}

```

## Rešenje 1.20

## Rešenje 1.21

## Rešenje 1.22

```

#include <stdio.h>
2  #define MAX_DIM 1000

4  /*
    Ako je n==0, onda je suma(a,0) = 0 Ako je n>0, onda je
6  suma(a,n) = a[n-1] + suma(a,n-1) Suma celog niza je jednaka
sumi prvih n-1 elementa uvecenoj za poslednji element celog
8  niza. */
int sumaNiza(int *a, int n)
10 {
    /* Ne stavljamo strogu jednakost n==0, za slucaj da korisnik
12    prilikom prvog poziva, posalje negativan broj za velicinu
niza. */

```

## 1 Uvodni zadaci

---

```
14     if (n <= 0)
15         return 0;
16
17     return a[n - 1] + sumaNiza(a, n - 1);
18 }
19
20 /*
21     n==0, suma(a,0) = 0 n >0, suma(a,n) = a[0]+suma(a+1,n-1) Suma
22     celog niza je jednaka zbiru prvog elementa niza i sume
23     preostalih n-1 elementa. */
24 int sumaNiza2(int *a, int n)
25 {
26     if (n <= 0)
27         return 0;
28
29     return a[0] + sumaNiza2(a + 1, n - 1);
30 }
31
32 int main()
33 {
34     int a[MAX_DIM];
35     int n, i = 0;
36
37     /* Ucitavamo broj elemenata niza */
38     scanf("%d", &n);
39
40     /* Ucitavamo n elemenata niza. */
41     for (i = 0; i < n; i++)
42         scanf("%d", &a[i]);
43
44     printf("Suma elemenata je %d\n", sumaNiza(a, n));
45     // printf("Suma elemenata je %d\n", sumaNiza2(a, n));
46
47     return 0;
48 }
```

### Rešenje 1.23

```
1 #include <stdio.h>
2 #define MAX_DIM 256
3
4 /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza
5    niz dimenzije n */
6 int maksimum_niza(int niz[], int n)
7 {
8     /* Izlazak iz rekurziije: ako je niz dimenzije jedan, najveći
9        je ujedno i jedini element niza */
10    if (n == 1)
11        return niz[0];
12
13    /* Rešavamo problem manje dimenzije */
```

```

14     int max = maksimum_niza(niz, n - 1);

16     /* Ako nam je poznato resenje problema dimenzije n-1, resavamo
       problem dimenzije n */
18     return niz[n - 1] > max ? niz[n - 1] : max;
19 }

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

25     /* Sve dok ne dodjemo do kraja ulaza, ucitavamo brojeve u niz;
       i predstavlja indeks tekućeg broja. */
26     int i = 0;
27     while (scanf("%d", &brojevi[i]) != EOF) {
28         i++;
29     }
30     n = i;

31     /* Stampamo maksimum unetog niza brojeva */
32     printf("%d\n", maksimum_niza(brojevi, n));
33     return 0;
34 }

```

### Rešenje 1.24

```

#include <stdio.h>
#define MAX_DIM 256

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

    /* Na osnovu rešenja problema dimenzije n-1, resavamo problem
       dimenzije n */
    else
        return a[n - 1] * b[n - 1] + skalarno(a, b, n - 1);
}

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

    /* Unosimo dimenziju nizova, */
    scanf("%d", &n);

    /* a zatim i same nizove. */
    for (i = 0; i < n; i++)

```

## 1 Uvodni zadaci

---

```
        scanf("%d", &a[i]);
26
    for (i = 0; i < n; i++)
28        scanf("%d", &b[i]);

30    /* Ispisujemo rezultat skalarnog proizvoda dva učitana niza. */
    printf("%d\n", skalarno(a, b, n));

32    return 0;
34 }
```

### Rešenje 1.25

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

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

10    int bp = br_pojave(x, a, n - 1);
    return a[n - 1] == x ? 1 + bp : bp;
12 }

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

18
20     /* Učitavamo broj koji se traži */
    scanf("%d", &x);

22     /* Sve dok ne dodjemo do kraja ulaza, učitavamo brojeve u niz;
        i predstavlja indeks tekućeg broja */
24     i = 0;
    while (scanf("%d", &a[i]) != EOF) {
26         i++;
    }

28     n = i;

30     /* Ispisujemo broj pojave broja x u niz a */
    printf("%d\n", br_pojave(x, a, i));
32    return 0;
}
```

### Rešenje 1.26



```

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

4 int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
{
6     /* Ako niz ima manje od tri elementa izlazimo iz rekurzije */
    if (n < 3)
8         return 0;

10     else
        return ((a[n - 3] == x) && (a[n - 2] == y)
12             && (a[n - 1] == z))
            || tri_uzastopna_clana(x, y, z, a, n - 1);
14 }

16 int main()
{
18     int x, y, z, a[MAX_DIM];
    int n;

20     /* Ucitavaju se tri cela broja za koje se ispituje da li su
22         uzastopni clanovi niza */
    scanf("%d%d%d", &x, &y, &z);

24     /* Sve dok ne dodjemo do kraja ulaza, ucitavamo brojeve u niz */
    int i = 0;
    while (scanf("%d", &a[i]) != EOF) {
26         i++;
    }
30     n = i;

32     if (tri_uzastopna_clana(x, y, z, a, i))
        printf("da\n");
34     else
        printf("ne\n");

36     return 0;
38 }

```

### Rešenje 1.27

```

#include <stdio.h>
2

4 /* funkcija koja broji bitove svog argumenta */
/*
6     ako je x ==0, onda je count(x) = 0 inace count(x) =
    najvisi_bit +count(x<<1)

8     Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
    siftovanjem sa desne strane uvek dopisuju 0, argument x ce u

```

## 1 Uvodni zadaci

```
10     nekom rekurzivnom pozivu biti bas 0 i izacicemo iz rekurzije. */
12 int count(int x)
13 {
14     /* izlaz iz rekurzije */
15     if (x == 0)
16         return 0;
17
18     /* Dakle, neki bit je postavljen na 1. */
19     /* Proveravamo vrednost najviseg bita Kako za rekurzivni poziv
20        moramo slati siftovano x i x je oznacen ceo broj, onda ne
21        smemo koristiti siftovanje desno, jer funkciji moze biti
22        prosleden i negativan broj. Iz tog razloga, odlucujemo se
23        da proveramo najvisi, umesto najnizeg bita */
24     if (x & (1 << (sizeof(x) * 8 - 1)))
25         return 1 + count(x << 1);
26     /* Najvisi bit je 1. Sacekacemo da zavrshi poziv koji racuna
27        koliko ima jedinica u ostatku binarnog zapisa x i potom
28        uvecati taj rezultat za 1. */
29     else
30         /* Najvisi bit je 0. Stoga je broj jedinica u zapisu x isti
31            kao broj jedinica u zapisu broja x<<1, jer se siftovanjem
32            u levo sa desne stane dopisuju 0. */
33         return count(x << 1);
34
35     /* jednolinijska return naredba sa proverom i rekurzivnim
36        pozivom return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) +
37        count(x<<1); */
38 }
39
40 int main()
41 {
42     int x;
43     scanf("%x", &x);
44     printf("%d\n", count(x));
45
46     return 0;
47 }
```

### Rešenje 1.29

```
#include<stdio.h>
2
3 /* Rekurzivna funkcija za odredjivanje najvece heksadekadne
4    cifre u broju */
5 int max_oktalna_cifra(unsigned x)
6 {
7     /* izlazak iz rekurzije */
8     if (x == 0)
9         return 0;
10    /* Odredjivanje poslednje heksadekadne cifre u broju */
```

```

12     int poslednja_cifra = x & 7;
    /* Odredjivanje maksimalne oktalne cifre u broju kada se iz
       njega izbrise poslednja oktalna cifra */
14     int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
    return poslednja_cifra >
16         max_bez_poslednje_cifre ? poslednja_cifra :
           max_bez_poslednje_cifre;
18 }

20 int main()
21 {
22     unsigned x;
23     scanf("%u", &x);
24     printf("%d\n", max_oktalna_cifra(x));
25     return 0;
26 }

```

### Rešenje 1.30

```

#include<stdio.h>

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

20 int main()
21 {
22     unsigned x;
23     scanf("%u", &x);
24     printf("%d\n", max_heksadekadna_cifra(x));
25     return 0;
26 }

```

### Rešenje 1.31

```
1  #include<stdio.h>
2  #include<string.h>
3  /* niska moze imati najviše 32 karaktera + 1 za terminalnu nulu */
4  #define MAX_DIM 33
5
6  int palindrom(char s[], int n)
7  {
8      if ((n == 1) || (n == 0))
9          return 1;
10     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
11 }
12
13 int main()
14 {
15     char s[MAX_DIM];
16     int n;
17
18     /* Ucitavamo nisku sa ulaza */
19     scanf("%s", s);
20
21     /* Odredjujemo duzinu niske */
22     n = strlen(s);
23
24     /* Ispisujemo na izlazu poruku da li je niska palindrom ili
25        nije */
26     if (palindrom(s, n))
27         printf("da\n");
28     else
29         printf("ne\n");
30
31     return 0;
32 }
```

### Rešenje 1.32

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_DUZINA_NIZA 50
4
5  void ispisiNiz(int a[], int n)
6  {
7      int i;
8
9      for (i = 1; i <= n; i++)
10         printf("%d ", a[i]);
11     printf("\n");
12 }
13
14 /* Funkcija proverava da li se x vec nalazi u permutaciji na
15    prethodnih 1...n mesta */
```

```

17 int koriscen(int a[], int n, int x)
18 {
19     int i;
20     for (i = 1; i <= n; i++)
21         if (a[i] == x)
22             return 1;
23     return 0;
24 }
25
26 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[]
27    je niz u koji smesta permutacije m - oznacava da se na m-tu
28    poziciju u permutaciji smesta jedan od preostalih celih
29    brojeva n- je velicina skupa koji se permutuje Funkciju
30    pozivamo sa argumentom m=1 jer krecemo da formiramo
31    permutaciju od 1. pozicije i nikada ne koristimo a[0]. */
32 void permutacija(int a[], int m, int n)
33 {
34     int i;
35
36     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti
37        broj premasila velicinu skupa, onda se svi brojevi vec
38        nalaze u permutaciji i ispisujemo permutaciju. */
39     if (m > n) {
40         ispisiNiz(a, n);
41         return;
42     }
43
44     /* Ideja: pronalazimo prvi broj koji mozemo da postavimo na
45        m-to mesto u nizu (broj koji se do sada nije pojavio u
46        permutaciji). Zatim, rekurzivno pronalazimo one permutacije
47        koje odgovaraju ovako postavljenom pocetku permutacije.
48        Kada to završimo, proveravamo da li postoji jos neki broj
49        koji moze da se stavi na m-to mesto u nizu (to se radi u
50        petlji). Ako ne postoji, funkcija je završila sa radom.
51        Ukoliko takav broj postoji, onda ponovo pozivamo rekurzivno
52        pronalazenje odgovarajucih permutacija, ali sada sa
53        drugacije postavljenim prefiksom. */
54
55     for (i = 1; i <= n; i++) {
56         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do
57            m-1 pozicije, onda ga stavljamo na poziciju m i pozivamo
58            funkciju da napravi permutaciju za jedan vece duzine, tj.
59            m+1. Inace, nastavljamo dalje, trazeci broj koji se nije
60            pojavio do sada u permutaciji. */
61         if (!koriscen(a, m - 1, i)) {
62             a[m] = i;
63             /* Pozivamo ponovo funkciju da dopuni ostatak permutacije
64                posle upisivanja i na poziciju m. */
65             permutacija(a, m + 1, n);
66         }
67     }

```

```
    }
69 }

71 int main(void)
72 {
73     int n;
74     int a[MAX_DUZINA_NIZA];

75     printf("Unesite duzinu permutacije: ");
76     scanf("%d", &n);
77     if (n < 0 || n >= MAX_DUZINA_NIZA) {
78         fprintf(stderr,
79             "Duzina permutacije mora biti broj veci od 0 i manji od %
80             d!\n",
81             MAX_DUZINA_NIZA);
82         exit(EXIT_FAILURE);
83     }

84     permutacija(a, 1, n);

85     exit(EXIT_SUCCESS);
86 }
87 }
```

### Rešenje 1.33

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Rekurzivna funkcija za racunanje binomnog koeficijenta. */
5  /* ako je k=0 ili k=n, onda je binomni koeficijent 0 ako je k
6     izmedju 0 i n, onda je bk(n,k) = bk(n-1,k-1) + bk(n-1,k) */
7  int binomniKoeficijent(int n, int k)
8  {
9      return (0 < k
10             && k < n) ? binomniKoeficijent(n - 1,
11                                     k - 1) +
12             binomniKoeficijent(n - 1, k) : 1;
13 }

14 /* Iterativno izracunavanje datog binomnog koeficijenta.

15
16
17     int binomniKoeficijent (int n, int k) { int i, j, b; for
18         (b=i=1, j=n; i<=k; b=b*j--/i++); return b; }
19
20 */
21
22 /* Prostim opažanjem se uocava da se svaki element n-te
23     hipotenuze (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1
24     hipotenuze. Uz pomenute dve nove ivicne jedinice lako se
25     zakljucuje da ce suma elementa n-te hipotenuze biti tacno 2
26     puta veca. */
```

```
27 int sumaElemenataHipotenuze(int n)
28 {
29     return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
30 }
31
32 int main()
33 {
34     int n, k, i, d;
35
36     scanf("%d %d", &d, &n);
37
38     /* Ispisivanje Paskalovog trougla */
39     putchar('\n');
40     for (n = 0; n <= d; n++) {
41         for (i = 0; i < d - n; i++)
42             printf(" ");
43         for (k = 0; k <= n; k++)
44             printf("%4d", binomniKoeficijent(n, k));
45         putchar('\n');
46     }
47
48     if (n < 0) {
49         fprintf(stderr,
50             "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
51         );
52         exit(EXIT_FAILURE);
53     }
54     printf("%d\n", sumaElemenataHipotenuze(n));
55     exit(EXIT_SUCCESS);
56 }
57 }
```





## Glava 2

# Pokazivači

### 2.1 Pokazivačka aritmetika

**Zadatak 2.1** Milen: ovako definisan zadatak zahteva dva programa kao resenja, a ne jedan sa definisane dve funkcije. Za dati celobrojni niz dimenzije  $n$ , napisati funkciju koja obrće njegove elemente:

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

Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju niza  $n$  ( $0 < n \leq 100$ ), a zatim elemente niza. Prikazati sadržaj niza posle poziva funkcije za obrtanje elemenata niza.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  3         1 -2 3    Izlaz: 3 -2 1</pre>	<pre>   Ulaz:  0         Izlaz: Greska: neodgovarajuca dimenzija niza.</pre>

[Rešenje 2.1]

**Zadatak 2.2** Dat je niz realnih brojeva dimenzije  $n$ .

- (a) Napisati funkciju `zbir` koja izračunava zbir elemenata niza.
- (b) Napisati funkciju `proizvod` koja izračunava proizvod elemenata niza.
- (c) Napisati funkciju `min_element` koja izračunava najmanji elemenat niza.

(d) Napisati funkciju `max_element` koja izračunava najveći elemenat niza.

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

```
Test 1
|| Ulaz: 3
||      -1.1 2.2 3.3
|| Izlaz: zbir = 4.400
||         proizvod = -7.986
||         min = -1.100
||         max = 3.300
```

```
Test 2
|| Ulaz: 5
||      1.2 3.4 0.0 -5.4 2.1
|| Izlaz: zbir = 1.300
||         proizvod = -0.000
||         min = -5.400
||         max = 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. **Jelena:** Sta kazete na to da prekoracenja dimenzije niza u razlicitim zadacima razlicito obradjujemo. Na primer, mozemo da unosimo dimenziju niza sve dok se ne unese broj koji je u odgovarajucem opsegu, ili mozemo da dimenziju postavimo na 1 ako je korisnik uneo broj manji od 1, a na MAX ako je korisnik uneo broj veci od MAX, itd? **Nina:** Slazem se da treba da vide razlicite nacine, uprkos tome sto bi unifomnost bila lepsa

```
Test 1
|| Ulaz: 5
||      1 2 3 4 5
|| Izlaz: 2 3 3 3 4
```

```
Test 2
|| Ulaz: 4
||      4 -3 2 -1
|| Izlaz: 5 -2 1 -2
```

```
Test 3
|| Ulaz: 0
|| Izlaz: Greska: neodgovarajuca dimenzija niza.
```

```
Test 4
|| Ulaz: 101
|| Izlaz: Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.3]

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

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

Jelena: Da li je ok da ovaj zadatak pod a i b resim na nacin na koji sam resila, odnosno, da jedno od ta dva resenja iskomentarisem? Milena: Meni se cini da je bolje bez komentarisanja, vec da su oba prisutna. Nina: Nisam odvajala u dva zadatka - za sada ispisuje dva puta izlaz...

<p><i>Test 1</i></p> <pre> Poziv: ./a.out prvi 2. treci -4 Izlaz: 5       0 ./a.out       1 prvi       2 2.       3 treci       4 -4       . p 2 - </pre>	<p><i>Test 2</i></p> <pre> Poziv: ./a.out Izlaz: 1       0 ./a.out       . </pre>
---	---

[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.

<p><i>Test 1</i></p> <pre> Poziv: ./a.out programiranje anavolimilovana topot ana anagram t Izlaz: 4 </pre>	<p><i>Test 2</i></p> <pre> Poziv: ./a.out a b 11 212 Izlaz: 4 </pre>	<p><i>Test 3</i></p> <pre> Poziv: ./a.out Izlaz: 0 </pre>
---	--	---

[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.

### Test 1

```
Poziv: ./a.out ulaz.txt 1
ulaz.txt: Ovo je sadržaj datoteke i u njoj ima reci koje imaju
          1 karakter
Izlaz: 3
```

### Test 2

```
Poziv: ./a.out ulaz.txt
Izlaz: Greska: Nedovoljan broj argumenata komandne linije.
       Program se poziva sa ./a.out ime_dat br_karaktera.
```

### Test 3

```
Poziv: ./a.out ulaz.txt 2
(ne postoji datoteka ulaz.txt)
Izlaz: 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 задаje 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.

Milena: Umesto komentara `-Funkcija strepy iz standardne biblioteke-` i ostalih sličnih, napisati `-Implementacije funkcije strepy iz standardne biblioteke-`

### Test 1

```
Poziv: ./a.out ulaz.txt ke -s
ulaz.txt: Ovo je sadržaj datoteke i u njoj ima reci koje se
          završavaju na ke
Izlaz: 2
```

### Test 2

```
Poziv: ./a.out ulaz.txt sa -p
ulaz.txt: Ovo je sadržaj datoteke i u njoj ima reci koje
          pocinju sa sa
Izlaz: 3
```

### Test 3

```
Poziv: ./a.out ulaz.txt sa -p
(ne postoji datoteka ulaz.txt)
Izlaz: Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

*Test 3*

```
|| Poziv: ./a.out ulaz.txt
|| Izlaz: 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  $n$ .

- (a) Napisati funkciju koja izračunava trag matrice (sumu elemenata na glavnoj dijagonali).
- (b) 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.

*Test 1*

```
|| Ulaz:  3 1 -2 3 4 -5 6 7 -8 9
|| Izlaz: 1 -2 3
||        4 -5 6
||        7 -8 9
||        trag = 5
||        euklidska norma = 16.88
||        vandijagonalna norma = 11
```

*Test 2*

```
|| Ulaz:  0
|| Izlaz: Greska: neodgovarajuca dimenzija matrice.
```

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

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

- (c) Napisati funkciju koja izračunava proizvod matrica.

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

```
Test 1
|
| Ulaz: 3
|       1 2 3 1 2 3 1 2 3
|       1 2 3 1 2 3 1 2 3
|
| Izlaz: da
|       Zbir matrica je:
|       2 4 6
|       2 4 6
|       2 4 6
|       Proizvod matrica je:
|       6 12 18
|       6 12 18
|       6 12 18
```

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

- (a) Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- (b) Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- (c) Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.
- (d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).
- (e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).
- (f) Napisati funkciju koja određuje refleksivno-tranzitivno zatvorenje relacije (najmanju refleksivnu i tranzitivnu relaciju 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.

### Test 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
Izlaz:    Refleksivnost: ne
          Simetricnost: ne
          Tranzitivnost: da
          Refleksivno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 1
          Simetricno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 1 1 0
          0 0 0 0
          Refleksivno-tranzitivno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 0
```

**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 najveći element matrice na sporednoj dijagonali, indeks kolone koja sadrži najmanji element, indeks vrste koja sadrži najveći element i broj negativnih elemenata učitane matrice.

Milena: Izbegavala bih komentare koji ulaze u kod i na taj način narušavaju citljivost koda, kao što je to npr u funkciji `indeks_min` i `indeks_max`. Resenje 2.15 - izbacila bih napomenu iz komentara. Slicno mi se čini i za zadatak 2.17.

Zadatak 2.17 - cini mi se da je resenje bez koriscnja biblioteckih funkcija visak?  
Zadatak 2.19 — izvuci komentare za učitaj i ispisi ispred funkcija, umesto sto su unutar funkcija. Zadatak 2.21 - cini mi se da komentari unutar funkcije izmeni narušavaju citljivost koda. Resenje 2.26 — izbaciti nasa slova iz komentara, izbaciti mozda napomenu sa pocetka jer je suvisna

### Test 1

```
Poziv: ./a.out 3
Ulaz:  1 2 3
      -4 -5 -6
      7 8 9
Izlaz: 7 2 2 3
```

### Test 2

```
Poziv: ./a.out 4
Ulaz:  -1 -2 -3 -4
      -5 -6 -7 -8
      -9 -10 -11 -12
      -13 -14 -15 -16
Izlaz: -4 3 0 16
```

### Test 3

```
Poziv: ./a.out
Izlaz: Greska: Nedovoljan broj argumenata komandne linije.
      Program se poziva sa ./a.out dim_matrice.
```

**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.

### Test 1

```
Ulaz: 4
      1 0 0 0
      0 1 0 0
      0 0 1 0
      0 0 0 1
Izlaz: da
```

### Test 2

```
Ulaz: 3
      1 2 3
      5 6 7
      1 4 2
Izlaz: ne
```

### Test 3

```
Ulaz: 33
Izlaz: Greska: neodgovarajuca dimenzija matrice.
```

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

- (a) Napsiati funkciju koja učitava elemente matrice sa standardnog ulaza
- (b) Napsiati 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 (pozivom gore napisane funkcije). Na standardni izlaz spiralno ispisati elemente učitane matrice.

```
Test 1
|| Ulaz:  3 3
||        1 2 3
||        4 5 6
||        7 8 9
|| Izlaz: 1 2 3 6 9 8 7 4 5

Test 2
|| Ulaz:  3 4
||        1 2 3 4
||        5 6 7 8
||        9 10 11 12
|| Izlaz: 1 2 3 4 8 12 11 10 9 5 6 7

Test 3
|| Ulaz:  11 4
|| Izlaz: Greska: neodgovarajuće dimenzije matrice.
```

**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.

```
Test 1
|| Ulaz:  3
||        1 2 3
||        4 5 6
||        7 8 9
||        8
|| Izlaz: 510008400 626654232 743300064
||        1154967822 1419124617 1683281412
||        1799927244 2211595002 2623262760
```

## 2.3 Dinamička alokacija memorije

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

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  3       1 -2 3    Izlaz: 3 -2 1</pre>	<pre>   Ulaz:  -1    Izlaz: malloc(): neuspela alokacija memorije.</pre>

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

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

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  1 -2 3 -4 0    Izlaz: -4 3 -2 1</pre>	<pre>   Ulaz:  0    Izlaz:</pre>

**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 1000 karaktera i da ne sadrže praznine). Na standardni izlaz ispisati nisku koja se dobija njihovim nadovezivanjem. Za rezultujuću nisku dinamički alocirati memoriju.

<i>Test 1</i>
<pre>   Ulaz:  Jedan Dva    Izlaz: JedanDva</pre>

**Zadatak 2.18** Napisati program koji sa standardnog ulaza učitava matricu celih 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.

<i>Test 1</i>
<pre>   Ulaz:  2 3       1.2 2.3 3.4       4.5 5.6 6.7    Izlaz: 6.80</pre>

**Zadatak 2.19** Data je celobrojna matrica dimenzije  $n \times m$  napisati:

- Napisati funkciju koja vrši učitavanje matrice sa standardnog ulaza.
- Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu).

Napisati program koji testira napisane funkcije. 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.

*Test 1*

```
Ulaz:  2 3
       1 -2 3
       -4 5 -6
Izlaz:  1
       -4 5
```

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

*Test 1*

```
Ulaz:  Unesite dimenzije matrice:
       2 3
       Unesite elemente matrice:
       1 2 3
       4 5 6
Izlaz: Kolona pod rednim brojem 3 ima najveći zbir.
```

**Zadatak 2.21** Data je kvadratna realna 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.

### Test 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
Izlaz: 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
```

**Zadatak 2.22** 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. **Jelena: treba dodati test primer.**

**Zadatak 2.23** 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. **Jelena: treba dodati test primer.**

**Zadatak 2.24** Napisati program koji na osnovu dve matrice dimenzija  $m \times n$  formira matricu dimenzije  $2 \cdot m \times n$  tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matrice su zapisane u datoteci „matrice.txt“. U prvom redu se nalaze dimenzije matrica  $m$  i  $n$ , u narednih  $m$  redova se nalaze vrste prve matrice, a u narednih  $m$  redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz. **Jelena: treba dodati test primer.**

**Zadatak 2.25** 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. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz. **Jelena: treba dodati test primer.**

## 2.4 Pokazivači na funkcije

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

*Test 1*

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

*Test 2*

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

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

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

*Test 1*

```
Ulaz:  tan 1.570795 10000
Izlaz: -10134.5
```

*Test 2*

```
Ulaz:  log 0 1000000
Izlaz: -13.81551
```

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

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

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

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

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

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

## 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 }
```

```
18 /* Funkcija obrce elemente niza koriscenjem pokazivacke
   sintakse. Umesto "void obrni_niz(int *a, int n)" potpis
20 metode bi mogao da bude i "void obrni_niz(int a[], int n)". U
   oba slucaja se argument funkcije "a" tumaci kao pokazivac,
22 ili tacnije, kao adresa prvog elementa niza. U odnosu na
   njega se odredjuju adrese ostalih elemenata u nizu */
24 void obrni_niz_v2(int *a, int n)
   {
26     /* Pokazivaci na elemente niza a */
     int *prvi, *poslednji;
28
30     for (prvi = a, poslednji = a + n - 1;
          prvi < poslednji; prvi++, poslednji--) {
32         int t = *prvi;
          *prvi = *poslednji;
34         *poslednji = t;
     }
36 }

38 /* Funkcija obrce elemente niza koriscenjem pokazivacke sintakse
   - modifikovano koriscenje pokazivaca */
40 void obrni_niz_v3(int *a, int n)
   {
42     /* Pokazivaci na elemente niza a */
     int *prvi, *poslednji;
44
     /* Obrcemo niz */
46     for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
         int t = *prvi;
48
         /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
           vrednost koja se nalazi na adresi na koju pokazuje
           pokazivac "poslednji". Nakon toga se pokazivac "prvi"
           uvecava za jedan sto za posledicu ima da "prvi" pokazuje
           na sledeci element u nizu */
52         *prvi++ = *poslednji;
54
         /* Vrednost promenljive "t" se postavlja na adresu na koju
           pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
           umanjuje za jedan, sto za posledicu ima da pokazivac
           "poslednji" sada pokazuje na element koji mu prethodi u
           nizu */
56         *poslednji-- = t;
58     }
60 }
62 }

64 int main()
66 {
   /* Deklaracija niza a od najvise MAX elemenata */
68   int a[MAX];
```

```
70  /* Broj elemenata niza a */
    int n;
72
    /* Pokazivac na elemente niza a */
74  int *p;
76
    /* Unosimo dimenziju niza */
    scanf("%d", &n);
78
    /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
80  if (n <= 0 || n > MAX) {
        fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
82      exit(EXIT_FAILURE);
    }
84
    /* Unosimo elemente niza */
86  for (p = a; p - a < n; p++)
        scanf("%d", p);
88
    obrni_niz_v1(a, n);
90    // obrni_niz_v2(a,n);
    // obrni_niz_v3(a,n);
92
    /* Prikazujemo sadrzaj niza nakon obrtanja */
94  for (p = a; p - a < n; p++)
        printf("%d ", *p);
96  printf("\n");
98
    return 0;
}
```

### Rešenje 2.2

```
#include <stdio.h>
2  #include <stdlib.h>
4
#define MAX 100

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

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

14   return s;
}
16

/* Funkcija racuna proizvod elemenata niza */
18 double proizvod(double a[], int n)
```



```
{
20     double p = 1;

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

24     return p;
26 }

28 /* Funkcija racuna najmanji element niza */
double min(double *a, int n)
30 {
    /* Za najmanji element se najpre postavlja prvi element */
32     double min = a[0];
    int i;

34     /* Ispitujemo da li se medju ostalim elementima niza nalazi
    najmanji */
36     for (i = 1; i < n; i++)
        if (a[i] < min)
            min = a[i];

40     return min;
42 }

44 /* Funkcija racuna najveći element niza */
double max(double *a, int n)
46 {
    /* Za najveći element se najpre postavlja prvi element */
48     double max = *a;

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

54     return max;
56 }
58

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

64     /* Ucitavamo dimenziju niza */
66     scanf("%d", &n);

68     /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
    if (n <= 0 || n > MAX) {
70         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
```

```
        exit(EXIT_FAILURE);
72    }

74    /* Unosimo elemente niza */
    for (i = 0; i < n; i++)
76        scanf("%lf", a + i);

78    /* Testiramo definisane funkcije */
    printf("zbir = %5.3f\n", zbir(a, n));
80    printf("proizvod = %5.3f\n", proizvod(a, n));
    printf("min = %5.3f\n", min(a, n));
82    printf("max = %5.3f\n", max(a, n));

84    return 0;
}
```

### Rešenje 2.3

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

4
/* Funkcija povecava za jedan sve elemente u prvoj polovini niza
6 a smanjuje za jedan sve elemente u drugoj polovini niza.
   Ukoliko niz ima neparan broj elemenata, srednji element ostaje
8 nepromenjen */
void povecaj_smanji(int *a, int n)
10 {
    int *prvi = a;
12    int *poslednji = a + n - 1;

14    while (prvi < poslednji) {

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

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

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

26        /* Pokazivac poslednji se pomera na prethodni element */
        poslednji--;
28    }
30 }

32 void povecaj_smanji_sazetije(int *a, int n)
{
```

```

34  int *prvi = a;
    int *poslednji = a + n - 1;
36
    while (prvi < poslednji) {
38        (*prvi++)++;
        (*poslednji--)--;
40    }
}
42
int main()
44 {
    int a[MAX];
46    int n;
    int *p;
48
    /* Unosimo broj elemenata */
50    scanf("%d", &n);
52
    /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
    if (n <= 0 || n > MAX) {
54        fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
        exit(EXIT_FAILURE);
56    }
58
    /* Unosimo elemente niza */
    for (p = a; p - a < n; p++)
60        scanf("%d", p);
62
    povecaj_smanji(a, n);
    /* povecaj_smanji_sazetije(a,n); */
64
    /* Prikaz niza nakon modifikacije */
66    for (p = a; p - a < n; p++)
        printf("%d ", *p);
68    printf("\n");
70
    return 0;
}

```

## Rešenje 2.4

```

#include <stdio.h>
2
/* Argumenti funkcije main mogu da budu broj argumenta komandne
4    linije (int argc) i niz arugmenata komandne linije (niz
    niski) (char *argv[] <=> char** argv) */
6 int main(int argc, char *argv[])
{
8     int i;
10
    /* Ispisujemo broj argumenata komandne linije */

```

```
printf("%d\n", argc);

12
/* Ispisujemo argumente komandne linije */
14 /* koristeći indeksnu sintaksu */
for (i = 0; i < argc; i++) {
16     printf("%d %s\n", i, argv[i]);
}

18
/* koristeći pokazivačku sintaksu */
20 i = argc;
for (; argc > 0; argc--)
22     printf("%d %s\n", i - argc, *argv++);

24
/* Nakon ove petlje "argc" će biti jednako nuli a "argv" će
26 pokazivati na polje u memoriji koje se nalazi iza
poslednjeg argumenta komandne linije. Kako smo u
28 promenljivoj "i" sacuvali vrednost broja argumenta komandne
linije to sada mozemo ponovo da postavimo "argv" da
30 pokazuje na nulti argument komandne linije */
argv = argv - i;
32 argc = i;

34 /* Ispisujemo 0-ti karakter svakog od argumenata komandne
linije */

36
/* koristeći indeksnu sintaksu */
38 for (i = 0; i < argc; i++)
    printf("%c ", argv[i][0]);
40 printf("\n");

42
/* koristeći pokazivačku sintaksu */

44 for (i = 0; i < argc; i++)
    printf("%c ", **argv++);

46
return 0;
48 }
```

### Rešenje 2.5

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

4
/* Funkcija ispituje da li je niska palindrom */
6 int palindrom(char *niska)
{
8     int i, j;
    for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
10         if (*(niska + i) != *(niska + j))
```

```

    return 0;
12  return 1;
}
14
int main(int argc, char **argv)
16 {
    int i, n = 0;
18
    /* Multi argument komandne linije je ime izvrsnog programa */
20    for (i = 1; i < argc; i++)
        if (palindrom(*(argv + i)))
22        n++;
24    printf("%d\n", n);
    return 0;
26 }

```

## Rešenje 2.6

```

1  #include<stdio.h>
   #include<stdlib.h>
3
   #define MAX_KARAKTERA 100
5
   /* Funkcija strlen() iz standardne biblioteke */
7  int duzina(char *s)
   {
9      int i;
      for (i = 0; *(s + i); i++);
11     return i;
   }
13
int main(int argc, char **argv)
15 {
    char rec[MAX_KARAKTERA];
17     int br = 0, n;
    FILE *in;
19
    /* Ako korisnik nije uneo trazene argumente, prijavljujemo
21     gresku */
    if (argc < 3) {
23         printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
25         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
                argv[0]);
        exit(EXIT_FAILURE);
27     }
29
    /* Otvaramo datoteku sa imenom koje se zadaje kao prvi
31     argument komandne linije. */
    in = fopen(*(argv + 1), "r");

```

## 2 Pokazivači

```
33  if (in == NULL) {
    fprintf(stderr, "Greska: ");
35  fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
        argv[1]);
37  exit(EXIT_FAILURE);
    }

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

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

47  printf("%d\n", br);

49  /* Zatvaramo datoteku */
51  fclose(in);
  return 0;
53 }
```

### Rešenje 2.7

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

3  #define MAX_KARAKTERA 100

5  /* Funkcija strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
   {
9      int i;
      for (i = 0; *(src + i); i++)
11         *(dest + i) = *(src + i);
   }

13  /* Funkcija strcmp() iz standardne biblioteke */
15  int poredjenje_niski(char *s, char *t)
   {
17      int i;
      for (i = 0; *(s + i) == *(t + i); i++)
19         if (*(s + i) == '\0')
             return 0;
21     return *(s + i) - *(t + i);
   }

23  /* Funkcija strlen() iz standardne biblioteke */
25  int duzina_niske(char *s)
   {
27     int i;
```

```
    for (i = 0; *(s + i); i++);
29     return i;
30 }
31
32 /* Funkcija ispituje da li je niska zadata drugim argumentom
33     funkcije sufiks niske zadate prvi argumentom funkcije */
34 int sufiks_niske(char *niska, char *sufiks)
35 {
36     if (duzina_niske(sufiks) <= duzina_niske(niska) &&
37         poredjenje_niski(niska + duzina_niske(niska) -
38             duzina_niske(sufiks), sufiks) == 0)
39         return 1;
40     return 0;
41 }
42
43 /* Funkcija ispituje da li je niska zadata drugim argumentom
44     funkcije prefiks niske zadate prvi argumentom funkcije */
45 int prefiks_niske(char *niska, char *prefiks)
46 {
47     int i;
48     if (duzina_niske(prefiks) <= duzina_niske(niska)) {
49         for (i = 0; i < duzina_niske(prefiks); i++)
50             if (*(prefiks + i) != *(niska + i))
51                 return 0;
52         return 1;
53     } else
54         return 0;
55 }
56
57 int main(int argc, char **argv)
58 {
59     /* Ako korisnik nije uneo trazene argumente, prijavljujemo
60         gresku */
61     if (argc < 4) {
62         printf("Greska: ");
63         printf("Nedovoljan broj argumenata komandne linije.\n");
64         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
65             argv[0]);
66         exit(EXIT_FAILURE);
67     }
68
69     FILE *in;
70     int br = 0;
71     char rec[MAX_KARAKTERA];
72
73     in = fopen(*(argv + 1), "r");
74     if (in == NULL) {
75         fprintf(stderr, "Greska: ");
76         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
77             argv[1]);
78         exit(EXIT_FAILURE);
79     }
```

```
81  /* Proveravamo kojom opcijom je pozvan program a zatim
      učitavamo reci iz datoteke brojimo koliko reci zadovoljava
83  traženi uslov */
      if (!(poredjenje_niski(*(argv + 3), "-s")))
85          while (fscanf(in, "%s", rec) != EOF)
              br += sufiks_niske(rec, *(argv + 2));
87      else if (!(poredjenje_niski(*(argv + 3), "-p")))
          while (fscanf(in, "%s", rec) != EOF)
89              br += prefiks_niske(rec, *(argv + 2));

91      printf("%d\n", br);
      fclose(in);
93      return 0;
  }
```

### Rešenje 2.8

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

4
  #define MAX 100

6
  /* Deklarisemo funkcije koje cemo kasnije da definisemo */
8  double euklidska_norma(int M[][MAX], int n);
  int trag(int M[][MAX], int n);
10 int gornja_vandijagonalna_norma(int M[][MAX], int n);

12 int main()
  {
14     int A[MAX][MAX];
      int i, j, n;

16
      /* Unosimo dimenziju kvadratne matrice */
18     scanf("%d", &n);

20
      /* Proveravamo da li je prekoraceno ogranicenje */
      if (n > MAX || n <= 0) {
22         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
          fprintf(stderr, "matrice.\n");
24         exit(EXIT_FAILURE);
      }

26
      /* Popunjavamo vrstu po vrstu matrice */
28     for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
30         scanf("%d", &A[i][j]);

32
      /* Ispis elemenata matrice koriscenjem indeksne sintakse.
          Ispis vrsimo vrstu po vrstu */
```



```

34  for (i = 0; i < n; i++) {
    /* Ispisujemo elemente i-te vrste */
36  for (j = 0; j < n; j++)
    printf("%d ", A[i][j]);
38  printf("\n");
    }
40
    /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
    Kod ovako definisane matrice, elementi su uzastopno
    smesteni u memoriju, kao na traci. To znaci da su svi
    elementi prve vrste redom smesteni jedan iza drugog. Odmah
    44 iza poslednjeg elementa prve vrste smesten je prvi element
    druge vrste za kojim slede svi elementi te vrste i tako
    46 dalje redom */
    /*
    48 for( i = 0; i<n; i++) { for ( j=0 ; j<n; j++) printf("%d ",
    50 *(A+i+j)); printf("\n"); } */

    int tr = trag(A, n);
    printf("trag = %d\n", tr);
54
    printf("euklidska norma = %.2f\n", euklidska_norma(A, n));
    printf("vandijagonalna norma = %d\n",
    56 gornja_vandijagonalna_norma(A, n));

    return 0;
60 }

62 /* Definisemo funkcije koju smo ranije deklarirali */

64 /* Funkcija izracunava trag matrice */
int trag(int M[][MAX], int n)
66 {
    int trag = 0, i;
    for (i = 0; i < n; i++)
    68     trag += M[i][i];
    return trag;
70 }

72 /* Funkcija izracunava euklidsku normu matrice */
double euklidska_norma(int M[][MAX], int n)
74 {
    double norma = 0.0;
    int i, j;
    76
    for (i = 0; i < n; i++)
    78     for (j = 0; j < n; j++)
    80         norma += M[i][j] * M[i][j];
    82
    return sqrt(norma);
    84 }

```

```
86 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
87 int gornja_vandijagonalna_norma(int M[][MAX], int n)
88 {
89     int norma = 0;
90     int i, j;
91
92     for (i = 0; i < n; i++) {
93         for (j = i + 1; j < n; j++)
94             norma += abs(M[i][j]);
95     }
96
97     return norma;
98 }
```

### 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
   standardnog ulaza */
7 void ucitaj_matricu(int m[][MAX], int n)
8 {
9     int i, j;
10
11     for (i = 0; i < n; i++)
12         for (j = 0; j < n; j++)
13             scanf("%d", &m[i][j]);
14 }
15
16 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
   standardni izlaz */
17 void ispisi_matricu(int m[][MAX], int n)
18 {
19     int i, j;
20
21     for (i = 0; i < n; i++) {
22         for (j = 0; j < n; j++)
23             printf("%d ", m[i][j]);
24         printf("\n");
25     }
26 }
27
28
29 /* Funkcija proverava da li su zadate kvadratne matrice a i b
   dimenzije n jednake */
30 int jednake_matrice(int a[][MAX], int b[][MAX], int n)
31 {
32     int i, j;
33
34     for (i = 0; i < n; i++)
35         for (j = 0; j < n; j++)
36             if (a[i][j] != b[i][j])
37                 return 0;
38     return 1;
39 }
```

```
36     for (i = 0; i < n; i++)
37         for (j = 0; j < n; j++)
38             /* Nasli smo elemente na istim pozicijama u matricama koji
39                se razlikuju */
40             if (a[i][j] != b[i][j])
41                 return 0;
42
43     /* Prosli je provera jednakosti za sve parove elemenata koji
44        su na istim pozicijama sto znaci da su matrice jednake */
45     return 1;
46 }
47
48 /* Funkcija izracunava zbir dve kvadratne matice */
49 void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
50 {
51     int i, j;
52
53     for (i = 0; i < n; i++)
54         for (j = 0; j < n; j++)
55             c[i][j] = a[i][j] + b[i][j];
56 }
57
58 /* Funkcija izracunava proizvod dve kvadratne matice */
59 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
60 {
61     int i, j, k;
62
63     for (i = 0; i < n; i++)
64         for (j = 0; j < n; j++) {
65             /* Mnozimo i-tu vrstu prve sa j-tom kolonom druge matrice */
66             c[i][j] = 0;
67             for (k = 0; k < n; k++)
68                 c[i][j] += a[i][k] * b[k][j];
69         }
70 }
71
72 int main()
73 {
74     /* Matrice ciji se elementi zadaju sa ulaza */
75     int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
76
77     /* Matrice zbira i proizvoda */
78     int zbir[MAX][MAX], proizvod[MAX][MAX];
79
80     /* Dimenzija matrica */
81     int n;
82     int i, j;
83
84     /* Ucitavamo dimenziju kvadratnih matrica i proveravamo njenu
85        korektnost */
86     scanf("%d", &n);
```

```
88  /* Proveravamo da li je prekoraceno ogranicenje */
    if (n > MAX || n <= 0) {
90      fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
      fprintf(stderr, "matrica.\n");
92      exit(EXIT_FAILURE);
    }

94
    /* Ucitavamo matrice */
96    ucitaj_matricu(a, n);
    ucitaj_matricu(b, n);

98
    /* Izracunavamo zbir i proizvod matrica */
100   saberi(a, b, zbir, n);
    pomnozi(a, b, proizvod, n);

102
    /* Ispisujemo rezultat */
104   if (jednake_matrice(a, b, n) == 1)
        printf("da\n");
106   else
        printf("ne\n");

108
    printf("Zbir matrica je:\n");
110   ispisi_matricu(zbir, n);

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

114
    return 0;
116 }
```

### Rešenje 2.10

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

4 #define MAX 64

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

14     /* Obilazimo glavnu dijagonalu matrice. Za elemente na glavnoj
       dijagonali vazi da je indeks vrste jednak indeksu kolone */
16     for (i = 0; i < n; i++) {
        if (m[i][i] != 1)
18         return 0;
    }
}
```

```

20     return 1;
22 }

24 /* Funkcija odredjuje refleksivno zatvorenje zadate relacije.
   Ono je odredjeno matricom koja sadrzi sve elemente polazne
26 matrice dopunjene jedinicama na glavnoj dijagonali */
void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
28 {
    int i, j;

30     /* Prepisujemo vrednosti elemenata matrice pocetne matrice */
32     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
34             zatvorenje[i][j] = m[i][j];

36     /* Postavljamo na glavnoj dijagonali jedinice */
38     for (i = 0; i < n; i++)
        zatvorenje[i][i] = 1;
39 }

40 /* Funkcija proverava da li je relacija simetricna. Relacija je
42 simetricna ako za svaki par elemenata vazi: ako je element
   "i" u relaciji sa elementom "j", onda je i element "j" u
44 relaciji sa elementom "i". Ovakve matrice su simetricne u
   odnosu na glavnu dijagonalu */
46 int simetricnost(int m[][MAX], int n)
47 {
48     int i, j;

50     /* Obilazimo elemente ispod glavne dijagonale matrice i
       uporedjujemo ih sa njima simetricnim elementima */
52     for (i = 0; i < n; i++)
        for (j = 0; j < i; j++)
54             if (m[i][j] != m[j][i])
                return 0;

56     return 1;
57 }

60 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono
   je odredjeno matricom koja sadrzi sve elemente polazne
62 matrice dopunjene tako da matrica postane simetricna u odnosu
   na glavnu dijagonalu */
64 void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
65 {
66     int i, j;

68     /* Prepisujemo vrednosti elemenata matrice m */
69     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
70             zatvorenje[i][j] = m[i][j];

```

```

72      /* Odredjujemo simetricno zatvorenje matrice */
74      for (i = 0; i < n; i++)
76          for (j = 0; j < n; j++)
78              if (zatvorenje[i][j] == 1)
79                  zatvorenje[j][i] = 1;
80
81      /* Funkcija proverava da li je relacija tranzitivna. Relacija je
82      tranzitivna ako ispunjava sledece svojstvo: ako je element
83      "i" u relaciji sa elementom "j" i element "j" u relaciji sa
84      elementom "k", onda je i element "i" u relaciji sa elementom
85      "k" */
86      int tranzitivnost(int m[][MAX], int n)
87      {
88          int i, j, k;
89
90          for (i = 0; i < n; i++)
91              for (j = 0; j < n; j++)
92                  /* Pokusavamo da pronadjemo element koji narušava *
93                  tranzitivnost */
94                  for (k = 0; k < n; k++)
95                      if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
96                          return 0;
97
98          return 1;
99      }
100
101      /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
102      relacije koriscenjem Varsalovog algoritma */
103      void tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
104      {
105          int i, j, k;
106
107          /* Kopiramo pocetnu matricu u matricu rezultata */
108          for (i = 0; i < n; i++)
109              for (j = 0; j < n; j++)
110                  zatvorenje[i][j] = m[i][j];
111
112          /* Primenom Varsalovog algoritma odredjujemo
113          refleksivno-tranzitivno zatvorenje matrice */
114          for (k = 0; k < n; k++)
115              for (i = 0; i < n; i++)
116                  for (j = 0; j < n; j++)
117                      if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
118                          && (zatvorenje[i][j] == 0))
119                          zatvorenje[i][j] = 1;
120      }
121
122      /* Funkcija ispisuje elemente matrice */

```

```
124 void pisi_matricu(int m[][MAX], int n)
125 {
126     int i, j;
127
128     for (i = 0; i < n; i++) {
129         for (j = 0; j < n; j++)
130             printf("%d ", m[i][j]);
131         printf("\n");
132     }
133 }
134
135 int main(int argc, char *argv[])
136 {
137     FILE *ulaz;
138     int m[MAX][MAX];
139     int pomocna[MAX][MAX];
140     int n, i, j, k;
141
142     /* Ako korisnik nije uneo trazene argumente, prijavljujemo
143        gresku */
144     if (argc < 2) {
145         printf("Greska: ");
146         printf("Nedovoljan broj argumenata komandne linije.\n");
147         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
148         exit(EXIT_FAILURE);
149     }
150
151     /* Otvaramo datoteku za citanje */
152     ulaz = fopen(argv[1], "r");
153     if (ulaz == NULL) {
154         fprintf(stderr, "Greska: ");
155         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
156             argv[1]);
157         exit(EXIT_FAILURE);
158     }
159
160     /* Ucitavamo dimenziju matrice */
161     fscanf(ulaz, "%d", &n);
162
163     /* Proveravamo da li je prekoraceno ogranicenje */
164     if (n > MAX || n <= 0) {
165         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
166         fprintf(stderr, "matrice.\n");
167         exit(EXIT_FAILURE);
168     }
169
170     /* Ucitavamo element po element matrice */
171     for (i = 0; i < n; i++)
172         for (j = 0; j < n; j++)
173             fscanf(ulaz, "%d", &m[i][j]);
174
175     /* Ispisujemo trazene vrednosti */
```

```
176     printf("Refleksivnost: %s\n",
177            refleksivnost(m, n) == 1 ? "da" : "ne");
178
179     printf("Simetricnost: %s\n",
180            simetricnost(m, n) == 1 ? "da" : "ne");
181
182     printf("Tranzitivnost: %s\n",
183            tranzitivnost(m, n) == 1 ? "da" : "ne");
184
185     printf("Refleksivno zatvorenje:\n");
186     ref_zatvorenje(m, n, pomocna);
187     pisi_matricu(pomocna, n);
188
189     printf("Simetricno zatvorenje:\n");
190     sim_zatvorenje(m, n, pomocna);
191     pisi_matricu(pomocna, n);
192
193     printf("Refleksivno-tranzitivno zatvorenje:\n");
194     tran_zatvorenje(m, n, pomocna);
195     pisi_matricu(pomocna, n);
196
197     /* Zatvaramo datoteku */
198     fclose(ulaz);
199
200     return 0;
201 }
```

### Rešenje 2.11

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 32
5
6  int max_sporedna_dijagonala(int m[][MAX], int n)
7  {
8      int i, j;
9      /* Trazimo najveći element na sporednoj dijagonali. Za
10         elemente sporedne dijagonale vazi da je zbir indeksa vrste
11         i indeksa kolone jednak n-1. Za pocetnu vrednost maksimuma
12         uzimamo element u gornjem desnom uglu */
13     int max_na_sporednoj_dijagonali = m[0][n - 1];
14     for (i = 1; i < n; i++)
15         if (m[i][n - 1 - i] > max_na_sporednoj_dijagonali)
16             max_na_sporednoj_dijagonali = m[i][n - 1 - i];
17
18     return max_na_sporednoj_dijagonali;
19 }
20
21 /* Funkcija izracunava indeks kolone najmanjeg elementa */
22 int indeks_min(int m[][MAX], int n)
```



```

23 {
24     int i, j;
25     /* Za pocetnu vrednost minimuma uzimamo element u gornjem
        levom uglu */
27     int min = m[0][0], indeks_kolone = 0;
28
29     for (i = 0; i < n; i++)
30         for (j = 0; j < n; j++)
31             /* Ako je tekuci element manji od minimalnog */
32             if (m[i][j] < min) {
33                 /* cuvamo njegovu vrednost */
34                 min = m[i][j];
35                 /* i cuvamo indeks kolone u kojoj se nalazi */
36                 indeks_kolone = j;
37             }
38     return indeks_kolone;
39 }
40
41 /* Funkcija izracunava indeks vrste najveceg elementa */
42 int indeks_max(int m[][MAX], int n)
43 {
44     int i, j;
45     /* Za maksimalni element uzimamo gornji levi ugao */
46     int max = m[0][0], indeks_vrste = 0;
47
48     for (i = 0; i < n; i++)
49         for (j = 0; j < n; j++)
50             /* Ako je tekuci element manji od minimalnog */
51             if (m[i][j] > max) {
52                 /* cuvamo njegovu vrednost */
53                 max = m[i][j];
54                 /* i cuvamo indeks vrste u kojoj se nalazi */
55                 indeks_vrste = i;
56             }
57     return indeks_vrste;
58 }
59
60 /* Funkcija izracunava broj negativnih elemenata matrice */
61 int broj_negativnih(int m[][MAX], int n)
62 {
63     int i, j;
64
65     int broj_negativnih = 0;
66
67     for (i = 0; i < n; i++)
68         for (j = 0; j < n; j++)
69             if (m[i][j] < 0)
70                 broj_negativnih++;
71     return broj_negativnih;
72 }
73
74 int main(int argc, char *argv[])

```

```
75 {
77     int m[MAX][MAX];
79     int n;
81     int i, j;

83     /* Proveravamo broj argumenata komandne linije */
85     if (argc < 2) {
87         printf("Greska: ");
89         printf("Nedovoljan broj argumenata komandne linije.\n");
91         printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
93         exit(EXIT_FAILURE);
95     }

97     /* Ucitavamo vrednost dimenzije i proveravamo njenu korektnost */
99     n = atoi(argv[1]);

101     if (n > MAX || n <= 0) {
103         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
105         fprintf(stderr, "matrice.\n");
107         exit(EXIT_FAILURE);
109     }

111     /* Ucitavamo element po element matrice */
113     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &m[i][j]);

115     int max_sd = max_speredna_dijagonala(m, n);
117     int i_min = indeks_min(m, n);
119     int i_max = indeks_max(m, n);
121     int bn = broj_negativnih(m, n);

123     /* Ispisujemo rezultat */
125     printf("%d %d %d %d\n", max_sd, i_min, i_max, bn);

127     /* Prekidamo izvršavanje programa */
129     return 0;
131 }
```

### Rešenje 2.12

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

4 #define MAX 32

6 /* Funkcija ucitava elemente kvadratne matrice sa standardnog
   ulaza */
8 void ucitaj_matricu(int m[][MAX], int n)
9 {
```

```
10     int i, j;

12     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
14         scanf("%d", &m[i][j]);
15 }

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

22     for (i = 0; i < n; i++) {
23         for (j = 0; j < n; j++)
24             printf("%d ", m[i][j]);
25         printf("\n");
26     }
27 }

28 /* Funkcija proverava da li je zadata matrica ortonormirana */
29 int ortonormirana(int m[][MAX], int n)
30 {
31     int i, j, k;
32     int proizvod;

33     /* Proveravamo uslov normiranosti, odnosno da li je proizvod
34        svake vrste matrice sa samom sobom jednak jedinici */
35     for (i = 0; i < n; i++) {

36         /* Izracunavamo skalarni proizvod vrste sa samom sobom */
37         proizvod = 0;

38         for (j = 0; j < n; j++)
39             proizvod += m[i][j] * m[i][j];

40         /* Ako proizvod bar jedne vrste nije jednak jedinici, odmah
41            zakljucujemo da matrica nije normirana */
42         if (proizvod != 1)
43             return 0;
44     }

45     /* Proveravamo uslov ortogonalnosti, odnosno da li je proizvod
46        dve bilo koje razlicite vrste matrice jednak nuli */
47     for (i = 0; i < n - 1; i++) {
48         for (j = i + 1; j < n; j++) {

49             /* Izracunavamo skalarni proizvod */
50             proizvod = 0;

51             for (k = 0; k < n; k++)
52                 proizvod += m[i][k] * m[j][k];

53         }
54     }
55 }
```

```
62         /* Ako proizvod dve bilo koje razlicite vrste nije jednak
64         nuli, odmah zakljucujemo da matrica nije ortogonalna */
        if (proizvod != 0)
66             return 0;
        }
68     }

70     /* Ako su oba uslova ispunjena, vracamo jedinicu kao rezultat */
    return 1;
72 }

74 int main()
75 {
76     int A[MAX][MAX];
77     int n;

78     /* Ucitavamo vrednost dimenzije i proveravamo njenu korektnost
80     */
    scanf("%d", &n);

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

88     /* Ucitavamo matricu */
89     ucitaj_matricu(A, n);

90     /* Ispisujemo rezultat rada funkcije */
    if (ortonormirana(A, n))
92         printf("da\n");
    else
94         printf("ne\n");

96     return 0;
98 }
}
```

### Rešenje 2.13

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

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 narušio prirodan poredak medju granicama */
9 int krajIspisa(int top, int bottom, int left, int right)
10 {
```

```
12     return !(top <= bottom && left <= right);
13 }
14 /* Funkcija spiralno ispisuje elemente matrice */
15 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16 {
17     int i, j, top, bottom, left, right;
18
19     top = left = 0;
20     bottom = n - 1;
21     right = m - 1;
22
23     while (!krajIspisa(top, bottom, left, right)) {
24         /* Ispisuje se prvi red */
25         for (j = left; j <= right; j++)
26             printf("%d ", a[top][j]);
27
28         /* Spustamo prvi red */
29         top++;
30
31         if (krajIspisa(top, bottom, left, right))
32             break;
33
34         for (i = top; i <= bottom; i++)
35             printf("%d ", a[i][right]);
36
37         /* Pomeramo desnu kolonu za naredni krug ispisa blize levom
38            kraju */
39         right--;
40
41         if (krajIspisa(top, bottom, left, right))
42             break;
43
44         /* Ispisujemo donju vrstu */
45         for (j = right; j >= left; j--)
46             printf("%d ", a[bottom][j]);
47
48         /* Podizemo donju vrstu za naredni krug ispisa */
49         bottom--;
50
51         if (krajIspisa(top, bottom, left, right))
52             break;
53
54         /* Ispisujemo prvu kolonu */
55         for (i = bottom; i >= top; i--)
56             printf("%d ", a[i][left]);
57
58         /* Pripremamo levu kolonu za naredni krug ispisa */
59         left++;
60     }
61     putchar('\n');
62 }
```

```
64 void ucitaj_matricu(int a[][MAX_K], int n, int m)
65 {
66     int i, j;
67
68     for (i = 0; i < n; i++)
69         for (j = 0; j < m; j++)
70             scanf("%d", &a[i][j]);
71 }
72
73 int main()
74 {
75     int a[MAX_V][MAX_K];
76     int m, n;
77
78     /* Ucitaj broj vrsta i broj kolona matrice */
79     scanf("%d", &n);
80     scanf("%d", &m);
81
82     if (n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
83         fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
84         fprintf(stderr, "matrice.\n");
85         exit(EXIT_FAILURE);
86     }
87
88     ucitaj_matricu(a, n, m);
89     ispisi_matricu_spiralno(a, n, m);
90
91     return 0;
92 }
```

### Rešenje 2.14

### Rešenje 2.15

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* NAPOMENA: Primer demonstrira dinamičku alokaciju niza od n
5    elemenata. Dovoljno je alocirati n * sizeof(T) bajtova, gde
6    je T tip elemenata niza. Povratnu adresu malloc()-a treba
7    pretvoriti iz void * u T *, kako bismo dobili pokazivac koji
8    pokazuje na prvi element niza tipa T. Na dalje se elementima
9    može pristupati na isti način kao da nam je dato ime niza
10   (koje se tako i ponasa - kao pokazivac na element tipa T koji
11   je prvi u nizu) */
12 int main()
13 {
14     int *p = NULL;
15     int i, n;
```

```

16  /* Unosimo dimenziju niza. Ova vrednost nije ogranicena bilo
18     kakvom konstantom, kao sto je to ranije bio slucaj kod
20     staticke alokacije gde je dimenzija niza bila unapred
        ogranicena definisanim prostorom. */
22  scanf("%d", &n);

24  /* Alociramo prostor za n celih brojeva */
26  if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
        fprintf(stderr, "malloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
28  }

30  /* Od ovog trenutka pokazivac "p" mozemo da koristimo kao da
        je ime niza, odnosno i-tom elementu se moze pristupiti sa
32     p[i] */

34  /* Unosimo elemente niza */
36  for (i = 0; i < n; i++)
        scanf("%d", &p[i]);

38  /* Ispisujemo elemente niza unazad */
40  for (i = n - 1; i >= 0; i--)
        printf("%d ", p[i]);
        printf("\n");

42  /* Oslobadjamo prostor */
44  free(p);

46  return 0;
}

```

### Rešenje 2.16

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define KORAK 10
4
5  int main(void)
6  {
7      /* Adresa prvog alociranog bajta */
8      int *a = NULL;
9
10     /* Velicina alocirane memorije */
11     int alocirano;
12
13     /* Broj elemenata niza */
14     int n;
15
16     /* Broj koji se učitava sa ulaza */

```

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

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

24  /* Unosimo brojeve sa ulaza */
    scanf("%d", &x);

26
    /* Sve dok je procitani broj razlicit od nule... */
28  while (x != 0) {

30      /* Ako broj ucitanih elemenata niza odgovara broju
        alociranih mesta, za smestanje novog elementa treba
32      obezbediti dodatni prostor. Da se ne bi za svaki sledeci
        element pojedinačno alocirala memorija, prilikom
34      alokacije se vrši rezervacija za još KORAK dodatnih mesta
        za buduće elemente */
36      if (n == alocirano) {
        /* Povećava se broj alociranih mesta */
38      alocirano = alocirano + KORAK;

40      /* Vrši se realokacija memorije sa novom veličinom */
        /******
42      Resenje sa funkcijom malloc() */
        /******
44      /* Vrši se alokacija memorije sa novom veličinom, a adresa
        početka novog memorijskog bloka se čuva u promenljivoj
46      b */
        b = (int *) malloc(alocirano * sizeof(int));

48
        /* Ako prilikom alokacije dodje do neke greske */
50      if (b == NULL) {
        /* poruku ispisujemo na izlaz za greske */
52      fprintf(stderr, "malloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");

54
        /* Pre kraja programa moramo svu dinamički alociranu
        memoriju da oslobodimo. U ovom slucaju samo memoriju
56      na adresi a */
58      free(a);

60      /* Završavamo program */
        exit(EXIT_FAILURE);
62      }

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

68
```



```

70     /* Posle prepisivanja oslobadjamo blok memorije sa
       pocetnom adresom u a */
       free(a);

72

74     /* Promenljivoj a dodeljujemo adresu pocetka novog, veceg
       bloka koji je prilikom alokacije zapamcen u
       promenljivoj b */
76     a = b;

78     /******
       /* Resenje sa funkcijom realloc() */
80     /******
       /* Zbog funkcije realloc je neophodno da i u prvoj
82     iteraciji "a" bude inicijalizovano na NULL */
       /*
84     a = (int*) realloc(a,alocirano*sizeof(int));

86     if(a == NULL) { fprintf(stderr, "realloc(): ");
       fprintf(stderr, "greska pri alokaciji memorije.\n");
88     exit(EXIT_FAILURE); } */

       }

90     /* Smestamo element u niz */
92     a[n++] = x;

94     /* i učitavamo sledeci element */
       scanf("%d", &x);
96 }

98     /* Ispisujemo brojeve u obrnutom poretку */
       for (n--; n >= 0; n--)
100     printf("%d ", a[n]);
       printf("\n");
102

       /* Oslobadjamo dinamički alociranu memoriju */
104     free(a);

106     /* Program se završava */
       exit(EXIT_SUCCESS);
108 }

```

### Rešenje 2.17

```

#include <stdio.h>
2  #include <stdlib.h>
   #include <string.h>
4
   #define MAX 1000
6
   /* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
8   Ovako nesto se moze improvizovati tako sto se u funkciji

```

```
10     dinamicki kreira niz potrebne velicine, popuni se potrebnim
12     informacijama, a zatim se vrati njegova adresa. Imajuci u
14     vidu cinjenicu da dinamicki kreiran objekat ne nestaje kada
16     se izadje iz funkcije koja ga je kreirala, vraceni pokazivac
18     se kasnije u pozivajucoj funkciji moze koristiti za pristup
20     "vracenom" nizu. Medjutim, pozivajuca funkcija ima
22     odgovornost i da se brine o dealokaciji istog prostora */
24
26     /* Funkcija dinamicki kreira niz karaktera u koji smesta
28     rezultat nadovezivanja niski. Adresa niza se vraca kao
30     povratna vrednost. */
32
34     char *nadovezi(char *s, char *t)
36     {
38         /* Dinamicki kreiramo prostor dovoljne velicine */
40         char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
42             * sizeof(char));
44
46         /* Proveravamo uspeh alokacije */
48         if (p == NULL) {
50             fprintf(stderr, "malloc(): ");
52             fprintf(stderr, "greska pri alokaciji memorije.\n");
54             exit(EXIT_FAILURE);
56         }
58
60         /* Kopiramo i nadovezujemo stringove */
62
64         /* Resenje bez koriscenja biblioteckih funkcija */
66         /*
68          int i,j; for(i=j=0; s[j]!='\0'; i++, j++) p[i]=s[j];
70
72          for(j=0; t[j]!='\0'; i++, j++) p[i]=t[j];
74
76          p[i]='\0'; */
78
80         /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
82         string.h */
84         strcpy(p, s);
86         strcat(p, t);
88
90         /* Vracamo pokazivac p */
92         return p;
94     }
96
98     int main()
100     {
102         char *s = NULL;
104         char s1[MAX], s2[MAX];
106
108         /* Ucitavamo dve niske koje cemo da nadovezemo */
110         scanf("%s", s1);
112         scanf("%s", s2);
114     }
```

```

62  /* Pozivamo funkciju da nadoveze stringove */
    s = nadovezi(s1, s2);

64  /* Prikazujemo rezultat */
    printf("%s\n", s);

66  /* Oslobadjamo memoriju alociranu u funkciji nadovezi() */
68  free(s);

70  return 0;
}

```

### Rešenje 2.18

```

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

4
int main()
6 {
    int i, j;

8
    /* Pokazivac na dinamički alociran niz pokazivaca na vrste
10     matrice */
    double **A = NULL;

12
    /* Broj vrsta i broj kolona */
14    int n = 0, m = 0;

16
    /* Trag matrice */
    double trag = 0;

18
    /* Unosimo dimenzije matrice */
20    scanf("%d%d", &n, &m);

22
    /* Dinamički alociramo prostor za n pokazivaca na double */
    A = malloc(sizeof(double *) * n);

24
    /* Proveramo da li je doslo do greske pri alokaciji */
26    if (A == NULL) {
        fprintf(stderr, "malloc(): ");
28        fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
30    }

32
    /* Dinamički alociramo prostor za elemente u vrstama */
    for (i = 0; i < n; i++) {
34        A[i] = malloc(sizeof(double) * m);

36        if (A[i] == NULL) {
            /* Alokacija je neuspesna. Pre zavrsetka programa moramo

```

```
38         da oslobodimo svih i-1 prethodno alociranih vrsta, i
        alociran niz pokazivaca */
40     for (j = 0; j < i; j++)
        free(A[j]);
42     free(A);

44     exit(EXIT_FAILURE);
    }
46 }

48 /* Unosimo sa standardnog ulaza brojeve u matricu. Popunjavamo
    vrstu po vrstu */
50 for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
52         scanf("%lf", &A[i][j]);

54 /* Racunamo trag matrice, odnosno sumu elemenata na glavnoj
    dijagonali */
56 trag = 0.0;

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

60 printf("%.2f\n", trag);

62 /* Oslobadjamo prostor rezervisan za svaku vrstu */
64 for (j = 0; j < n; j++)
    free(A[j]);

66 /* Oslobadjamo memoriju za niz pokazivaca na vrste */
68 free(A);

70 return 0;
}
```

### Rešenje 2.19

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

5 void ucitaj_matricu(int **M, int n, int m)
{
7     int i, j;

9     /* Popunjavamo matricu vrstu po vrstu */
    for (i = 0; i < n; i++)
11         /* Popunjavamo i-tu vrstu matrice */
        for (j = 0; j < m; j++)
13             scanf("%d", &M[i][j]);
}
```

```
15 void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
16 {
17     int i, j;
18
19     for (i = 0; i < n; i++) {
20         /* Ispisujemo elemente ispod glavne dijagonale matrice */
21         for (j = 0; j <= i; j++)
22             printf("%d ", M[i][j]);
23         printf("\n");
24     }
25 }
26
27 int main()
28 {
29     int m, n, i, j;
30     int **matrica = NULL;
31
32     /* Unosimo dimenzije matrice */
33     scanf("%d %d", &n, &m);
34
35     /* Alociramo prostor za niz pokazivaca na vrste matrice */
36     matrica = (int **) malloc(n * sizeof(int *));
37     if (matrica == NULL) {
38         fprintf(stderr, "malloc(): Neuspela alokacija\n");
39         exit(EXIT_FAILURE);
40     }
41
42     /* Alociramo prostor za svaku vrstu matrice */
43     for (i = 0; i < n; i++) {
44         matrica[i] = (int *) malloc(m * sizeof(int));
45
46         if (matrica[i] == NULL) {
47             fprintf(stderr, "malloc(): Neuspela alokacija\n");
48             for (j = 0; j < i; j++)
49                 free(matrica[j]);
50             free(matrica);
51             exit(EXIT_FAILURE);
52         }
53     }
54
55     ucitaj_matricu(matrica, n, m);
56
57     ispisi_elemente_ispod_dijagonale(matrica, n, m);
58
59     /* Oslobadjamo dinamicki alociranu memoriju za matricu. Prvo
60        oslobadjamo prostor rezervisan za svaku vrstu */
61     for (j = 0; j < n; j++)
62         free(matrica[j]);
63
64     /* Zatim oslobadjamo memoriju za niz pokazivaca na vrste
65        matrice */
```

## 2 Pokazivači

---

```
67     free(matrica);
69     return 0;
}
```

### Rešenje 2.20

```
#include<stdio.h>
2
int main()
4 {
    printf("Hello pokazivaci!\n");
6     return 0;
}
```

### Rešenje 2.21

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

10     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
12         /* Ako je indeks vrste manji od indeksa kolone */
            if (i < j)
14                 /* element se nalazi iznad glavne dijagonale pa ga
                    polovimo */
16                 a[i][j] /= 2;
            else
18                 /* Ako je indeks vrste veci od indeksa kolone */
                if (i > j)
20                     /* element se nalazi ispod glavne dijagonale pa ga
                        dupliramo */
22                     a[i][j] *= 2;
}

24
/* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
26 sporedne dijagonale */
float zbir_ispod_sporedne_dijagonale(float **m, int n)
28 {
    int i, j;
30     float zbir = 0;

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

```
34     for (j = 0; j < n; j++)
        /* Ukoliko je zbir indeksa vrste i indeksa kolone elementa
36         veci od n-1, to znaci da se element nalazi ispod
           sporedne dijagonale */
        if (i + j > n - 1)
38             zbir += fabs(m[i][j]);

40     return zbir;
}

42 /* Funkcija ucitava elemente kvadratne matrice dimenzije n iz
44     zadate datoteke */
void ucitaj_matricu(FILE * ulaz, float **m, int n)
46 {
    int i, j;

48     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
50             fscanf(ulaz, "%f", &m[i][j]);
52 }

54 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
    standardni izlaz */
56 void ispisi_matricu(float **m, int n)
{
58     int i, j;

60     for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
62             printf("%.2f ", m[i][j]);
        printf("\n");
64     }
}

66 /* Funkcija alokira memoriju za kvadratnu matricu dimenzije n */
68 float **alociraj_memoriju(int n)
{
70     int i, j;
    float **m;

72     m = (float **) malloc(n * sizeof(float *));
74     if (m == NULL) {
        fprintf(stderr, "malloc(): Neuspela alokacija\n");
76         exit(EXIT_FAILURE);
    }

78     /* Za svaku vrstu matrice */
80     for (i = 0; i < n; i++) {
        /* Alociramo memoriju */
82         m[i] = (float *) malloc(n * sizeof(float));

84         /* Proveravamo da li je doslo do greske pri alokaciji */
```

```

    if (m[i] == NULL) {
86      /* Ako jeste, ispisujemo poruku */
      printf("malloc(): neuspela alokacija memorije!\n");
88
      /* Oslobadjamo memoriju zauzetu do ovog koraka */
90      for (j = 0; j < i; j++)
        free(m[i]);
92      free(m);
      exit(EXIT_FAILURE);
94    }
  }
96  return m;
}

98  /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom
100   dimenzije n */
void oslobodi_memoriju(float **m, int n)
102 {
    int i;
104
    for (i = 0; i < n; i++)
106      free(m[i]);
    free(m);
108 }

110 int main(int argc, char *argv[])
{
112     FILE *ulaz;
    float **a;
114     int n;

116     /* Ako korisnik nije uneo trazene argumente, prijavljujemo
       gresku */
118     if (argc < 2) {
        printf("Greska: ");
120        printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_dat.\n", argv[0]);
122        exit(EXIT_FAILURE);
    }

124
    /* Otvaramo datoteku za citanje */
126    ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
128        fprintf(stderr, "Greska: ");
        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
130                argv[1]);
        exit(EXIT_FAILURE);
132    }

134    /* citamo dimenziju matrice */
    fscanf(ulaz, "%d", &n);
136
```



```

138  /* Alociramo memoriju */
    a = alociraj_memoriju(n);

140  /* Ucitavamo elemente matrice */
    ucitaj_matricu(ulaz, a, n);

142

144  float zbir = zbir_ispod_sporedne_dijagonale(a, n);

    /* Pozivamo funkciju za modifikovanje elemenata */
146  izmeni(a, n);

148  /* Ispisujemo rezultat */
    printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
150  printf("je %.2f.\n", zbir);

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

154

156  /* Oslobadjamo memoriju */
    oslobodi_memoriju(a, n);

158  /* Zatvaramo datoteku */
    fclose(ulaz);

160

162  /* i prekidamo sa izvršavanjem programa */
    return 0;
}

```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

Rešenje 2.26

```

2  #include <stdio.h>
   #include <stdlib.h>
4  #include <math.h>
   #include <string.h>
6
   /* NAPOMENA: Zaglavlje math.h sadrzi deklaracije raznih
8  matematičkih funkcija. dIzmeu ostalog, to su čsledee
   funkcije: double sin(double x); double cos(double x); double

```

```
10     tan(double x); double asin(double x); double acos(double x);
    double atan(double x); double atan2(double y, double x);
12     double sinh(double x); double cosh(double x); double
    tanh(double x); double exp(double x); double log(double x);
14     double log10(double x); double pow(double x, double y);
    double sqrt(double x); double ceil(double x); double
16     floor(double x); double fabs(double x); */

18 /* Funkcija tabela() prihvata granice intervala a i b, broj
    ekvidistantnih čtaaka n, kao i čpokaziva f koji pokazuje na
20     funkciju koja prihvata double argument, i čvraa double
    vrednost. Za tako datu funkciju ispisuje njene vrednosti u
22     intervalu [a,b] u n ekvidistantnih čtaaka intervala */
void tabela(double a, double b, int n, double (*fp) (double))
24 {
    int i;
26     double x;

28     printf("-----\n");
    for (i = 0; i < n; i++) {
30         x = a + i * (b - a) / (n - 1);
        printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
32     }
    printf("-----\n");
34 }

36 /* Umesto da koristimo stepenu funkciju iz zaglavlja math.h ->
    pow(a,2) čpozivaemo čkorisniku sqr(a) */
38 double sqr(double a)
    {
40     return a * a;
    }

42
44 int main(int argc, char *argv[])
    {
46     double a, b;
    int n;
    /* Imena funkcija koja ćemo navoditi su čkraa ili čtano
48     duga 5 karaktera */
    char ime_fje[6];
50     /* Pokazivac na funkciju koja ima jedan argument tipa double i
    povratnu vrednost istog tipa */
52     double (*fp) (double);

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

```
62 }
63
64 /* Niska ime_fje žsadi ime žtraene funkcije koja je
   navedena u komandnoj liniji */
65 strcpy(ime_fje, argv[1]);
66
67 /* Inicijalizujemo čpokaziva na funkciju koja treba da se
   tabelira */
68 if (strcmp(ime_fje, "sin") == 0)
69     fp = &sin;
70 else if (strcmp(ime_fje, "cos") == 0)
71     fp = &cos;
72 else if (strcmp(ime_fje, "tan") == 0)
73     fp = &tan;
74 else if (strcmp(ime_fje, "atan") == 0)
75     fp = &atan;
76 else if (strcmp(ime_fje, "acos") == 0)
77     fp = &acos;
78 else if (strcmp(ime_fje, "asin") == 0)
79     fp = &asin;
80 else if (strcmp(ime_fje, "exp") == 0)
81     fp = &exp;
82 else if (strcmp(ime_fje, "log") == 0)
83     fp = &log;
84 else if (strcmp(ime_fje, "log10") == 0)
85     fp = &log10;
86 else if (strcmp(ime_fje, "sqrt") == 0)
87     fp = &sqrt;
88 else if (strcmp(ime_fje, "floor") == 0)
89     fp = &floor;
90 else if (strcmp(ime_fje, "ceil") == 0)
91     fp = &ceil;
92 else if (strcmp(ime_fje, "sqr") == 0)
93     fp = &sqr;
94 else {
95     printf("Program jos uvek ne podrzava trazenu funkciju!\n");
96     exit(EXIT_SUCCESS);
97 }
98
99
100 printf("Unesite krajeve intervala:\n");
101 scanf("%lf %lf", &a, &b);
102
103
104 printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
105 printf("(ukljucujuci krajeve intervala)?\n");
106 scanf("%d", &n);
107
108 /* Mreza mora da čukljuuje bar krajeve intervala, tako da se
   mora uneti broj veci od 2 */
109 if (n < 2) {
110     fprintf(stderr, "Broj čtaaka žmree mora biti bar 2!\n");
111     exit(EXIT_FAILURE);
112 }
```

```
114      /* Ispisujemo ime funkcije */
116      printf("      x %10s(x)\n", ime_fje);

118      /* dProsleujemo funkciji tabela() funkciju zadatu kao
        argument komandne linije */
120      tabela(a, b, n, fp);

122      exit(EXIT_SUCCESS);
    }
```

Rešenje [2.27](#)

Rešenje [2.28](#)

Rešenje [2.29](#)

## Glava 3

# Algoritmi pretrage i sortiranja

### 3.1 Pretraživanje

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

- (a) Napisati funkciju koja vrši linearnu pretragu niza celih brojeva  $\mathbf{a}$ , dužine  $\mathbf{n}$ , tražeći u njemu broj  $\mathbf{x}$ .
- (b) Napisati funkciju koja vrši binarnu pretragu sortiranog niza  $\mathbf{a}$ , dužine  $\mathbf{n}$ , tražeći u njemu broj  $\mathbf{x}$ .
- (c) Napisati funkciju koja vrši interpoacionu pretragu sortiranog niza  $\mathbf{a}$ , dužine  $\mathbf{n}$ , tražeći u njemu broj  $\mathbf{x}$ .

Napisati i program koji generiše slučajni rastući niz dimenzije  $\mathbf{n}$  (prvi argument komandne linije, ne veći od 1000000), i u njemu već napisanim funkcijama traži element  $\mathbf{x}$  (drugi argument komandne linije). Potrebna vremena za izvršavanje ovih funkcija upisati u fajl `vremena.txt`.

#### Test 1

```
Poziv: ./a.out 1000000 235423
```

```
Izlaz:
```

```
Linearna pretraga  
Element nije u nizu
```

```
-----  
Binarna pretraga  
Element nije u nizu
```

```
-----  
Interpolaciona pretraga  
Element nije u nizu
```

#### Test 2

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

```
Izlaz:
```

```
Linearna pretraga  
Element nije u nizu
```

```
-----  
Binarna pretraga  
Element nije u nizu
```

```
-----  
Interpolaciona pretraga  
Element nije u nizu
```

[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. Pretpostaviti da niz brojeva koji se unosi neće biti duži od 1024 elemenata. Prvo se unosi broj koji se traži, a zatim sortirani elementi niza sve do kraja ulaza.

#### Upotreba programa 1

```
Interakcija programa:
```

```
Unesite traženi broj: 11  
Unesite sortirani niz elemenata:  
2 5 6 8 10 11 23
```

```
Izlaz:
```

```
Linearna pretraga  
Pozicija elementa je 5.
```

```
Binarna pretraga  
Pozicija elementa je 5.
```

```
Interpolaciona pretraga  
Pozicija elementa je 5.
```

#### Upotreba programa 2

```
Interakcija programa:
```

```
Unesite traženi broj: 14  
Unesite sortirani niz elemenata:  
10 32 35 43 66 89 100
```

```
Izlaz:
```

```
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 studenta čije informacije se potom prikazuju na ekranu. Zatim, korisnik učitava prezime studenta i prikazuju mu se informacije o prvom studentu sa unetim prezimenom. U slučaju neuspješnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što bolje manje složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

*Upotreba programa 1*

```

Datoteka:
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 programa:
Unesite indeks studenta cije informacije zelite: 20140076
Indeks: 20140076, Ime i prezime: Sonja Stevanovic
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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
Poziv: ./a.out dat.txt -x	Poziv: ./a.out dat.txt	Poziv: ./a.out dat.txt -y
Datoteka:	Datoteka:	Datoteka:
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
Izlaz:	Izlaz:	Izlaz:
-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: Korisiti 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 niz elemenata koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

*Upotreba programa 1*

```
|| Interakcija programa:
|| Unesi rastuce sortiran
|| niz celih brojeva:
|| -151 -44 5 12 13 15
|| Izlaz:
|| 2
```

*Upotreba programa 2*

```
|| Interakcija programa:
|| Unesi rastuce sortiran
|| niz celih brojeva:
|| -100 -15 -11 -8 -7 -5
|| Izlaz:
|| -1
```

*Upotreba programa 3*

```
|| Interakcija programa:
|| Unesi rastuce sortiran
|| niz celih brojeva:
|| -100 -15 0 13 55 124
|| 258 315 516 7000
|| Izlaz:
|| 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 niz elemenata koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

*Upotreba programa 1*

```
|| Interakcija programa:
|| Unesi opadajuce
|| sortiran niz celih
|| brojeva:
|| 151 44 5 -12 -13 -15
|| Izlaz:
|| 3
```

*Upotreba programa 2*

```
|| Interakcija programa:
|| Unesi opadajuce
|| sortiran niz celih
|| brojeva:
|| 100 55 15 0 -15 -124
|| -155 -258 -315 -516
|| Izlaz:
|| 4
```

*Upotreba programa 3*

```
|| Interakcija programa:
|| Unesi opadajuce
|| sortiran niz celih
|| brojeva:
|| 100 15 11 8 7 5 4 3 2
|| Izlaz:
|| -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 broj učitava sa standardnog ulaza, a logaritam ispisuje na standardnom izlazu.

<i>Upotreba programa 1</i>	<i>Upotreba programa 2</i>	<i>Upotreba programa 3</i>
<pre> Interakcija programa:   Unesi pozitivan ceo     broj: 4 Izlaz:   2 2 </pre>	<pre> Interakcija programa:   Unesi pozitivan ceo     broj: 17 Izlaz:   4 4 </pre>	<pre> Interakcija programa:   Unesi pozitivan ceo     broj: 1031 Izlaz:   10 10 </pre>

[Rešenje 3.9]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> Ulaz:   2   2 0 2 1   1 2 2 2 Izlaz:   26.57 </pre>	<pre> Ulaz:   2   1 0 1 1   0 1 1 1 Izlaz:   45 </pre>	<pre> Ulaz:   3   1 0 1 1   2 0 2 1   1 2 2 2 Izlaz:   26.57 </pre>

**Zadatak 3.11** Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime (uređen u rastućem poretку prezimena) sa manje od 10 elemenata, a zatim se učitava jedan karakter i pronalazi (bibliotečkom funkcijom `bsearch`) i štampa jedna struktura iz niza osoba čije prezime počinje tim karakterom. Ako takva osoba ne postoji, štampati `-1` na standardnom izlazu.

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>   Ulaz:    R    Izlaz:    Dusko Radovic</pre>	<pre>   Ulaz:    E    Izlaz:    Dobrica Eric</pre>	<pre>   Ulaz:    A    Izlaz:    Mika Antic</pre>

## 3.2 Sortiranje

**Zadatak 3.12** U datom nizu brojeva pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, i neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati njihovu razliku. *Uputstvo: Prvo sortirati niz.*

<i>Upotreba programa 1</i>	<i>Upotreba programa 2</i>	<i>Upotreba programa 3</i>
<pre>   Interakcija programa:    Unesite elemente niza:    23 64 123 76 22 7    Izlaz:    1</pre>	<pre>   Interakcija programa:    Unesite elemente niza:    21 654 65 123 65 12 61    Izlaz:    0</pre>	<pre>   Interakcija programa:    Unesite elemente niza:    34 30    Izlaz:    4</pre>

[Rešenje 3.12]

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

<i>Upotreba programa 1</i>	<i>Upotreba programa 2</i>	<i>Upotreba programa 3</i>
<pre> Interakcija programa: Unesite prvu nisku:     anagram Unesite drugu nisku:     ramgana Izlaz: jesu </pre>	<pre> Interakcija programa: Interakcija programa:     Unesite prvu nisku:         anagram     Unesite drugu nisku:         anagrm Izlaz: nisu </pre>	<pre> Interakcija programa: Interakcija programa:     Unesite prvu nisku: test     Unesite drugu nisku: tset Izlaz: jesu </pre>

[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.*

<i>Upotreba programa 1</i>	<i>Upotreba programa 2</i>	<i>Upotreba programa 3</i>
<pre> Interakcija programa: Unesite elemente niza: 4 23 5 2 4 6 7 34 6 4 5 Izlaz: 4 </pre>	<pre> Interakcija programa: Unesite elemente niza: 2 4 6 2 6 7 99 1 Izlaz: 2 </pre>	<pre> Interakcija programa: Unesite elemente niza: 123 Izlaz: 123 </pre>

[Rešenje 3.14]

**Zadatak 3.15** Napisati funkciju koja proverava da li u datom nizu postoje dva elementa kojima je zbir zadati ceo broj. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz (pretpostaviti da za niz neće biti uneto više od 256 brojeva). Elementi niza se unose sve do kraja ulaza. *Uputstvo: Prvo sortirati niz.*

<i>Upotreba programa 1</i>	<i>Upotreba programa 2</i>	<i>Upotreba programa 3</i>
<pre> Interakcija programa: Unesite trazeni zbir: 34 Unesite elemente niza: 134 4 1 6 30 23 Izlaz: da </pre>	<pre> Interakcija programa: Unesite trazeni zbir: 12 Unesite elemente niza: 53 1 43 3 56 13 Izlaz: ne </pre>	<pre> Interakcija programa: Unesite trazeni zbir: 52 Unesite elemente niza: 52 Izlaz: ne </pre>

[Rešenje 3.15]

### 3 Algoritmi pretrage i sortiranja

**Zadatak 3.16** Napraviti biblioteku `sort.h` i `sort.c` koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži `selection`, `merge`, `quick`, `bubble`, `insertion` i `shell sort`. Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Vreme meriti programom `time`. Analizirati porast vremena sa porastom dimenzije `n`.

Upotreba programa 1	Upotreba programa 2	Upotreba programa 3
<pre>Poziv: time a.out 100000 -i -o Izlaz: real    0m17.631s user    0m17.604s sys     0m0.000s</pre>	<pre>Poziv: time a.out 100000 -b -r Izlaz: real    0m0.005s user    0m0.004s sys     0m0.000s</pre>	<pre>Poziv: time a.out 100000 -s Izlaz: real    0m0.071s user    0m0.068s sys     0m0.000s</pre>

[Rešenje 3.16]

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

Upotreba programa 1	Upotreba programa 2
<pre>Interakcija programa: Unesite elemente prvog niza: 3 6 7 11 14 35 0 Unesite elemente drugog niza: 3 5 8 0 Izlaz: 3 3 5 6 7 8 11 14 35</pre>	<pre>Interakcija programa: Unesite elemente prvog niza: 1 4 7 0 Unesite elemente drugog niza: 9 11 23 54 75 0 Izlaz: 1 4 7 9 11 23 54 75</pre>

[Rešenje 3.17]

**Zadatak 3.18** Napisati program koji čita sadržaj dveju datoteka od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima 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
Ulazne datoteke:
    prvi-deo.txt:                drugi-deo.txt:
    Andrija Petrovic            Aleksandra Cvetic
    Anja Ilic                   Bojan Golubovic
    Ivana Markovic              Dragan Markovic
    Lazar Micic                 Filip Dukic
    Nenad Brankovic             Ivana Stankovic
    Sofija Filipovic            Marija Stankovic
    Vladimir Savic              Ognjen Peric
    Uros Milic

Izlazna datoteka (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 3.18]

**Zadatak 3.19** Napisati funkcije koje sortiraju niz struktura tačaka na osnovu sledećih kriterijuma:

- (a) njihovog rastojanja od koordinatnog početka,
- (b)  $x$  koordinata tačaka,
- (c)  $y$  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
Ulazna datoteka (in.txt):
 3 4
11 6
 7 3
 2 82
-1 6
Izlazna datoteka (out.txt):
-1 6
 2 82
 3 4
 7 3
11 6
```

#### Test 2

```
Poziv: ./a.out -o in.txt out.txt
Ulazna datoteka (on.txt):
 3 4
11 6
 7 3
 2 82
-1 6
Izlazna datoteka (out.txt):
 3 4
-1 6
 7 3
11 6
 2 82
```

[Rešenje 3.19]

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

#### Test 1

```
Ulazna datoteka (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

```
Ulazna datoteka (biracki-spisak.txt):
Milan Milicevic
Izlaz:
1
```

[Rešenje 3.20]

**Zadatak 3.21** Definisana je struktura podataka

```
typedef struct dete
{
    char ime[MAX_IME];
    char prezime[MAX_IME];
    unsigned godiste;
} Dete;
```

Napisati funkciju koja sortira niz dece po godištu, a kada su deca istog godišta, tada ih 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 deteta.

#### Test 1

```
Poziv: ./a.out in.txt out.txt
Ulazna datoteka (in.out):
  Petar Petrovic 2007
  Milica Antonic 2008
  Ana Petrovic 2007
  Ivana Ivanovic 2009
  Dragana Markovic 2010
  Marija Antic 2007
Izlazna datoteka (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
Ulazna datoteka (in.out):
  Milijana Maric 2009
Izlazna datoteka (out.txt):
  Milijana Maric 2009
```

**Zadatak 3.22** Napisati funkciju koja sortira niz niski po broju suglasnika u niski. Ukoliko reči imaju isti broj suglasnika tada sortirati ih po dužini niske, a ukoliko su i dužine jednake onda leksikografski. 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

```
Ulazna datoteka (niske.txt):
  ana petar andjela milos nikola aleksandar ljubica matej milica
Izlaz:
  ana matej milos petar milica nikola andjela ljubica aleksandar
```

[Rešenje 3.22]

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

#### Upotreba programa 1

```
Ulazna datoteka (artikli.txt):
1001 Keks Jaffa 120
2530 Napolitanke Bambi 230
0023 Medeno_srce Pionir 150
2145 Pardon Marbo 70

Interakcija programa:
Asortiman:
KOD          Naziv artikla      Ime proizvođača      Cena
      23          Medeno_srce      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: Medeno_srce Pionir      150.00
Unesite kod artikla [ili 0 za prekid]: 0

      UKUPNO: 270.00 dinara.

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

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

      UKUPNO: 230.00 dinara.

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

Kraj rada kase!
```

[Rešenje 3.23]

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



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

### Test 1

Ulazna datoteka (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
```

Izlazna datoteka (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
```

Izlazna datoteka (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
```

Izlazna datoteka (dat3.txt):

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

[Rešenje 3.24]

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

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

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

### 3 Algoritmi pretrage i sortiranja

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

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

Test 1	Test 2	Test 3
<code>Poziv: ./a.out</code> <code>Ulazna datoteka:</code> 5 Ana - Nebo, 2342 Laza - Oblaci, 29 Pera - Ptice, 327 Jelena - Sunce, 92321 Mika - Kisa, 5341 <code>Izlaz:</code> Jelena - Sunce, 92321 Mika - Kisa, 5341 Ana - Nebo, 2342 Pera - Ptice, 327 Laza - Oblaci, 29	<code>Poziv: ./a.out -i</code> <code>Ulazna datoteka:</code> 5 Ana - Nebo, 2342 Laza - Oblaci, 29 Pera - Ptice, 327 Jelena - Sunce, 92321 Mika - Kisa, 5341 <code>Izlaz:</code> Ana - Nebo, 2342 Jelena - Sunce, 92321 Laza - Oblaci, 29 Mika - Kisa, 5341 Pera - Ptice, 327	<code>Poziv: ./a.out -n</code> <code>Ulazna datoteka:</code> 5 Ana - Nebo, 2342 Laza - Oblaci, 29 Pera - Ptice, 327 Jelena - Sunce, 92321 Mika - Kisa, 5341 <code>Izlaz:</code> Mika - Kisa, 5341 Ana - Nebo, 2342 Laza - Oblaci, 29 Pera - Ptice, 327 Jelena - Sunce, 92321

[Rešenje 3.25]

**\*\* Zadatak 3.26** Razmatrajmo dve operacije: operacija U je unos novog broja x, a operacija N određivanje n-tog po veličini od unetih brojeva. Implementirati program koji izvršava ove operacije. Može postojati najviše 100000 operacija unosa, a uneti elementi se mogu ponavljati, pri čemu se i ponavljanja računaju prilikom brojanja. *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.

Test 1  
`Ulaz: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5`  
`Izlaz: 0 2 8 2 6`

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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

#### Upotreba programa 1

```
Interakcija programa:
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
```

#### Upotreba programa 2

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

[Rešenje 3.28]

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

#### Upotreba programa 1

```
Interakcija programa:
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
```

[Rešenje 3.29]

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

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

Niske se učitavaju iz fajla `niske.txt`, neće ih biti više od 1000 i svaka će biti dužine najviše 30 karaktera. Program prvo leksikografski sortira niz, primenjuje binarnu pretragu (`bsearch`) zarad traženja niske unete sa standardnog ulaza, a potom linearnu pretragu koristeći funkciju `lfind`. Na kraju, niske bivaju sortirane po dužini. Rezultate svih sortiranja i pretraga ispisati na standardnom izlazu.

#### Upotreba programa 1

```
Ulazna datoteka (niske.txt):
ana petar andjela milos nikola aleksandar ljubica matej milica
Interakcija programa:
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
Niska "matej" je pronadjena u nizu na poziciji 4
Niske sortirane po duzini:
ana matej milos petar milica nikola andjela ljubica aleksandar
```

[Rešenje 3.30]

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

[Rešenje 3.31]

**Zadatak 3.32** Napisati program koji korišćenjem biblioteke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće.

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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.

#### Upotreba programa 1

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

[Rešenje 3.32]

**Zadatak 3.33** Uraditi zadatak 3.13, ali korišćenjem bibliotečke `qsort` funkcije.

[Rešenje 3.33]

**Zadatak 3.34** Napisati program koji sa standardnog ulaza učitava prvo ceo broj  $n$  ( $n \leq 10$ ), a zatim niz  $S$  od  $n$  stringova (maksimalna dužina stringa je 31 karakter). Sortirati niz  $S$  (bibliotečkom funkcijom `qsort`) i proveriti da li u njemu ima identičnih stringova.

#### Upotreba programa 1

```
Interakcija programa:
  Unesite broj niski: 4
  Unesite niske:
    prog search sort search
Izlaz:
  ima
```

#### Upotreba programa 2

```
Interakcija programa:
  Unesite broj niski: 3
  Unesite niske:
    test kol ispit
Izlaz:
  nema
```

#### Upotreba programa 3

```
Interakcija programa:
  Unesite broj niski: 5
  Unesite niske:
    a ab abc abcd abcde
Izlaz:
  nema
```

[Rešenje 3.34]

**Zadatak 3.35** Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr97125`, `mm09001`), ime i prezime i broj poena. I ime i prezime neće biti duže od 20 karaktera. Napisati program koji sortira (korišćenjem funkcije `qsort`) studente po broju poena opadajuće (ukoliko je prisutna opcija `-p`) ili po nalogu (ukoliko je prisutna opcija `-n`). Studenti se po nalogu sortiraju tako što se sortiraju na osnovu godine, zatim na osnovu smera, i na kraju na osnovu broja indeksa. Sortirane studente upisati u datoteku `izlaz.txt`. Ukoliko je u komandnoj liniji uz opciju `-n` naveden i nalog nekog studenta, funkcijom `bsearch` potražiti i prijaviti broj poena studenta sa tim nalogom.

#### Test 1

```
Poziv: ./a.out -n mm13321
Ulazna datoteka (studenti.txt):
  mr14123 Marko Antic 20
  mm13321 Marija Radic 12
  ml13011 Ivana Mitrovic 19
  ml13066 Pera Simic 15
  mv14003 Jovan Jovanovic 17
Izlazna datoteka (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
Ulazna datoteka (studenti.txt):
  mr14123 Marko Antic 20
  mm13321 Marija Radic 12
  ml13011 Ivana Mitrovic 19
  ml13066 Pera Simic 15
  mv14003 Jovan Jovanovic 17
Izlazna datoteka (izlaz.txt):
  mr14123 Marko Antic 20
  ml13011 Ivana Mitrovic 19
  mv14003 Jovan Jovanovic 17
  ml13066 Pera Simic 15
  mm13321 Marija Radic 12
```

[Rešenje 3.35]

**Zadatak 3.36** Definisana je struktura:

```
typedef struct { int dan; int mesec; int godina; } Datum;
```

Napisati funkciju koja poredi dva datuma i program koji učitava datume iz datoteke koja se zadaje kao prvi argument komandne linije (ne više od 128 datuma),

### 3.3 Bibliotečke funkcije pretrage i sortiranja

sortira ih pozivajući funkciju `qsort` iz standardne biblioteke i potom pozivanjem funkcije `bsearch` iz standardne biblioteke proverava da li datumi učitani sa standardnog ulaza (sve do kraja ulaza) postoje među prethodno unetim datumima.

#### Test 1

<b>Poziv:</b> <code>./a.out datoteka.txt</code>		
<b>Datoteka:</b>	<b>Ulaz:</b>	<b>Izlaz:</b>
1.1.2013.	13.12.2016.	<code>postoji</code>
13.12.2016.	10.5.2015.	<code>ne postoji</code>
11.11.2011.	5.2.2009.	<code>postoji</code>
3.5.2015.		
5.2.2009.		

**Zadatak 3.37** 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.

#### Upotreba programa 1

```
Interakcija programa:
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
```

#### Upotreba programa 2

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

[Rešenje 3.37]

**Zadatak 3.38** Za 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.

#### Upotreba programa 1

```
Interakcija programa:
  Unesite dimenziju matrice: 2
  Unesite elemente matrice po vrstama:
    6 -5
    -4 3
  Sortirana matrica je:
    -5 6
    3 -4
```

#### Upotreba programa 2

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

## 3.4 Rešenja

### Rešenje 3.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
7    -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
10    njemu element x. Pretraga se vrši prostom iteracijom kroz niz. Ako
11    se element pronadje funkcija vraca indeks pozicije na kojoj je
12    pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
13    pronadjen u nizu, funkcija vraca -1, kao indikator neuspesne
14    pretrage. */
15 int linearna_pretraga(int a[], int n, int x)
16 {
17     int i;
18     for (i = 0; i < n; i++)
19         if (a[i] == x)
20             return i;
21     return -1;
22 }
23
24 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
25    pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
26 int binarna_pretraga(int a[], int n, int x)
27 {
28     int levi = 0;
29     int desni = n - 1;
30     int srednji;
```



```

31  /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
33      /* Srednji indeks je njihova aritmeticka sredina */
      srednji = (levi + desni) / 2;
35      /* Ako je element sa sredisnjim indeksom veci od x, tada se x
        mora nalaziti u levoj polovini niza */
37      if (x < a[srednji])
          desni = srednji - 1;
39      /* Ako je element sa sredisnjim indeksom manji od x, tada se x
        mora nalaziti u desnoj polovini niza */
41      else if (x > a[srednji])
          levi = srednji + 1;
43      else
          /* 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

```

### 3 Algoritmi pretrage i sortiranja

---

```
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
          /* Ako je element sa indeksom srednji jednak trazenom, onda se
          pretraga zavrшава na poziciji srednji */
          return srednji;
97      }
99      /* U slucaju neuspesne pretrage vraca se -1 */
101     return -1;
103 }

104 /* Funkcija main */
105 int main(int argc, char **argv)
106 {
107     int a[MAX];
108     int n, i, x;
109     struct timespec time1, time2, time3, time4, time5, time6;
110     FILE *f;
111
112     /* Provera argumenata komandne linije */
113     if (argc != 3) {
114         fprintf(stderr,
115             "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
116         ;
117         exit(EXIT_FAILURE);
118     }
119
120     /* Dimenzija niza */
121     n = atoi(argv[1]);
122     if (n > MAX || n <= 0) {
123         fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
124         exit(EXIT_FAILURE);
125     }
126
127     /* Broj koji se trazi */
128     x = atoi(argv[2]);
129
130     /* Elementi niza se generisu slucajno, tako da je svaki sledeci
131     veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
132     pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
133     izgledao, seed se postavlja na tekuce vreme u sekundama od Nove
```

```

133     godine 1970. random()%100 daje brojeve izmedju 0 i 99 */
134     srandom(time(NULL));
135     for (i = 0; i < n; i++)
136         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
137
138     /* Linearna pretraga */
139     printf("Linearna pretraga\n");
140     /* Vreme proteklo od Nove godine 1970 */
141     clock_gettime(CLOCK_REALTIME, &time1);
142     i = linearna_pretraga(a, n, x);
143     /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
144         linearnu pretragu */
145     clock_gettime(CLOCK_REALTIME, &time2);
146     if (i == -1)
147         printf("Element nije u nizu\n");
148     else
149         printf("Element je u nizu na poziciji %d\n", i);
150     printf("-----\n");
151
152     /* Binarna pretraga */
153     printf("Binarna pretraga\n");
154     clock_gettime(CLOCK_REALTIME, &time3);
155     i = binarna_pretraga(a, n, x);
156     clock_gettime(CLOCK_REALTIME, &time4);
157     if (i == -1)
158         printf("Element nije u nizu\n");
159     else
160         printf("Element je u nizu na poziciji %d\n", i);
161     printf("-----\n");
162
163     /* Interpolaciona pretraga */
164     printf("Interpolaciona pretraga\n");
165     clock_gettime(CLOCK_REALTIME, &time5);
166     i = interpolaciona_pretraga(a, n, x);
167     clock_gettime(CLOCK_REALTIME, &time6);
168     if (i == -1)
169         printf("Element nije u nizu\n");
170     else
171         printf("Element je u nizu na poziciji %d\n", i);
172     printf("-----\n");
173
174     /* Podaci o izvrsavanju programa bivaju upisani u log fajl */
175     if ((f = fopen("vremena.txt", "a")) == NULL) {
176         fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
177         exit(EXIT_FAILURE);
178     }
179
180     fprintf(f, "Dimenzija niza od %d elemenata.\n", n);
181     fprintf(f, "\tLinearna pretraga:%10ld ns\n",
182             (time2.tv_sec - time1.tv_sec) * 1000000000 +
183             time2.tv_nsec - time1.tv_nsec);
184     fprintf(f, "\tBinarna: %19ld ns\n",

```

```
185         (time4.tv_sec - time3.tv_sec) * 1000000000 +
            time4.tv_nsec - time3.tv_nsec);
187     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
            (time6.tv_sec - time5.tv_sec) * 1000000000 +
189         time6.tv_nsec - time5.tv_nsec);

191     /* Zatvaranje datoteke */
    fclose(f);
193
    return 0;
195 }
```

#### Rešenje 3.2

```
1  #include <stdio.h>

3  int lin_pretgraga_rek_sufiks(int a[], int n, int x)
{
5     int tmp;
    /* Izlaz iz rekurzije */
7     if (n <= 0)
        return -1;
9     /* Ako je prvi element trazeni */
    if (a[0] == x)
11        return 0;
    /* Pretraga ostatka niza */
13    tmp = lin_pretgraga_rek_sufiks(a + 1, n - 1, x);
    return tmp < 0 ? tmp : tmp + 1;
15 }

17 int lin_pretgraga_rek_prefiks(int a[], int n, int x)
{
19     /* Izlaz iz rekurzije */
    if (n <= 0)
21        return -1;
    /* Ako je poslednji element trazeni */
23    if (a[n - 1] == x)
        return n - 1;
25    /* Pretraga ostatka niza */
    return lin_pretgraga_rek_prefiks(a, n - 1, x);
27 }

29 int bin_pretgraga_rek(int a[], int l, int d, int x)
{
31     int srednji;
    if (l > d)
33        return -1;
    /* Sredisnja pozicija na kojoj se trazi vrednost x */
35    srednji = (l + d) / 2;
    /* Ako je element na sredisnjoj poziciji trazeni */
37    if (a[srednji] == x)
```

```

    return srednji;
39 /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
    pretrazuje se desna polovina niza */
41 if (a[srednji] < x)
    return bin_pretgraga_rek(a, srednji + 1, d, x);
43 /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
    pretrazuje se leva polovina niza */
45 else
    return bin_pretgraga_rek(a, l, srednji - 1, x);
47 }

49 int interp_pretgraga_rek(int a[], int l, int d, int x)
51 {
    int p;
53 if (x < a[l] || x > a[d])
    return -1;
55 if (a[d] == a[l])
    return l;
57 /* Pozicija na kojoj se trazi vrednost x */
    p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
59 if (a[p] == x)
    return p;
61 if (a[p] < x)
    return interp_pretgraga_rek(a, p + 1, d, x);
63 else
    return interp_pretgraga_rek(a, l, p - 1, x);
65 }

67 #define MAX 1024

69 int main()
71 {
    int a[MAX];
    int x;
    int i, indeks;

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

79 /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
    korisnik pritisne CTRL+D za naznaku kraja */
81 i = 0;
    printf("Unesite sortiran niz elemenata: ");
83 while (scanf("%d", &a[i]) == 1) {
    i++;
85 }

87 /* Linearna pretraga */
    printf("Linearna pretraga\n");
89 indeks = lin_pretgraga_rek_sufiks(a, i, x);

```

### 3 Algoritmi pretrage i sortiranja

```
91     if (indeks == -1)
92         printf("Element se ne nalazi u nizu.\n");
93     else
94         printf("Pozicija elementa je %d.\n", indeks);
95
96     /* Binarna pretraga */
97     printf("Binarna pretraga\n");
98     indeks = bin_pretgraga_rek(a, 0, i - 1, x);
99     if (indeks == -1)
100         printf("Element se ne nalazi u nizu.\n");
101     else
102         printf("Pozicija elementa je %d.\n", indeks);
103
104     /* Interpolaciona pretraga */
105     printf("Interpolaciona pretraga\n");
106     indeks = interp_pretgraga_rek(a, 0, i - 1, x);
107     if (indeks == -1)
108         printf("Element se ne nalazi u nizu.\n");
109     else
110         printf("Pozicija elementa je %d.\n", indeks);
111
112     return 0;
113 }
```

#### Rešenje 3.3

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
7
8 /* 0 svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
9
10 typedef struct {
11     /* Indeks mora biti tipa long jer su podaci u datoteci preveliki za
12      int, npr. 20140123 */
13     long indeks;
14     char ime[MAX_DUZINA];
15     char prezime[MAX_DUZINA];
16 } Student;
17
18 /* Učitani niz studenata će biti sortiran rastuće prema indeksu, jer
   su studenti u datoteci već sortirani. Iz tog razloga pretraga po
   indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
   jer nema garancije da postoji uredjenje po prezimenu. */
19
20
21
22 /* Funkcija traži u sortiranom nizu studenata a[] dužine n studenata
   sa indeksom x i vraća indeks pozicije nadjenog člana niza ili -1,
   ako element nije pronađen. */
23
24
```

```
26 int binarna_pretraga(Student a[], int n, long 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         /* Racuna se srednja pozicija */
34         srednji = (levi + desni) / 2;
35         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
36            se trazeni indeks mora nalaziti u levoj polovini niza */
37         if (x < a[srednji].indeks)
38             desni = srednji - 1;
39         /* Ako je pak manji od trazenog, tada se on mora nalaziti u
40            desnoj polovini niza */
41         else if (x > a[srednji].indeks)
42             levi = srednji + 1;
43         else
44             /* Ako je jednak trazenom indeksu x, tada je pronadjen student
45                sa trazenom indeksom na poziciji srednji */
46             return srednji;
47     }
48     /* Ako nije pronadjen, vraca se -1 */
49     return -1;
50 }

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

62 /* Main funkcija mora imati argumente jer se ime datoteke prosledjuje
63    kao argument 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 trazeni_indeks = 0;
71     char trazeno_prezime[MAX_DUZINA];

72     /* Provera da li je korisnik prilikom poziva programa prosledio ime
73        datoteke sa informacijama o studentima */
74     if (argc != 2) {
75         fprintf(stderr,
```

### 3 Algoritmi pretrage i sortiranja

---

```
78         "Greska: Program se poziva sa %s ime_datoteke\n",
          argv[0]);
80     exit(EXIT_FAILURE);
81 }
82
83 /* Otvaranje datoteke */
84 fin = fopen(argv[1], "r");
85 if (fin == NULL) {
86     fprintf(stderr,
87         "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
88     exit(EXIT_FAILURE);
89 }
90
91 /* Citanje se vrši sve dok postoji red sa informacijama o studentu
92  */
93 i = 0;
94 while (1) {
95     if (i == MAX_STUDENATA)
96         break;
97     if (fscanf
98         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
99          dosije[i].prezime) != 3)
100         break;
101     i++;
102 }
103 br_studenata = i;
104
105 /* Nakon citanja, datoteka vise nije neophodna i odmah se zatvara
106  */
107 fclose(fin);
108
109 /* Unos indeksa koji se binarno trazi u nizu */
110 printf("Unesite indeks studenta cije informacije želite: ");
111 scanf("%ld", &trazen_indeks);
112 i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
113 /* Rezultat binarne pretrage */
114 if (i == -1)
115     printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
116 else
117     printf("Indeks: %ld, Ime i prezime: %s %s\n",
118           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
119
120 /* Unos prezimena koje se linearno trazi u nizu */
121 printf("Unesite prezime studenta cije informacije želite: ");
122 scanf("%s", trazeno_prezime);
123 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
124 /* Rezultat linearne pretrage */
125 if (i == -1)
126     printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
127 else
128     printf("Indeks: %ld, Ime i prezime: %s %s\n",
129           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
```



```

128     return 0;
130 }

```

### Rešenje 3.4

```

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

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

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

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
46     if (strcmp(a[0].prezime, x) == 0)
47         return 0;
48     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
49     return i >= 0 ? 1 + i : -1;
50 }

52 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
53 {
54     /* Ako je niz prazan, vraca se -1 */
55     if (n == 0)
56         return -1;
57     /* Ako se trazi poslednji student sa trazanim prezimenom, pocinje
58        se sa poslednjim studentom u nizu. */
59     if (strcmp(a[n - 1].prezime, x) == 0)
60         return n - 1;
61     return linearna_pretraga_rekurzivna(a, n - 1, x);
62 }

64 /* Main funkcija mora imate argumente jer se ime datoteke prosledjuje
65    kao argument komandne linije */
66 int main(int argc, char *argv[])
67 {
68     Student dosije[MAX_STUDENATA];
69     FILE *fin = NULL;
70     int i;
71     int br_studenata = 0;
72     long trazeni_indeks = 0;
73     char trazeno_prezime[MAX_DUZINA];
74
75     /* Provera da li je korisnik prilikom poziva prosledio ime datoteke
76        sa informacijama o studentima */
77     if (argc != 2) {
78         fprintf(stderr,
79             "Greska: Program se poziva sa %s ime_datoteke\n",
80             argv[0]);
81         exit(EXIT_FAILURE);
82     }
83
84     /* Otvaranje datoteke */
85     fin = fopen(argv[1], "r");
86     if (fin == NULL) {
87         fprintf(stderr,
88             "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
89         exit(EXIT_FAILURE);
90     }
91
92     /* Citanje se vrshi sve dok postoji sledeci red sa informacijama o
93        studentu */
94     i = 0;
95     while (1) {
96         if (i == MAX_STUDENATA)
97             break;
```

```

98     if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
100         dosije[i].prezime) != 3)
        break;
102     i++;
    }
104     br_studenata = i;

106     /* Nakon citanja datoteka nam vise nije neophodna i odmah se
        zatvara */
108     fclose(fin);

110     /* Unos indeksa koji se binarno trazi u nizu */
    printf("Unesite indeks studenta cijje informacije zelite: ");
112     scanf("%ld", &trazen_indeks);
    i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
114         trazen_indeks);

    if (i == -1)
116         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
118         printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
120

    /* Unos prezimena koje se linearno trazi u nizu */
122     printf("Unesite prezime studenta cijje informacije zelite: ");
    scanf("%s", trazeno_prezime);
124     i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
        trazeno_prezime);

126     if (i == -1)
        printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
128     else
        printf
130         ("Prvi takav student:\nIndeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132

    return 0;
134 }

```

### 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

```

### 3 Algoritmi pretrage i sortiranja

---

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

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];
    for (i = 1; i < n; i++) {
25      if (rastojanje(t[i]) < rastojanje(najbliza)) {
27          najbliza = t[i];
29      }
    }
    return najbliza;
31 }

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

49 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
    t dimenzije n */
51 Tacka najbliza_y_osi(Tacka t[], int n)
53 {
    Tacka najbliza;
    int i;
    najbliza = t[0];
    for (i = 1; i < n; i++) {
55      if (fabs(t[i].y) < fabs(najbliza.y)) {
57          najbliza = t[i];
59      }
    }
    return najbliza;
61 }
63 }
```

```
65 #define MAX 1024
67 int main(int argc, char *argv[])
68 {
69     FILE *ulaz;
70     Tacka tacke[MAX];
71     Tacka najbliza;
72     int i, n;
73
74     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75        ime datoteke sa tackama */
76     if (argc < 2) {
77         fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
79         return EXIT_FAILURE;
80     }
81
82     /* Otvaranje datoteke za citanje */
83     ulaz = fopen(argv[1], "r");
84     if (ulaz == NULL) {
85         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
87         return EXIT_FAILURE;
88     }
89
90     /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
91        tackama; i predstavlja indeks tekuce tacke */
92     i = 0;
93     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94         i++;
95     }
96     n = i;
97
98     /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
99        dodatnih argumenata */
100     if (argc == 2)
101         /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
102         najbliza = najbliza_koordinatnom(tacke, n);
103     /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
104        pitanju opcija -x */
105     else if (strcmp(argv[2], "-x") == 0)
106         /* Racuna se rastojanje u odnosu na x osu */
107         najbliza = najbliza_x_osi(tacke, n);
108     /* Ako je u pitanju opcija -y */
109     else if (strcmp(argv[2], "-y") == 0)
110         /* Racuna se rastojanje u odnosu na y osu */
111         najbliza = najbliza_y_osi(tacke, n);
112     else {
113         /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
114            korisnika i prekida se izvršavanje programa */
115         fprintf(stderr, "Pogresna opcija\n");
```

```
117     return EXIT_FAILURE;
118 }
119 /* Stapanje koordinata trazene tacke */
120 printf("%g %g\n", najbliza.x, najbliza.y);
121
122 /* Zatvaranje datoteke */
123 fclose(ulaz);
124
125 return 0;
126 }
```

#### Rešenje 3.6

```
1 #include <stdio.h>
2 #include <math.h>
3
4 /* Tacnost */
5 #define EPS 0.001
6
7 int main()
8 {
9     double l, d, s;
10
11     /* Kako je u pitanju interval [0, 2] leva granica je 0, a desna 2
12      */
13     l = 0;
14     d = 2;
15
16     /* Sve dok se ne pronadje trazena vrednost argumenta */
17     while (1) {
18         /* Polovi se interval */
19         s = (l + d) / 2;
20         /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
21            tacnosti, prekida se pretraga */
22         if (fabs(cos(s)) < EPS) {
23             break;
24         }
25         /* Ako je nula u levom delu intervala, nastavlja se pretraga na
26            [l, s] */
27         if (cos(l) * cos(s) < 0)
28             d = s;
29         else
30             /* Inace, na intervalu [s, d] */
31             l = s;
32     }
33
34     /* Stapanje vrednost trazene tacke */
35     printf("%g\n", s);
36
37     return 0;
38 }
```

```
}
```

### Rešenje 3.7

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

35   /* Unos niza */
   printf("Unesi rastuce sortiran niz celih brojeva: ");
37   while (scanf("%d", &niz[n]) == 1)
       n++;

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

43   return 0;
   }
```

#### Rešenje 3.8

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int prvi_manji_od_nule(int niz[], int n)
5 {
6     /* Granice pretrage */
7     int l = 0, d = n - 1;
8     int s;
9     /* Sve dok je leva manja od desne granice */
10    while (l <= d) {
11        /* Racuna se sredisnja pozicija */
12        s = (l + d) / 2;
13        /* Ako je broj na toj poziciji manji od nule, a eventualni njegov
14         prethodnik veci ili jednak nuli, pretraga se zavrшава */
15        if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
16            return s;
17        /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
18         niza */
19        if (niz[s] >= 0)
20            l = s + 1;
21        /* A inace leva */
22        else
23            d = s - 1;
24    }
25    return -1;
26 }
27
28 #define MAX 256
29
30 int main()
31 {
32     int niz[MAX];
33     int n = 0;
34
35     /* Unos niza */
36     printf("Unesi opadajuće sortiran niz celih brojeva: ");
37     while (scanf("%d", &niz[n]) == 1)
38         n++;
39
40     /* Stampanje rezultata */
41     printf("%d\n", prvi_manji_od_nule(niz, n));
42
43     return 0;
44 }
```

#### Rešenje 3.9

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



```
3  #include <stdlib.h>
5  unsigned int logaritam_a(unsigned int x)
6  {
7      /* Izlaz iz rekurzije */
8      if (x == 1)
9          return 0;
10     /* Rekurzivni korak */
11     return 1 + logaritam_a(x >> 1);
12 }
13 unsigned int logaritam_b(unsigned int x)
14 {
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
16     najvece vaznosti, tj. najlevlja. Pretragu se vrši od pozicije 0
17     do 31 */
18     int d = 0, l = sizeof(unsigned int) * 8 - 1;
19     int s;
20     /* Sve dok je desna granica pretrage desnije od leve */
21     while (d <= l) {
22         /* Racuna se sredisnja pozicija */
23         s = (l + d) / 2;
24         /* Proverava se da li je na toj poziciji trazena jedinica */
25         if ((1 << s) <= x && (1 << (s + 1)) > x)
26             return s;
27         /* Pretraga desne polovine binarnog zapisa */
28         if ((1 << s) > x)
29             l = s - 1;
30         /* Pretraga leve polovine binarnog zapisa */
31         else
32             d = s + 1;
33     }
34     return s;
35 }
36
37 int main()
38 {
39     unsigned int x;
40
41     /* Unos podatka */
42     printf("Unesi pozitivan ceo broj: ");
43     scanf("%u", &x);
44
45     /* Provera da li je uneti broj pozitivan */
46     if (x == 0) {
47         fprintf(stderr, "Logaritam od nule nije definisan\n");
48         exit(EXIT_FAILURE);
49     }
50
51     /* Ispis povratnih vrednosti funkcija */
52     printf("%u %u\n", logaritam_a(x), logaritam_b(x));
53 }
```

```
    return 0;
55 }
```

#### Rešenje 3.12

```
1  #include<stdio.h>
2  #define MAX 256
3
4  /* Iterativna verzija funkcije koja sortira niz celih brojeva,
5   primenom algoritma Selection Sort */
6  void selectionSort(int a[], int n)
7  {
8      int i, j;
9      int min;
10     int pom;
11
12     /* U svakoj iteraciji ove petlje se pronalazi najmanji element
13      medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
14      poziciju i, dok se element na poziciji i premesta na
15      poziciju min, na kojoj se nalazio najmanji od gore
16      navedenih elemenata. */
17     for (i = 0; i < n - 1; i++) {
18         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
19          nalazi najmanji od elemenata a[i],...,a[n-1]. */
20         min = i;
21         for (j = i + 1; j < n; j++)
22             if (a[j] < a[min])
23                 min = j;
24         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
25          samo ako su (i) i min razliciti, inace je nepotrebno. */
26         if (min != i) {
27             pom = a[i];
28             a[i] = a[min];
29             a[min] = pom;
30         }
31     }
32 }
33
34 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja
35 u sortiranom nizu celih brojeva */
36 int najmanje_rastojanje(int a[], int n)
37 {
38     int i, min;
39     min = a[1] - a[0];
40     for (i = 2; i < n; i++)
41         if (a[i] - a[i - 1] < min)
42             min = a[i] - a[i - 1];
43     return min;
44 }
45
```

```

47 int main()
48 {
49     int i, a[MAX];

51     /* Ucitavaju se elementi niza sve do kraja ulaza */
52     i = 0;
53     printf("Unesite elemente niza: ");
54     while (scanf("%d", &a[i]) != EOF)
55         i++;

57     /* Sortiranje */
58     selectionSort(a, i);

59     /* Ispis rezultata */
60     printf("%d\n", najmanje_rastojanje(a, i));

63     return 0;
64 }

```

### Rešenje 3.13

```

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

#define MAX_DIM 128

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

/* Funkcija vraca: 1 - ako jesu anagrami; 0 - inace. */
int anagrami(char s[], char t[], int n_s, int n_t)
{
    int i, n;

    /* Ako dve niske imaju razlicit broj elemenata onda nisu
       anagrami */
}

```

### 3 Algoritmi pretrage i sortiranja

---

```

    if (n_s != n_t)
32         return 0;

    /* Sortiramo niske */
    selectionSort(s, n_s);
36     selectionSort(t, n_t);

    n = n_s;

    /* Dve sortirane niske su anagrami akko su jednake */
    for (i = 0; i < n; i++)
40         if (s[i] != t[i])
42             return 0;
44     return 1;
}

46
int main()
48 {
    char s[MAX_DIM], t[MAX_DIM];
50     int n_s, n_t;

    /* Ucitavamo dve niske sa ulaza */
    printf("Unesite prvu nisku: ");
54     scanf("%s", s);
    printf("Unesite drugu nisku: ");
56     scanf("%s", t);

    /* Odredjujemo duzinu niski */
    n_s = strlen(s);
60     n_t = strlen(t);

    /* Proveravamo da li su niske anagrami */
    if (anagrami(s, t, n_s, n_t))
62         printf("jesu\n");
    else
64         printf("nisu\n");
66     return 0;
68 }
```

#### Rešenje 3.14

```

1  #include<stdio.h>
   #define MAX_DIM 256
3
   /* Funkcija za sortiranje niza */
5  void selectionSort(int s[], int n)
   {
7      int i, j, min;
       char pom;
9      for (i = 0; i < n; i++) {
           min = i;
```

```
11     for (j = i + 1; j < n; j++)
12         if (s[j] < s[min])
13             min = j;
14     if (min != i) {
15         pom = s[i];
16         s[i] = s[min];
17         s[min] = pom;
18     }
19 }
20 }
21
22 /* Funkcija za odredjivanje onog elementa sortiranog niza koji
23    se najvise puta pojavio u tom nizu */
24 int najvise_puta(int a[], int n)
25 {
26     int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
27     /* Za i-ti element izracunavamo koliko se puta pojavio u nizu */
28     for (i = 0; i < n; i = j) {
29         br_pojava = 1;
30         for (j = i + 1; j < n && a[i] == a[j]; j++)
31             br_pojava++;
32         /* Ispitujemo da li se do tog trenutka i-ti element pojavio
33            najvise puta u nizu */
34         if (br_pojava > max_br_pojava) {
35             max_br_pojava = br_pojava;
36             i_max_pojava = i;
37         }
38     }
39     /* Vracamo element koji se najvise puta pojavio u nizu */
40     return a[i_max_pojava];
41 }
42
43 int main()
44 {
45     int a[MAX_DIM], i;
46
47     /* Ucitavaju se elementi niza sve do kraja ulaza */
48     i = 0;
49     printf("Unesite elemente niza: ");
50     while (scanf("%d", &a[i]) != EOF)
51         i++;
52
53     /* Niz se sortira */
54     selectionSort(a, i);
55
56     /* Odredjuje se broj koji se najvise puta pojavio u nizu */
57     printf("%d\n", najvise_puta(a, i));
58
59     return 0;
60 }
```

#### Rešenje 3.15

```
1 #include<stdio.h>
2 #define MAX_DIM 256
3
4 /* Funkcija za sortiranje niza */
5 void selectionSort(int a[], int n)
6 {
7     int i, j, min, pom;
8     for (i = 0; i < n - 1; i++) {
9         min = i;
10        for (j = i + 1; j < n; j++)
11            if (a[j] < a[min])
12                min = j;
13        if (min != i) {
14            pom = a[i];
15            a[i] = a[min];
16            a[min] = pom;
17        }
18    }
19 }
20
21 /* Funkcija za binarnu pretragu niza. funkcija vraca: 1 - ako se
22    element x nalazi u nizu; 0 - inace. pretpostavlja se da je
23    niz sortiran u rastucem poretku */
24 int binarna_pretraga(int a[], int n, int x)
25 {
26     int levi = 0, desni = n - 1, srednji;
27
28     while (levi <= desni) {
29         srednji = (levi + desni) / 2;
30         if (a[srednji] == x)
31             return 1;
32         else if (a[srednji] > x)
33             desni = srednji - 1;
34         else if (a[srednji] < x)
35             levi = srednji + 1;
36     }
37     return 0;
38 }
39
40 int main()
41 {
42     int a[MAX_DIM], n = 0, zbir, i;
43
44     /* Ucitava se trazeni zbir */
45     printf("Unesite trazeni zbir: ");
46     scanf("%d", &zbir);
47
48     /* Ucitavaju se elementi niza sve do kraja ulaza */
49     i = 0;
50     printf("Unesite elemente niza: ");
```

```

51 while (scanf("%d", &a[i]) != EOF)
    i++;
53 n = i;

55 /* Sortira se niz */
    selectionSort(a, n);

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

```

### Rešenje 3.16

```

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

```

```

/* Datoteka sort.c */
2
#include "sort.h"
4
/* Funkcija sortira niz celih brojeva metodom sortiranja
6 izborom. Ideja algoritma je sledeca: U svakoj iteraciji
    pronalazimo najmanji element i postavljamo ga na pocetak
8 niza. Dakle, u prvoj iteraciji, pronalazimo najmanji element,
    i dovodimo ga na nultu mesto u nizu. U i-toj iteraciji

```

### 3 Algoritmi pretrage i sortiranja

---

```
10     najmanjih i elemenata su vec na svojim pozicijama, pa od i+1
11     do n-1 elementa trazimo najmanji, koji dovodimo na i+1
12     poziciju. */
13 void selectionsort(int a[], int n)
14 {
15     int i, j;
16     int min;
17     int pom;
18
19     /* U svakoj iteraciji ove petlje se pronalazi najmanji element
20     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
21     poziciju i, dok se element na poziciji i premesta na
22     poziciju min, na kojoj se nalazio najmanji od gore
23     navedenih elemenata. */
24     for (i = 0; i < n - 1; i++) {
25         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
26         nalazi najmanji od elemenata a[i],...,a[n-1]. */
27         min = i;
28         for (j = i + 1; j < n; j++)
29             if (a[j] < a[min])
30                 min = j;
31
32         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
33         samo ako su (i) i min razliciti, inace je nepotrebno. */
34         if (min != i) {
35             pom = a[i];
36             a[i] = a[min];
37             a[min] = pom;
38         }
39     }
40 }
41
42
43
44 /* Funkcija sortira niz celih brojeva metodom sortiranja
45 umetanjem. Ideja algoritma je sledeca: neka je na pocetku
46 i-te iteracije niz prvih i elemenata (a[0],a[1],...,a[i-1])
47 sortirano. U i-toj iteraciji zelimo da element a[i] umetnemo
48 na pravu poziciju medju prvih i elemenata tako da dobijemo
49 niz duzine i+1 koji je sortiran. Ovo radimo tako sto i-ti
50 element najpre uporedimo sa njegovim prvim levim susedom
51 (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i
52 niz a[0],a[1],...,a[i] je sortiran, pa mozemo preci na
53 sledecu iteraciju. Ako je a[i-1] vece, tada zamenjujemo a[i]
54 i a[i-1], a zatim proveravamo da li je potrebno dalje
55 potiskivanje elementa u levo, poredeci ga sa njegovim novim
56 levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
57 pravo mesto u nizu. */
58 void insertionsort(int a[], int n)
59 {
60     int i, j;
```



```

62  /* Na pocetku iteracije pretpostavljamo da je niz
    a[0],...,a[i-1] sortiran */
64  for (i = 1; i < n; i++) {

66      /* U ovoj petlji redom potiskujemo element a[i] u levo
        koliko je potrebno, dok ne zauzme pravo mesto, tako da
68      niz a[0],...,a[i] bude sortiran. Indeks j je trenutna
        pozicija na kojoj se element koji umecemo nalazi. Petlja
70      se završava ili kada element dodje do levog kraja (j==0)
        ili dok ne naidjemo na element a[j-1] koji je manji od
72      a[j]. */
        for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
74            int temp = a[j];
            a[j] = a[j - 1];
76            a[j - 1] = temp;
        }
78    }
80 }

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

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

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

108         /* Promenljiva ind registruje da je bilo premestanja.
            Samo u tom slucaju ima smisla ici na sledecu
110 iteraciju, jer ako nije bilo premestanja, znaci da su
            svi elementi vec u dobrom poretku, pa nema potrebe
112 prelaziti na kraci prefiks niza. Moglo je naravno i
            bez ovoga, algoritam bi radio ispravno, ali bi bio

```

### 3 Algoritmi pretrage i sortiranja

---

```
114         manje efikasan, jer bi cesto nepotrebno vrsio mnoga
115         uporedjivanja, kada je vec jasno da je sortiranje
116         zavrшено. */
117     ind = 1;
118 }
119 }
120
121 /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
122 dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
123 sastoji u tome da se kroz algoritam umetanja prolazi vise
124 puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
125 koji je manji od n (sto omogucuje razmenu udaljenih
126 elemenata) i tako se dobija h-sortiran niz, tj. niz u kome su
127 elementi na rastojanju h sortirani, mada susedni elementi to
128 ne moraju biti. U drugom prolazu kroz isti algoritam sprovodi
129 se isti postupak ali za manji korak h. Sa prolazima se
130 nastavlja sve do koraka h = 1, u kome se dobija potpuno
131 sortirani niz. Izbor pocetne vrednosti za h, i nacina
132 njegovog smanjivanja menja u nekim slucajevima brzinu
133 algoritma, ali bilo koja vrednost ce rezultovati ispravnim
134 sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
135 vrednost 1. */
136 void shellsort(int a[], int n)
137 {
138     int h = n / 2, i, j;
139     while (h > 0) {
140         /* Insertion sort sa korakom h */
141         for (i = h; i < n; i++) {
142             int temp = a[i];
143             j = i;
144             while (j >= h && a[j - h] > temp) {
145                 a[j] = a[j - h];
146                 j -= h;
147             }
148             a[j] = temp;
149         }
150         h = h / 2;
151     }
152 }
153
154 #define MAX 1000000
155
156 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem.
157 Sortiranje se vrsi od elementa na poziciji l do onog na
158 poziciji d. Na pocetku, da bismo dobili niz kompletno
159 sortirani, l mora biti 0, a d je jednako poslednjem validnom
160 indeksu u nizu. Funkcija niz podeli na dve polovine, levu i
161 desnu, koje zatim rekurzivno sortira. Od ova dva sortirana
162 podniza, dobijamo sortirani niz ucesljavanjem, tj.
163 istovremenim prolaskom kroz oba niza i izborom trenutnog
164 manjeg elementa koji se smesta u pomocni niz. Na kraju
165 algoritma, sortirani elementi su u pomocnom nizu, koji se
```

```
166     kopira u originalni niz. */
void mergesort(int a[], int l, int d)
168 {
    int s;
170     static int b[MAX];          /* Pomocni niz */
    int i, j, k;

172     /* Izlaz iz rekurzije */
174     if (l >= d)
        return;

176     /* Odredjujemo sredisnji indeks */
178     s = (l + d) / 2;

180     /* Rekurzivni pozivi */
    mergesort(a, l, s);
182     mergesort(a, s + 1, d);

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

190     /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
192     while (i <= s && j <= d) {
        if (a[i] < a[j])
194             b[k++] = a[i++];
        else
196             b[k++] = a[j++];
    }

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

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

210     /* Prepisujemo "ucesljani" niz u originalni niz */
212     for (k = 0, i = l; i <= d; i++, k++)
        a[i] = b[k];
214 }

216 /* Funkcija menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
```

### 3 Algoritmi pretrage i sortiranja

---

```
218 {
219     int tmp = a[i];
220     a[i] = a[j];
221     a[j] = tmp;
222 }
223
224 /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r.
225    Njena ideja sortiranja je izbor jednog elementa niza, koga
226    nazivamo pivot, koga cemo dovesti na svoje mesto. Posle ovog
227    koraka, svi elementi levo od njega bice manji, a svi desno
228    bice veci od njega. Kako smo pivota doveli na svoje mesto, da
229    bismo imali kompletno sortirani niz, treba sortirati elemente
230    levo (manje) od njega, i elemente desno (vece). Kako je
231    dimenzija ova dva podniza manja od dimenzije pocetnog niza
232    koji je trebalo sortirati, ovaj deo ce za nas uraditi
233    rekurzija. */
234 void quicksort(int a[], int l, int r)
235 {
236     int i, pivot_position;
237
238     /* Izlaz iz rekurzije -- prazan niz */
239     if (l >= r)
240         return;
241
242
243     /* Particionisanje niza. Svi elementi na pozicijama <=
244        pivot_position (izuzev same pozicije l) su strogo manji od
245        pivota. Kada se pronadje neki element manji od pivota,
246        uvecava se pivot_position i na tu poziciju se premesta
247        nadjeni element. Na kraju ce pivot_position zaista biti
248        pozicija na koju treba smestiti pivot, jer ce svi elementi
249        levo od te pozicije biti manji a desno biti veci ili
250        jednaki od pivota. */
251     pivot_position = l;
252     for (i = l + 1; i <= r; i++)
253         if (a[i] < a[l])
254             swap(a, ++pivot_position, i);
255
256     /* Postavljamo pivot na svoje mesto */
257     swap(a, l, pivot_position);
258
259     /* Rekurzivno sortiramo elemente manje od pivota */
260     quicksort(a, l, pivot_position - 1);
261     /* Rekurzivno sortiramo elemente vece pivota */
262     quicksort(a, pivot_position + 1, r);
263 }
264
265 #include <stdio.h>
266 #include <stdlib.h>
267 #include <time.h>
268 #include "sort.h"
```

```

6  /* Maksimalna duzina niza */
   #define MAX 1000000
8
10 int main(int argc, char *argv[])
11 {
12     /******
13      tip_sortiranja == 0 => selectionsort
14                               (podrazumevano)
15      tip_sortiranja == 1 => insertionsort
16                               -i opcija komandne linije
17      tip_sortiranja == 2 => bubblesort
18                               -b opcija komandne linije
19      tip_sortiranja == 3 => shellsort
20                               -s opcija komandne linije
21      tip_sortiranja == 4 => mergesort
22                               -m opcija komandne linije
23      tip_sortiranja == 5 => quicksort
24                               -q opcija komandne linije
25     *****/
26     int tip_sortiranja = 0;
27     /******
28      tip_niza == 0 => slucajno generisani nizovi
29                               (podrazumevano)
30      tip_niza == 1 => rastuce sortirani nizovi
31                               -r opcija komandne linije
32      tip_niza == 2 => opadajuće sortirani nizovi
33                               -o opcija komandne linije
34     *****/
35     int tip_niza = 0;
36
37     /* Dimenzija niza koji se sortira */
38     int dimenzija;
39     int i;
40     int niz[MAX];
41
42     /* Provera argumenata komandne linije */
43     if (argc < 2) {
44         fprintf(stderr,
45             "Program zahteva bar 2 argumenta komandne linije!\n");
46         exit(EXIT_FAILURE);
47     }
48
49     /* Ocitavamo opcije i argumente prilikom poziva programa */
50     for (i = 1; i < argc; i++) {
51         /* Ako je u pitanju opcija... */
52         if (argv[i][0] == '-') {
53             switch (argv[i][1]) {
54                 case 'i':
55                     tip_sortiranja = 1;
56                     break;
57                 case 'b':

```

```
        tip_sortiranja = 2;
58     break;
    case 's':
60     tip_sortiranja = 3;
        break;
62     case 'm':
        tip_sortiranja = 4;
64     break;
    case 'q':
66     tip_sortiranja = 5;
        break;
68     case 'r':
        tip_niza = 1;
70     break;
    case 'o':
72     tip_niza = 2;
        break;
74     default:
        printf("Pogresna opcija -%c\n", argv[i][1]);
76     return 1;
        break;
78     }
    }
80     /* Ako je u pitanju argument, onda je to duzina niza koji
        treba da se sortira */
82     else {
        dimenzija = atoi(argv[i]);
84     if (dimenzija <= 0 || dimenzija > MAX) {
        fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
86     exit(EXIT_FAILURE);
        }
88     }
    }
}

90
92     /* Elemente niza odredjujemo slucajno, ali vodeci racuna o
        tipu niza dobijenom iz komandni linije. srandom funkcija
        obezbedjuje novi seed za pozivanje random funkcije, i kako
94     nas niz ne bi uvek isto izgledao seed smo postavili na
        tekuce vreme u sekundama od Nove godine 1970. random()%100
96     daje brojeve izmedju 0 i 99 */
    srandom(time(NULL));
98     if (tip_niza == 0)
        for (i = 0; i < dimenzija; i++)
100         niz[i] = random();
    else if (tip_niza == 1)
        for (i = 0; i < dimenzija; i++)
102         niz[i] =
104         i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
    else
106         for (i = 0; i < dimenzija; i++)
            niz[i] =
108         i == 0 ? random() % 100 : niz[i - 1] - random() % 100;
```

```

110  /* Ispisujemo elemente niza */
111  /*****
112   Ovaj deo je iskomentaran jer ne zelimo da se sledeci ispis
113   nadje na izlazu. Njegova svrha je samo bila provera da li je
114   niz generisan u skladu sa opcijama komandne linije.

116   printf("Niz koji sortiramo je:\n");
117   for (i = 0; i < dimenzija; i++)
118     printf("%d\n", niz[i]);
119   *****/

120

122  /* Sortiramo niz na odgovarajuci nacin */
123  if (tip_sortiranja == 0)
124    selectionsort(niz, dimenzija);
125  else if (tip_sortiranja == 1)
126    insertion sort(niz, dimenzija);
127  else if (tip_sortiranja == 2)
128    bubblesort(niz, dimenzija);
129  else if (tip_sortiranja == 3)
130    shellsort(niz, dimenzija);
131  else if (tip_sortiranja == 4)
132    mergesort(niz, 0, dimenzija - 1);
133  else
134    quicksort(niz, 0, dimenzija - 1);

136  /* Ispisujemo elemente niza */
137  /*****
138   Ovaj deo je iskomentaran jer nismo zeleli da vreme potrebno
139   za njegovo izvrsavanje bude ukljuceno u vreme izmereno
140   programom time. Takodje, kako je svrha ovog programa da prikaze
141   vremena razlicitih algoritama sortiranja, dimenzije nizova ce
142   biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
143   od toliko elemenata. Ovaj deo je koriscen u razvoju programa
144   zarad testiranja korektnosti.

146   printf("Sortiran niz je:\n");
147   for (i = 0; i < dimenzija; i++)
148     printf("%d\n", niz[i]);
149   *****/

150  return 0;
151 }

```

### Rešenje 3.17

```

1  #include <stdio.h>
2  #define MAX_DIM 256
3
4  int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,

```

```
5         int dim3)
6     {
7         int i = 0, j = 0, k = 0;
8         /* U slucaju da je dimenzija treceg niza manja od neophodne,
9            funkcija vraca -1 */
10        if (dim3 < dim1 + dim2)
11            return -1;
12
13        /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja
14           jednog od njih */
15        while (i < dim1 && j < dim2) {
16            if (niz1[i] < niz2[j])
17                niz3[k++] = niz1[i++];
18            else
19                niz3[k++] = niz2[j++];
20        }
21        /* Ostatak prvog niza prepisujemo u treci */
22        while (i < dim1)
23            niz3[k++] = niz1[i++];
24
25        /* Ostatak drugog niza prepisujemo u treci */
26        while (j < dim2)
27            niz3[k++] = niz2[j++];
28        return dim1 + dim2;
29    }
30
31    int main()
32    {
33        int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
34        int i = 0, j = 0, k, dim3;
35
36        /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
37           Pretpostavka je da na ulazu nece biti vise od MAX_DIM
38           elemenata */
39        printf("Unesite elemente prvog niza: ");
40        while (1) {
41            scanf("%d", &niz1[i]);
42            if (niz1[i] == 0)
43                break;
44            i++;
45        }
46        printf("Unesite elemente drugog niza: ");
47        while (1) {
48            scanf("%d", &niz2[j]);
49            if (niz2[j] == 0)
50                break;
51            j++;
52        }
53
54        /* Poziv trazene funkcije */
55        dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);
```



```

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

61  return 0;
63 }

```

### Rešenje 3.18

```

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

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

12   /* Ako nema dovoljno argumenata komandne linije */
14   if (argc < 3) {
        fprintf(stderr,
16           "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
        exit(EXIT_FAILURE);
18   }

20   /* Otvaramo datoteku zadatu prvim argumentom komandne linije */
    fin1 = fopen(argv[1], "r");
22   if (fin1 == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke %s\n",
24           argv[1]);
        exit(EXIT_FAILURE);
26   }

28   /* Otvaramo datoteku zadatu drugim argumentom komandne linije */
    fin2 = fopen(argv[2], "r");
30   if (fin2 == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke %s\n",
32           argv[2]);
        exit(EXIT_FAILURE);
34   }

36   /* Otvaranje datoteke za upis rezultata */
    fout = fopen("ceo-tok.txt", "w");
38   if (fout == NULL) {
        fprintf(stderr,
40           "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
        exit(EXIT_FAILURE);

```

```
42 }

44 /* Citamo narednog studenta iz prve datoteke */
45 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
46     kraj1 = 1;

48 /* Citamo narednog studenta iz druge datoteke */
49 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
50     kraj2 = 1;

52 /* Sve dok nismo dosli do kraja neke datoteke */
53 while (!kraj1 && !kraj2) {
54     if (strcmp(ime1, ime2) < 0) {
55         /* Ime i prezime iz prve datoteke je leksikografski
56            ranije, upisujemo ga u izlaznu datoteku */
57         fprintf(fout, "%s %s\n", ime1, prezime1);
58         /* Citamo narednog studenta iz prve datoteke */
59         if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
60             kraj1 = 1;
61     } else {
62         /* Ime i prezime iz druge datoteke je leksikografski
63            ranije, upisujemo ga u izlaznu datoteku */
64         fprintf(fout, "%s %s\n", ime2, prezime2);
65         /* Citamo narednog studenta iz druge datoteke */
66         if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
67             kraj2 = 1;
68     }
69 }

70 /* Ako smo iz prethodne petlje izašli zato što se doslo do
71    kraja druge datoteke, onda ima još imena u prvoj datoteci,
72    i prepisujemo ih, redom, jer su već sortirani po imenu. */
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 /* Ako smo iz prve petlje izašli zato što se doslo do kraja
79    prve datoteke, onda ima još imena u drugoj datoteci, i
80    prepisujemo ih, redom, jer su već sortirani po imenu. */
81 while (!kraj2) {
82     fprintf(fout, "%s %s\n", ime2, prezime2);
83     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
84         kraj2 = 1;
85 }

86 /* Zatvaramo datoteke */
87 fclose(fin1);
88 fclose(fin2);
89 fclose(fout);
```

```
94     return 0;
    }
```

### Rešenje 3.19

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  #define MAX_BR_TACAKA 128
7
8  /* Struktura koja reprezentuje koordinate tacke */
9  typedef struct Tacka {
10     int x;
11     int y;
12 } Tacka;
13
14 /* Funkcija racuna rastojanje zadate tacke od koordinatnog
15    pocetka (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
22    koordinatnog 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;
```

### 3 Algoritmi pretrage i sortiranja

---

```
47     Tacka tmp;

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

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++) {
        min = i;
71         for (j = i + 1; j < n; j++) {
            if (abs(t[j].y) < abs(t[min].y)) {
73                 min = j;
            }
75         }
        if (min != i) {
77             tmp = t[i];
            t[i] = t[min];
79             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     /* Proveravamo broj argumenata komandne linije: ocekujemo ime
93        izvrsnog programa, opciju, ime ulazne datoteke i ime
        izlazne datoteke tj. ocekujemo 4 argumenta */
95     if (argc != 4) {
        fprintf(stderr,
97             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
        return 0;
    }
```

```
99  }

101  /* Otvaramo datoteku 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  /* Otvaramo datoteku 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 ne stignemo do kraja ulazne datoteke učitavamo
    koordinate tacaka i smestamo ih na odgovarajucu poziciju
119  odredjenu brojacem i; prilikom učitavanja oslanjamo se na
    svojstvo funkcije fscanf povratka EOF vrednosti kada stigne
121  do kraja ulaza */
    i = 0;
123  while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
        i++;
125  }

127  /* Cuvamo broj procitanih tacaka */
    n = i;

129

131  /* Analiziramo zadatu opciju: kako ocekujemo da je argv[1]
    "-x" ili "-y" ili "-o" sigurni smo da je argv[1][0] crtica
    (karakter -) i dalje proveravamo sta je na sledecoj
133  poziciji tj. sta je argv[1][1] */

135  switch (argv[1][1]) {
    case 'x':
137      /* Ako je u pitanju karakter x, pozivamo funkciju za
        sortiranje po vrednosti x koordinate */
139      sortiraj_po_x(tacke, n);
        break;
141  case 'y':
        /* Ako je u pitanju karakter y, pozivamo funkciju za
143      sortiranje po vrednosti y koordinate */
        sortiraj_po_y(tacke, n);
145      break;
147  case 'o':
        /* Ako je u pitanju karakter o, pozivamo funkciju za
        sortiranje po udaljenosti od koorinatnog pocetka */
149      sortiraj_po_rastojanju(tacke, n);
        break;
```

### 3 Algoritmi pretrage i sortiranja

---

```
151 }
153 /* Upisujemo dobijeni niz u izlaznu datoteku */
154 for (i = 0; i < n; i++) {
155     fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
156 }
157
158 /* Zatvaramo otvorene datoteke */
159 fclose(ulaz);
160 fclose(izlaz);
161
162 return 0;
163 }
```

#### Rešenje 3.20

```
#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
48             a[i].prezime,...,a[n-1].prezime. */
            min = i;
50         for (j = i + 1; j < n; j++)
            if (strcmp(a[j].prezime, a[min].prezime) < 0)
52             min = j;
        /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
54             samo ako su (i) i min razliciti, inace je nepotrebno. */
        if (min != i) {
56             pom = a[i];
            a[i] = a[min];
58             a[min] = pom;
        }
60     }
}

62 /* Pretraga niza Gradjana */
64 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
{
66     int i;
    for (i = 0; i < n; i++)
68         if (strcmp(a[i].ime, x->ime) == 0
            && strcmp(a[i].prezime, x->prezime) == 0)
70             return i;
    return -1;
72 }

74 int main()
76 {
    Gradjanin spisak1[MAX], spisak2[MAX];
78     int isti_rbr = 0;
    int i, n;
80     FILE *fp = NULL;

82     /* Otvaranje datoteke */
    if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
84         fprintf(stderr,
            "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
86         exit(EXIT_FAILURE);
    }
}

```

### 3 Algoritmi pretrage i sortiranja

```
88      /* Citanje sadrzaja */
89      for (i = 0;
90           fscanf(fp, "%s %s", spisak1[i].ime,
91                 spisak1[i].prezime) != EOF; i++)
92          spisak2[i] = spisak1[i];
93      n = i;
94
95      /* Zatvaranje datoteke */
96      fclose(fp);
97
98      sort_ime(spisak1, n);
99
100     /******
101     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
102     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
103
104     printf("Biracki spisak [uredjen prema imenima]:\n");
105     for(i=0; i<n; i++)
106         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107     *****/
108
109     sort_prezime(spisak2, n);
110
111     /******
112     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
113     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
114
115     printf("Biracki spisak [uredjen prema prezimenima]:\n");
116     for(i=0; i<n; i++)
117         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
118     *****/
119
120     /* Linearno pretrazivanje nizova */
121     for (i = 0; i < n; i++)
122         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
123             isti_rbr++;
124
125     /* Alternativno (efikasnije) resenje */
126     /******
127     for(i=0; i<n ;i++)
128         if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
129             strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
130             isti_rbr++;
131     *****/
132
133     /* Ispis rezultata */
134     printf("%d\n", isti_rbr);
135
136     exit(EXIT_SUCCESS);
137 }
```



## Rešenje 3.22

```

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

#define MAX_BR_RECII 128
#define MAX_DUZINA_RECII 32

/* Funkcija koja izracunava broj suglasnika u reci */
int broj_suglasnika(char s[])
{
    char c;
    int i;
    int suglasnici = 0;
    /* Obilazimo karakter po karakter zadate niske */
    for (i = 0; s[i]; i++) {
        /* Ako je u pitanju slovo */
        if (isalpha(s[i])) {
            /* Pretvaramo ga u veliko da bismo mogli da pokrijemo
            slucaj i malih i velikih suglasnika */
            c = toupper(s[i]);
            /* Ukoliko slovo nije samoglasnik */
            if (c != 'A' && c != 'E' && c != 'I' && c != 'O'
            && c != 'U') {
                /* Uvecavamo broj suglasnika */
                suglasnici++;
            }
        }
    }
    /* Vracamo izracunatu vrednost */
    return suglasnici;
}

/* Funkcija koja sortira reci po zadatom kriterijumu.
Informacija o duzini reci se mora proslediti zbog pravilnog
upravljanja memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)
{
    int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
        duzina_j, duzina_min;
    char tmp[MAX_DUZINA_RECII];
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i; j < n; j++) {
            /* Prvo uporedjujemo broj suglasnika */
            broj_suglasnika_j = broj_suglasnika(reci[j]);
            broj_suglasnika_min = broj_suglasnika(reci[min]);
            if (broj_suglasnika_j < broj_suglasnika_min)
                min = j;
            else if (broj_suglasnika_j == broj_suglasnika_min) {

```

### 3 Algoritmi pretrage i sortiranja

---

```
52      /* Zatim, reci imaju isti broj suglasnika uporedjujemo
      duzine */
54      duzina_j = strlen(reci[j]);
      duzina_min = strlen(reci[min]);

56      if (duzina_j < duzina_min)
          min = j;
58      else
          /* A ako reci imaju i isti broj suglasnika i iste
          duzine, uporedjujemo ih leksikografski */
60      if (duzina_j == duzina_min
62          && strcmp(reci[j], reci[min]) < 0)
          min = j;
64      }
    }
66    if (min != i) {
        strcpy(tmp, reci[min]);
68        strcpy(reci[min], reci[i]);
        strcpy(reci[i], tmp);
70    }
    }
72 }

74 int main()
75 {
76     FILE *ulaz;
78     int i = 0, n;

80     /* Niz u kojem ce biti smestane reci. Prvi broj oznacava broj
      reci, a drugi maksimalnu duzinu pojedinačne reci */
82     char reci[MAX_BR_RECII][MAX_DUZINA_RECII];

84     /* Otvaramo datoteku niske.txt za citanje */
      ulaz = fopen("niske.txt", "r");
86     if (ulaz == NULL) {
        fprintf(stderr,
88             "Greska prilikom otvaranja datoteke niske.txt!\n");
        return 0;
90     }

92     /* Sve dok mozemo da procitamo sledecu rec */
      while (fscanf(ulaz, "%s", reci[i]) != EOF) {
94         /* Proveravamo da li smo ucitali najvise dozvoljenih reci i
          ako jesmo, prekidamo ucitavanje */
96         if (i == MAX_BR_RECII)
            break;
98         /* Pripremamo brojac za narednu iteraciju */
            i++;
100     }

102     /* n je duzina naseg niza reci i predstavlja poslednju
```

```

    vrednost koriscenog brojaca */
104 n = i;
    /* Pozivamo funkciju za sortiranje reci - OPREZ: nacin
106    prosledjivanja niza reci */
    sortiraj_reci(reci, n);
108
    /* Ispisujemo sortirani niz reci */
110 for (i = 0; i < n; i++) {
    printf("%s ", reci[i]);
112 }
    printf("\n");
114
    /* Zatvaramo datoteku */
116 fclose(ulaz);
118
    return 0;
}

```

### Rešenje 3.23

```

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

5  #define MAX_ARTIKALA 100000

7  /* Struktura koja predstavlja jedan artikal */
   typedef struct art {
9      long kod;
      char naziv[20];
11     char proizvođjac[20];
      float cena;
13 } Artikal;

15 /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj
   sa traženim bar kodom */
17 int binarna_pretraga(Artikal a[], int n, long x)
{
19     int levi = 0;
     int desni = n - 1;
21
     /* Dokle god je indeks levi levo od indeksa desni */
23     while (levi <= desni) {
        /* Racunamo sredisnji indeks */
25         int srednji = (levi + desni) / 2;
        /* Ako je sredisnji element veci od x, tada se x mora
27         nalaziti u levoj polovini niza */
        if (x < a[srednji].kod)
29             desni = srednji - 1;
        /* Ako je sredisnji element manji od x, tada se x mora
31         nalaziti u desnoj polovini niza */
    }
}

```

```

    else if (x > a[srednji].kod)
33         levi = srednji + 1;
    else
35         /* Ako je sredisnji element jednak x, tada smo pronasli x
           na poziciji srednji */
37         return srednji;
    }
39     /* Ako nije pronadjen vratamo -1 */
    return -1;
41 }

43 /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
void selection_sort(Artikal a[], int n)
45 {
    int i, j;
    int min;
    Artikal pom;
49
    for (i = 0; i < n - 1; i++) {
51         min = i;
        for (j = i + 1; j < n; j++)
53             if (a[j].kod < a[min].kod)
                min = j;
55         if (min != i) {
            pom = a[i];
57             a[i] = a[min];
            a[min] = pom;
59         }
    }
61 }

63 int main()
{
65     Artikal asortiman[MAX_ARTIKALA];
    long kod;
    int i, n;
    float racun;
69

    FILE *fp = NULL;
71
    /* Otvaranje datoteke */
73     if ((fp = fopen("artikli.txt", "r")) == NULL) {
        fprintf(stderr,
75             "Neuspesno otvaranje datoteke artikli.txt.\n");
        exit(EXIT_FAILURE);
77     }

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

```

i++;
85
/* Zatvaranje datoteke */
87 fclose(fp);

89 n = i;

91 /* Sortiracemo celokupan asortiman prodavnice prema kodovima
jer ce pri kucanju racuna prodavac unositi kod artikla.
93 Prilikom kucanja svakog racuna pretrazuje se asortiman, da
bi se utvrdila cena artikla. Kucanje racuna obuhvata vise
95 pretraga asortimana i u interesu nam je da ta operacija
bude sto efikasnija. Zelimo da koristimo algoritam binarne
97 pretrage priliko pretrazivanje po kodu artikla. Iz tog
razloga, potrebno je da nam asortiman bude sortiran po
99 kodovima i to cemo uraditi primenom selection sort
algoritma. Sortiramo samo jednom na pocetku, ali zato posle
101 brzo mozemo da pretrazujemo prilikom kucanja proizvoljno
puno racuna. Vreme koje se utrosi na sortiranje na pocetku
103 izvorsavanja programa, kasnije se isplati jer za brojna
trazenja artikla mozemo umesto linearne da koristimo
105 efikasniju binarnu pretragu. */
selection_sort(asortiman, n);

107
/* Ispis stanja u prodavnici */
109 printf
("Asortiman:\nKOD          Naziv artikla      Ime
proizvodjaca      Cena\n");
111 for (i = 0; i < n; i++)
printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113      asortiman[i].naziv, asortiman[i].proizvodjac,
asortiman[i].cena);
115

kod = 0;
117 while (1) {
printf("-----\n");
119 printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
printf("- Za nov racun unesite kod artikla!\n\n");
121 /* Unos bar koda provog artikla sledeceg kupca */
if (scanf("%ld", &kod) == EOF)
123     break;
/* Trenutno racun novog kupca */
125 racun = 0;
/* Za sve artikle trenutnog kupca */
127 while (1) {
/* Nalazimo ih u nizu */
129 if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
printf
131     ("\tGRESKA: Ne postoji proizvod sa trazanim kodom!\n");
} else {
133     printf("\tTrazili ste:\t%s %s %12.2f\n",
asortiman[i].naziv, asortiman[i].proizvodjac,

```

### 3 Algoritmi pretrage i sortiranja

---

```
135         asortiman[i].cena);
136         /* I dodajemo na ukupan racun */
137         racun += asortiman[i].cena;
138     }
139     /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0
140        ako on nema vise artikla */
141     printf("Unesite kod artikla [ili 0 za prekid]: \t");
142     scanf("%ld", &kod);
143     if (kod == 0)
144         break;
145 }
146 /* Stampanje ukupnog racuna trenutnog kupca */
147 printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
148 }
149
150 printf("Kraj rada kase!\n");
151
152 exit(EXIT_SUCCESS);
153 }
```

#### Rešenje 3.24

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX 500
6
/* Struktura koja nam je neophodna za sve informacije o
8   pojedinačnom studentu */
typedef struct {
10     char ime[20];
11     char prezime[25];
12     int prisustvo;
13     int zadaci;
14 } Student;

15
16 /* Funkcija kojom sortiramo niz struktura po prezimenu
17    leksikografski rastuće */
18 void sort_ime_leksikografski(Student niz[], int n)
19 {
20     int i, j;
21     int min;
22     Student pom;

23
24     for (i = 0; i < n - 1; i++) {
25         min = i;
26         for (j = i + 1; j < n; j++)
27             if (strcmp(niz[j].ime, niz[min].ime) < 0)
28                 min = j;
```

```

30     if (min != i) {
31         pom = niz[min];
32         niz[min] = niz[i];
33         niz[i] = pom;
34     }
35 }
36 }

38 /* Funkcija kojom sortiramo niz struktura po ukupnom broju
39    uradjenih zadataka opadajuće, ukoliko neki studenti imaju
40    isti broj uradjenih zadataka sortiraju se po dužini imena
41    rastuće. */
42 void sort_zadatke_pa_imena(Student niz[], int n)
43 {
44     int i, j;
45     int max;
46     Student pom;
47     for (i = 0; i < n - 1; i++) {
48         max = i;
49         for (j = i + 1; j < n; j++)
50             if (niz[j].zadaci > niz[max].zadaci)
51                 max = j;
52         else if (niz[j].zadaci == niz[max].zadaci
53                 && strlen(niz[j].ime) < strlen(niz[max].ime))
54             max = j;
55         if (max != i) {
56             pom = niz[max];
57             niz[max] = niz[i];
58             niz[i] = pom;
59         }
60     }
61 }

62 /* Funkcija kojom sortiramo niz struktura po broju casova na
63    kojima su bili opadajuće, a ukoliko * neki studenti imaju
64    isti broj casova, sortiraju se opadajuće po broju uradjenih
65    zadataka, * a ukoliko se i po broju zadataka poklapaju
66    sortirati ih po prezimenu opadajuće. */
67 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
68 {
69     int i, j;
70     int max;
71     Student pom;
72     for (i = 0; i < n - 1; i++) {
73         max = i;
74         for (j = i + 1; j < n; j++)
75             if (niz[j].prisustvo > niz[max].prisustvo)
76                 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

```

### 3 Algoritmi pretrage i sortiranja

---

```
82         && niz[j].zadaci == niz[max].zadaci
83         && strcmp(niz[j].prezime, niz[max].prezime) > 0)
84     max = j;
85     if (max != i) {
86         pom = niz[max];
87         niz[max] = niz[i];
88         niz[i] = pom;
89     }
90 }
91 }
92
93
94
95 int main(int argc, char *argv[])
96 {
97     Student praktikum[MAX];
98     int i, br_studenata = 0;
99
100     FILE *fp = NULL;
101
102     /* Otvaranje datoteke za citanje */
103     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
104         fprintf(stderr,
105             "Neuspesno otvaranje datoteke aktivnost.txt.\n");
106         exit(EXIT_FAILURE);
107     }
108
109     /* Ucitavanje sadrzaja */
110     for (i = 0;
111          fscanf(fp, "%s%s%d", praktikum[i].ime,
112                praktikum[i].prezime, &praktikum[i].prisustvo,
113                &praktikum[i].zadaci) != EOF; i++);
114
115     /* Zatvaranje datoteke */
116     fclose(fp);
117     br_studenata = i;
118
119     /* Kreiramo prvi spisak studenata na kom su sortirani
120        leksikografski po imenu rastuce */
121     sort_ime_leksikografski(praktikum, br_studenata);
122     /* Otvaranje datoteke za pisanje */
123     if ((fp = fopen("dat1.txt", "w")) == NULL) {
124         fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
125         exit(EXIT_FAILURE);
126     }
127     /* Upis niza u datoteku */
128     fprintf
129     (fp,
130      "Studenti sortirani po imenu leksikografski rastuce:\n");
131     for (i = 0; i < br_studenata; i++)
132         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
133                praktikum[i].prezime, praktikum[i].prisustvo,
134                praktikum[i].zadaci);
```



```

134  /* Zatvaranje datoteke */
      fclose(fp);
136
      /* Na drugom su sortirani po ukupnom broju uradjenih zadataka
138  opadajuće, ukoliko neki studenti imaju isti broj uradjenih
      zadataka sortiraju se po dužini imena rastuće. */
140  sort_zadatke_pa_imena(praktikum, br_studenata);
      /* Otvaranje datoteke za pisanje */
142  if ((fp = fopen("dat2.txt", "w")) == NULL) {
      fprintf(stderr, "Neuspješno otvaranje datoteke dat2.txt.\n");
144  exit(EXIT_FAILURE);
  }
146  /* Upis niza u datoteku */
      fprintf(fp,
148  "Studenti sortirani po broju zadataka opadajuće,\n");
      fprintf(fp, "pa po dužini imena rastuće:\n");
150  for (i = 0; i < br_studenata; i++)
      fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
152  praktikum[i].prezime, praktikum[i].prisustvo,
      praktikum[i].zadaci);
154  /* Zatvaranje datoteke */
      fclose(fp);
156
      /* Na trećem spisku su sortirani po broju časova na kojima su
158  bili opadajuće, a ukoliko neki studenti imaju isti broj
      časova, sortiraju se opadajuće po broju uradjenih zadataka,
160  a ukoliko se i po broju zadataka poklapaju sortirati ih po
      prezimenu opadajuće. */
162  sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum,
      br_studenata);
164  /* Otvaranje datoteke za pisanje */
      if ((fp = fopen("dat3.txt", "w")) == NULL) {
166  fprintf(stderr, "Neuspješno otvaranje datoteke dat3.txt.\n");
      exit(EXIT_FAILURE);
168  }
      /* Upis niza u datoteku */
170  fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
      fprintf(fp, "pa po broju zadataka,\n");
172  fprintf(fp, "pa po prezimenima leksikografski opadajuće:\n");
      for (i = 0; i < br_studenata; i++)
174  fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
      praktikum[i].prezime, praktikum[i].prisustvo,
176  praktikum[i].zadaci);
      /* Zatvaranje datoteke */
178  fclose(fp);
180
      return 0;
  }

```

## Rešenje 3.25

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #define KORAK 10
5
6  /* Struktura koja opisuje jednu pesmu */
7  typedef struct {
8      char *izvodjac;
9      char *naslov;
10     int broj_gledanja;
11 } Pesma;
12
13 /* Funkcija za uporedjivanje pesama po broju gledanosti
14    (potrebna za rad qsort funkcije) */
15 int uporedi_gledanost(const void *pp1, const void *pp2)
16 {
17     Pesma *p1 = (Pesma *) pp1;
18     Pesma *p2 = (Pesma *) pp2;
19
20     return p2->broj_gledanja - p1->broj_gledanja;
21 }
22
23 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad
24    qsort funkcije) */
25 int uporedi_naslove(const void *pp1, const void *pp2)
26 {
27     Pesma *p1 = (Pesma *) pp1;
28     Pesma *p2 = (Pesma *) pp2;
29
30     return strcmp(p1->naslov, p2->naslov);
31 }
32
33 /* Funkcija za uporedjivanje pesama po izvodjaku (potrebna za
34    rad qsort funkcije) */
35 int uporedi_izvodjace(const void *pp1, const void *pp2)
36 {
37     Pesma *p1 = (Pesma *) pp1;
38     Pesma *p2 = (Pesma *) pp2;
39
40     return strcmp(p1->izvodjac, p2->izvodjac);
41 }
42
43 int main(int argc, char *argv[])
44 {
45     FILE *ulaz;
46     Pesma *pesme;          /* Pokazivac na deo memorije za
47                             cuvanje pesama */
48     int alocirano_za_pesme; /* Broj mesta alociranih za
49                             pesme */
50     int i;                 /* Redni broj pesme cije se
```

```
52             informacije citaju */
53     int n;                               /* Ukupan broj pesama */
54     int j, k;
55     char c;
56     int alocirano;                       /* Broj mesta alociranih za
57                                           propratne informacije o
58                                           pesmama */
59
60     int broj_gledanja;
61
62     /* Pripremamo datoteku za citanje */
63     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
64     if (ulaz == NULL) {
65         printf("Greska pri otvaranju ulazne datoteke!\n");
66         return 0;
67     }
68
69     /* Citamo informacije o pesmama */
70     pesme = NULL;
71     alocirano_za_pesme = 0;
72     i = 0;
73
74     while (1) {
75
76         /* Proveravamo da li smo stigli do kraja datoteke */
77         c = fgetc(ulaz);
78         if (c == EOF) {
79             /* Ako smo dobili kao rezultat EOF, jesmo, nema vise
80                sadrzaja za citanje */
81             break;
82         } else {
83             /* Ako nismo, vracamo procitani karakter nazad */
84             ungetc(c, ulaz);
85         }
86
87         /* Proveravamo da li imamo dovoljno memorije za citanje nove
88            pesme */
89         if (alocirano_za_pesme == i) {
90
91             /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
92                alociramo novih KORAK mesta */
93             alocirano_za_pesme += KORAK;
94             pesme =
95                 (Pesma *) realloc(pesme,
96                                   alocirano_za_pesme * sizeof(Pesma));
97
98             /* Proveravamo da li je nova memorija uspesno realocirana */
99             if (pesme == NULL) {
100                 /* Ako nije ... */
101                 /* Ispisujemo obavestenje */
102                 printf("Problem sa alokacijom memorije!\n");
```

### 3 Algoritmi pretrage i sortiranja

---

```
104      /* I oslobadjamo svu memoriju zauzetu do ovog koraka */
105      for (k = 0; k < i; k++) {
106          free(pesme[k].izvodjac);
107          free(pesme[k].naslov);
108      }
109      free(pesme);
110      return 0;
111  }
112
113  }
114
115  /* Ako jeste, nastavljamo sa citanjem pesama ... */
116  /* Citamo ime izvodjaca */
117
118  j = 0;                                /* Oznacava poziciju na koju
119                                         treba smestiti procitani
120                                         karakter */
121
122  alocirano = 0;                        /* Oznacava broj alociranih
123                                         mesta */
124  pesme[i].izvodjac = NULL;             /* Memoriya koju mozemo
125                                         koristiti za smestanje
126                                         procitanih karaktera */
127
128  /* Sve dok ne stignemo do prve beline u liniji - beline koja
129     se nalazi nakon imena izvodjaca - citamo karaktere iz
130     datoteke */
131  while ((c = fgetc(ulaz)) != ' ') {
132
133      /* Proveravamo da li imamo dovoljno memorije za smestanje
134         procitanog karaktera */
135      if (j == alocirano) {
136
137          /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
138             alociramo novih KORAK mesta */
139          alocirano += KORAK;
140          pesme[i].izvodjac =
141              (char *) realloc(pesme[i].izvodjac,
142                              alocirano * sizeof(char));
143
144          /* Proveravamo da li je nova alokacija uspesna */
145          if (pesme[i].izvodjac == NULL) {
146              /* Ako nije... */
147              /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
148              for (k = 0; k < i; k++) {
149                  free(pesme[k].izvodjac);
150                  free(pesme[k].naslov);
151              }
152              free(pesme);
153              /* I prekidamo sa izvrsavanjem programa */
154              return 0;
155          }
156      }
157      pesme[i].izvodjac[j] = c;
158      j++;
159  }
```

```
156     }
157     /* Ako imamo dovoljno memorije, smestamo procitani
158        karakter */
159     pesme[i].izvodjac[j] = c;
160     j++;
161     /* I nastavljamo sa citanjem */
162 }

163
164 /* Upisujemo terminirajucu nulu na kraju reci */
165 pesme[i].izvodjac[j] = '\0';
166
167
168 /* Citamo - */
169 fgetc(ulaz);
170
171 /* Citamo razmak */
172 fgetc(ulaz);
173
174
175 /* Citamo naslov pesme */
176 j = 0;                                /* Oznacava poziciju na koju
177                                         treba smestiti procitani
178                                         karakter */
179
180 alocirano = 0;                        /* Oznacava broj alociranih
181                                         mesta */
182
183 pesme[i].naslov = NULL;               /* Memorija koju mozemo
184                                         koristiti za smestanje
185                                         procitanih karaktera */
186
187 /* Sve dok ne stignemo do zareza - zareza koji se nalazi
188    nakon naslova pesme - citamo karaktere iz datoteke */
189
190 while ((c = fgetc(ulaz)) != ',') {
191     /* Proveravamo da li imamo dovoljno memorije za smestanje
192        procitanog karaktera */
193     if (j == alocirano) {
194         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
195            alociramo novih KORAK mesta */
196         alocirano += KORAK;
197         pesme[i].naslov =
198             (char *) realloc(pesme[i].naslov,
199                             alocirano * sizeof(char));
200
201         /* Proveravamo da li je nova alokacija uspesna */
202         if (pesme[i].naslov == NULL) {
203             /* Ako nije... */
204             /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
205             for (k = 0; k < i; k++) {
206                 free(pesme[k].izvodjac);
207                 free(pesme[k].naslov);
208             }
209             free(pesme[i].izvodjac);
```

```
208         free(pesme);
210         /* I prekidamo izvršavanje programa */
212         return 0;
214     }
216     /* Ako imamo dovoljno memorije, smestamo procitani
    karakter */
218     pesme[i].naslov[j] = c;
220     j++;
222     /* I nastavljamo dalje sa citanjem */
224     }
226     /* Upisujemo terminirajucu nulu na kraju reci */
228     pesme[i].naslov[j] = '\0';
230
232     /* Citamo razmak */
234     fgetc(ulaz);
236
238     /* Citamo broj gledanja */
240     broj_gledanja = 0;
242
244     /* Sve dok ne stignemo do znaka za novi red - kraja linije -
    citamo karaktere iz datoteke */
246     while ((c = fgetc(ulaz)) != '\n') {
248         broj_gledanja = broj_gledanja * 10 + (c - '0');
250     }
252     pesme[i].broj_gledanja = broj_gledanja;
254
256     /* Prelazimo na citanje sledece pesme */
258     i++;
260 }
262
264 /* Cuvamo informaciju o broju pesama koje smo procitali */
266 n = i;
268
270 /* Zatvaramo datoteku jer nam nece vise trebati */
272 fclose(ulaz);
274
276 /* Analiziramo argumente komandne linije */
278 if (argc == 1) {
280     /* Nema dodatnih opcija - sortiramo po broju gledanja */
282     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
284 } else {
286     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
288         /* Sortiramo po naslovu */
290         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
292     }
294 }
```

```

260     } else {
261         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
262             /* Sortiramo po izvodjaku */
263             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
264         } else {
265             printf("Nedozvoljeni argumenti!\n");
266             free(pesme);
267             return 0;
268         }
269     }
270 }

272 /* Ispisujemo rezultat */
273 for (i = 0; i < n; i++) {
274     printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
275           pesme[i].broj_gledanja);
276 }

278 /* Oslobadjamo memoriju */
279 for (i = 0; i < n; i++) {
280     free(pesme[i].izvodjac);
281     free(pesme[i].naslov);
282 }
283 free(pesme);
284
285 return 0;
286 }

```

### Rešenje 3.28

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <search.h>
5
6  #define MAX 100
7
8  /* Funkcija poredi dva cela broja */
9  int compare_int(const void *a, const void *b)
10 {
11     /* Konvertujemo void pokazivace u int pokazivace koje zatim
12     dereferenciramo, dobijamo int-ove koje oduzimamo i razliku
13     vracamo. */
14
15     /* Zbog moguceg prekoracenja opsega celih brojeva necemo ih
16     oduzimati return *((int *)a) - *((int *)b); */
17
18     int b1 = *((int *) a);
19     int b2 = *((int *) b);
20
21     if (b1 > b2)

```

```
    return 1;
23 else if (b1 < b2)
    /* Ovo uredjenje favorizujemo jer zelimo rastuci poredak */
25     return -1;
    else
27     return 0;
}

29 int compare_int_desc(const void *a, const void *b)
31 {
    /* Za obrnuti poredak mozemo samo oduzimati a od b */
33     /* return *((int *)b) - *((int *)a); */

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

39 /* Test program */
41 int main()
43 {
    size_t n;
    int i, x;
45     int a[MAX], *p = NULL;

    /* Unosimo dimenziju */
47     printf("Uneti dimenziju niza: ");
49     scanf("%ld", &n);
    if (n > MAX)
51         n = MAX;

    /* Unosimo elemente niza */
53     printf("Uneti elemente niza:\n");
55     for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

57     /* Sortiramo niz celih brojeva */
59     qsort(a, n, sizeof(int), &compare_int);

    /* Prikazujemo sortirani niz */
61     printf("Sortirani niz u rastucem poretku:\n");
63     for (i = 0; i < n; i++)
        printf("%d ", a[i]);
65     putchar('\n');

    /* Pretrazivanje niza */
    /* Vrednost koju cemo traziti u nizu */
69     printf("Uneti element koji se trazi u nizu: ");
    scanf("%d", &x);

71     /* Binarna pretraga */
73     printf("Binarna pretraga: \n");
```



```

75 p = bsearch(&x, a, n, sizeof(int), &compare_int);
76 if (p == NULL)
77     printf("Elementa nema u nizu!\n");
78 else
79     printf("Element je nadjen na poziciji %ld\n", p - a);
80
81 /* Linearna pretraga */
82 printf("Linearna pretraga (lfind): \n");
83 p = lfind(&x, a, &n, sizeof(int), &compare_int);
84 if (p == NULL)
85     printf("Elementa nema u nizu!\n");
86 else
87     printf("Element je nadjen na poziciji %ld\n", p - a);
88
89 return 0;
90 }

```

### Rešenje 3.29

```

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     /* Ako je argument negativan broj menjamo mu znak */
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     /* Sve dok je */
24     for (i = 2; i < sqrt(x); i++)
25         if (x % i == 0)
26             /* Ako i deli x onda su delioci: i, x/i */
27             br += 2;
28     /* Ako je broj bas kvadrat, onda smo iz petlje izašli kada je
29        i bilo bas jednako korenu od x, tada x ima jos jednog
30        delioca */
31     if (i * i == x)
32         br++;
33 }

```

```
34     return br;
35 }
36
37 /* Funkcija poredjenja dva cela broja po broju delilaca */
38 int compare_no_deviders(const void *a, const void *b)
39 {
40     int ak = *(int *) a;
41     int bk = *(int *) b;
42     int n_d_a = no_of_deviders(ak);
43     int n_d_b = no_of_deviders(bk);
44
45     if (n_d_a > n_d_b)
46         return 1;
47     else if (n_d_a < n_d_b)
48         return -1;
49     else
50         return 0;
51 }
52
53 /* Test program */
54 int main()
55 {
56     size_t n;
57     int i;
58     int a[MAX];
59
60     /* Unosimo dimenziju */
61     printf("Uneti dimenziju niza: ");
62     scanf("%ld", &n);
63     if (n > MAX)
64         n = MAX;
65
66     /* Unosimo elemente niza */
67     printf("Uneti elemente niza:\n");
68     for (i = 0; i < n; i++)
69         scanf("%d", &a[i]);
70
71     /* Sortiramo niz celih brojeva prema broju delilaca */
72     qsort(a, n, sizeof(int), &compare_no_deviders);
73
74     /* Prikazujemo sortirani niz */
75     printf
76         ("Sortirani niz u rastucem poretku prema broju delilaca:\n");
77     for (i = 0; i < n; i++)
78         printf("%d ", a[i]);
79     putchar('\n');
80
81     return 0;
82 }
```

## Rešenje 3.30

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <search.h>
5  #define MAX_NISKI 1000
6  #define MAX_DUZINA 30

8  /*****
   Niz nizova karaktera ovog potpisa
10 char niske[3][4];
   se moze graficki predstaviti ovako:
12 -----
   | a | b | c | \0 | | d | e | \0 |   | f | g | h | \0 |
14 -----
   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu.
16 Za svaku je rezervisano po 4 karaktera ukljucujuci \0.
   Druga rec sa nalazi na adresi koja je za 4 veca od prve reci,
18 a za 4 manja od adrese na kojoj se nalazi treca rec.
   Adresa i-te reci je niske[i] i ona je tipa char*.

20 Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
22 elemenata koji trebaju biti uporedjeni, (npr. pri porecenju
   prve i poslednje reci, pokazivac a ce pokazivati na slovo 'a',
24 a pokazivac b na slovo 'f') kastujemo ih na char*, i pozivamo
   funkciju strcmp nad njima.
26 *****/
int poredi_leksikografski(const void *a, const void *b)
28 {
   return strcmp((char *) a, (char *) b);
30 }

32 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
34 int poredi_duzine(const void *a, const void *b)
   {
36     return strlen((char *) a) - strlen((char *) b);
   }

38
40 int main()
   {
42     int i;
     size_t n;
     FILE *fp = NULL;
44     char niske[MAX_NISKI][MAX_DUZINA];
     char *p = NULL;
46     char x[MAX_DUZINA];

48     /* Otvaranje datoteke */
     if ((fp = fopen("niske.txt", "r")) == NULL) {
50         fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");

```

```
    exit(EXIT_FAILURE);
52 }

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

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

60 /* Sortiramo niske leksikografski, tako sto bibliotečkoj
62 funkciji qsort prosledjujemo funkciju kojom se zadaje
    kriterijum poredjenja 2 niske po dužini */
64 qsort(niske, n, MAX_DUZINA * sizeof(char),
    &poredi_leksikografski);

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

72 /* Unosimo trazeni nisku */
printf("Uneti trazenu nisku: ");
74 scanf("%s", x);

76 /* Binarna pretraga */
/* Prosledjujemo pokazivac na funkciju poredi_leksikografski
78 jer nam je niz sortiran leksikografski. */
p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
80 &poredi_leksikografski);

82 if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
84 p, (p - (char *) niske) / MAX_DUZINA);
else
86 printf("Niska nije pronadjena u nizu\n");

88 /* Linearna pretraga */
/* Prosledjujemo pokazivac na funkciju poredi_leksikografski
90 jer nam je niz sortiran leksikografski. */
p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
92 &poredi_leksikografski);

94 if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
96 p, (p - (char *) niske) / MAX_DUZINA);
else
98 printf("Niska nije pronadjena u nizu\n");

100 /* Sada ih sortiramo po dužini i ovaj put saljemo drugu
    funkciju poredjenja */
102 qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
```

```

104 printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
106     printf("%s ", niske[i]);
    printf("\n");
108
    exit(EXIT_SUCCESS);
110 }

```

### Rešenje 3.31

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #include <search.h>
#define MAX_NISKI 1000
6 #define MAX_DUZINA 30

/*****
Niz pokazivaca na karaktere ovog potpisa
char *niske[3];
posle alokacije u main-u se moze graficki predstaviti ovako:
12
| X | -----> | a | b | c | \0|
14
| Y | -----> | d | e | \0|
16
| Z | -----> | f | g | h | \0|
18
Sa leve strane je vertikalno prikazan niz pokazivaca, gde
20 je i-ti njegov element pokazivac koji pokazuje na alocirane
karaktere i-te reci. Njegov tip je char*.

22
Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
24 elemenata koji trebaju biti uporedjeni (recimo adresu od X
i adresu od Z), i kako su X i Z tipa char*, onda a i b su
26 tipa char**, pa ih tako moramo i kastovati. Da bi uporedili
leksikografski elemente X i Z, moramo uporediti stringove
28 na koje oni pokazuju, pa zato u sledecoj funkciji pozivamo
strcmp() nad onim na sta pokazuju a i b, kastovani na
30 odgovarajuci tip.
*****/
32 int poredi_leksikografski(const void *a, const void *b)
{
34     return strcmp(*(char **) a, *(char **) b);
}

36
/* Funkcija slicna prethodnoj, osim sto elemente ne uporeduje
38 leksikografski, vec po duzini */
int poredi_duzine(const void *a, const void *b)
40 {

```

### 3 Algoritmi pretrage i sortiranja

---

```
42     return strlen(*(char **) a) - strlen(*(char **) b);
43 }
44 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje
45    na element u nizu sa kojim se poredi, pa njega treba
46    kastovati na char** i dereferencirati, (videti obrazlozenje
47    za prvu funkciju u ovom zadatku, a pokazivac a pokazuje na
48    element koji se trazi. U main funkciji je to x, koji je tipa
49    char*, tako da pokazivac a ovde samo kastujemo i ne
50    dereferenciramo. */
51 int poredi_leksikografski_b(const void *a, const void *b)
52 {
53     return strcmp((char *) a, *(char **) b);
54 }
55
56 int main()
57 {
58     int i;
59     size_t n;
60     FILE *fp = NULL;
61     char *niske[MAX_NISKI];
62     char **p = NULL;
63     char x[MAX_DUZINA];
64
65     /* Otvaranje datoteke */
66     if ((fp = fopen("niske.txt", "r")) == NULL) {
67         fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
68         exit(EXIT_FAILURE);
69     }
70
71     /* Citanje sadrzaja datoteke */
72     i = 0;
73     while (fscanf(fp, "%s", x) != EOF) {
74         /* Alociranje dovoljne memorije za i-tu nisku */
75         if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
76             fprintf(stderr, "Greska pri alociranju niske\n");
77             exit(EXIT_FAILURE);
78         }
79         /* Kopiranje procitane niske na svoje mesto */
80         strcpy(niske[i], x);
81         i++;
82     }
83
84     /* Zatvaranje datoteke */
85     fclose(fp);
86     n = i;
87
88     /* Sortiramo niske leksikografski, tako sto biblioteckoj
89        funkciji qsort prosledjujemo funkciju kojom se zadaje
90        kriterijum poredjenja 2 niske po duzini */
91     qsort(niske, n, sizeof(char *), &poredi_leksikografski);
92 }
```

```

124 printf("Leksikografski sortirane niske:\n");
125 for (i = 0; i < n; i++)
126     printf("%s ", niske[i]);
127 printf("\n");

128 /* Unosimo trazeni nisku */
129 printf("Uneti trazenu nisku: ");
130 scanf("%s", x);

131 /* Binarna pretraga */
132 /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
133     jer nam je niz sortiran leksikografski. */
134 p = bsearch(x, niske, n, sizeof(char *),
135             &poredi_leksikografski_b);

136 if (p != NULL)
137     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
138           *p, p - niske);
139 else
140     printf("Niska nije pronadjena u nizu\n");

141 /* Linearna pretraga */
142 /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
143     jer nam je niz sortiran leksikografski. */
144 p = lfind(x, niske, &n, sizeof(char *),
145           &poredi_leksikografski_b);

146 if (p != NULL)
147     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
148           *p, p - niske);
149 else
150     printf("Niska nije pronadjena u nizu\n");

151 /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
152     funkciju poredjenja */
153 qsort(niske, n, sizeof(char *), &poredi_duzine);

154 printf("Niske sortirane po duzini:\n");
155 for (i = 0; i < n; i++)
156     printf("%s ", niske[i]);
157 printf("\n");

158 /* Oslobadjanje zauzete memorije */
159 for (i = 0; i < n; i++)
160     free(niske[i]);

161 exit(EXIT_SUCCESS);
162 }

```

## Rešenje 3.32

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <search.h>
5
6 #define MAX 500
7
8 /* Struktura koja nam je neophodna za sve informacije o
9    pojedinačnom studentu */
10 typedef struct {
11     char ime[21];
12     char prezime[21];
13     int bodovi;
14 } Student;
15
16 /* Funkcija poredjenja koju cemo koristiti za sortiranje po
17    broju bodova, a studente sa istim brojevem bodova dodatno
18    sortiramo leksikografski po prezimenu */
19 int compare(const void *a, const void *b)
20 {
21     Student *prvi = (Student *) a;
22     Student *drugi = (Student *) b;
23
24     if (prvi->bodovi > drugi->bodovi)
25         return -1;
26     else if (prvi->bodovi < drugi->bodovi)
27         return 1;
28     else
29         /* Jednaki su po broju bodova, treba ih uporediti po
30            prezimenu */
31         return strcmp(prvi->prezime, drugi->prezime);
32 }
33
34 /* Funkcija za poredjenje koje ce porediti samo po broju bodova
35    Prvi parametar je ono sto trazimo u nizu, ovde je to broj
36    bodova, a drugi parametar ce biti element niza ciji se bodovi
37    porede. */
38 int compare_zabsearch(const void *a, const void *b)
39 {
40     int bodovi = *(int *) a;
41     Student *s = (Student *) b;
42     return s->bodovi - bodovi;
43 }
44
45 /* Funkcija za poredjenje koje ce porediti samo po prezimenu
46    Prvi parametar je ono sto trazimo u nizu, ovde je to prezime,
47    a drugi parametar ce biti element niza cije se prezime
48    porede. */
49 int compare_zalinearaprezimena(const void *a, const void *b)
50 {
51     char *prezime = (char *) a;
```



```
Student *s = (Student *) b;
53 return strcmp(prezime, s->prezime);
55 }

57 int main(int argc, char *argv[])
58 {
59     Student kolokvijum[MAX];
60     int i;
61     size_t br_studenata = 0;
62     Student *nadjen = NULL;
63     FILE *fp = NULL;
64     int bodovi;
65     char prezime[21];

67     /* Ako je program pozvan sa nedovoljnim brojem argumenata
        informisemo korisnika kako se program koristi i prekidamo
        izvršavanje. */
69     if (argc < 2) {
71         fprintf(stderr,
72             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
73             argv[0]);
74         exit(EXIT_FAILURE);
75     }

77     /* Otvaranje datoteke */
78     if ((fp = fopen(argv[1], "r")) == NULL) {
79         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
80         exit(EXIT_FAILURE);
81     }

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

89     /* Zatvaranje datoteke */
90     fclose(fp);
91     br_studenata = i;

93     /* Sortiramo niz studenata po broju bodova, pa unutar grupe
        studenata sa istim brojem bodova sortiranje se vrši po
        prezimenu */
95     qsort(kolokvijum, br_studenata, sizeof(Student), &compare);

97     printf("Studenti sortirani po broju poena opadajuće, ");
98     printf("pa po prezimenu rastuće:\n");
99     for (i = 0; i < br_studenata; i++)
101         printf("%s %s %d\n", kolokvijum[i].ime,
102             kolokvijum[i].prezime, kolokvijum[i].bodovi);
103 }
```

### 3 Algoritmi pretrage i sortiranja

```
105  /* Pretrazivanje studenata po broju bodova se vrši binarnom
    printf("Unesite broj bodova: ");
107  scanf("%d", &bodovi);

109  nadjen =
        bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
111          &compare_za_bsearch);

113  if (nadjen != NULL)
        printf
115      ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
117  else
        printf("Nema studenta sa unetim brojem bodova\n");
119
121  /* Pretraga po prezimenu se mora vršiti linearnom pretragom
    jer nam je niz sortiran po bodovima, globalno gledano. */
    printf("Unesite prezime: ");
123  scanf("%s", prezime);

125  nadjen =
        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
127          &compare_za_linearna_prezimana);

129  if (nadjen != NULL)
        printf
131      ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
133  else
        printf("Nema studenta sa unetim prezimenom\n");
135
        return 0;
137 }
```

#### Rešenje 3.33

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

5  #define MAX 128

7  /* Funkcija poredi dva karaktera */
    int uporedi_char(const void *pa, const void *pb)
9  {
        return *(char *) pa - *(char *) pb;
11 }

13 /* Funkcija vraća: 1 - ako jesu anagrami 0 - inace */
    int anagrami(char s[], char t[], int n_s, int n_t)
```

```

15 {
16     /* Ako dve niske imaju razlicitu duzinu => nisu anagrami */
17     if (n_s != n_t)
18         return 0;
19
20     /* Sortiramo niske */
21     qsort(s, strlen(t) / sizeof(char), sizeof(char),
22           &uporedi_char);
23     qsort(t, strlen(t) / sizeof(char), sizeof(char),
24           &uporedi_char);
25
26     /* Ako su niske nakon sortiranja iste => jesu anagrami, u
27        suprotnom, nisu */
28     return !strcmp(s, t);
29 }
30
31 int main()
32 {
33     char s[MAX], t[MAX];
34
35     /* Unose se dve niske sa ulaza */
36     printf("Unesite prvu nisku: ");
37     scanf("%s", s);
38
39     printf("Unesite drugu nisku: ");
40     scanf("%s", t);
41
42     /* Ispituje se da li su niske anagrami */
43     if (anagrami(s, t, strlen(s), strlen(t)))
44         printf("jesu\n");
45     else
46         printf("nisu\n");
47
48     return 0;
49 }

```

### Rešenje 3.34

```

#include <stdio.h>
2 #include <string.h>
#include <stdlib.h>
4
#define MAX 10
6 #define MAX_DUZINA 32
7
8 /* Funkcija poredi dve niske (stringove) */
9 int uporedi_niske(const void *pa, const void *pb)
10 {
11     return strcmp((char *) pa, (char *) pb);
12 }

```

### 3 Algoritmi pretrage i sortiranja

```
14 int main()
15 {
16     int i, n;
17     char S[MAX][MAX_DUZINA];
18
19     /* Unos broja niski */
20     printf("Unesite broj niski:");
21     scanf("%d", &n);
22
23     /* Unos niza niski */
24     printf("Unesite niske:\n");
25     for (i = 0; i < n; i++)
26         scanf("%s", S[i]);
27
28     /* Sortiramo niz niski */
29     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
30
31     /******
32     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
33     sortiranih niski. Koriscen je samo u fazi testiranja programa.
34
35     printf("Sortirane niske su:\n");
36     for(i = 0; i < n; i++)
37         printf("%s ", S[i]);
38     *****/
39
40     /* Ako postoje dve iste niske u nizu, onda ce one nakon
41     sortiranja niza biti jedna do druge */
42     for (i = 0; i < n - 1; i++)
43         if (strcmp(S[i], S[i + 1]) == 0) {
44             printf("ima\n");
45             return 0;
46         }
47
48     printf("nema\n");
49     return 0;
50 }
```

#### Rešenje 3.35

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 #define MAX 21
6
7 /* Struktura koja predstavlja jednog studenta */
8 typedef struct student {
9     char nalog[8];
10    char ime[MAX];
11    char prezime[MAX];
12 }
```

```
13     int poeni;  
14 } Student;  
  
15  
16 /* Funkcija poredi studente prema broju poena, rastuce */  
17 int uporedi_poeni(const void *a, const void *b)  
18 {  
19     Student s = *(Student *) a;  
20     Student t = *(Student *) b;  
21     return s.poeni - t.poeni;  
22 }  
  
23  
24 /* Funkcija poredi studente prvo prema godini, zatim prema smeru  
25    i na kraju prema indeksu */  
26 int uporedi_nalog(const void *a, const void *b)  
27 {  
28     Student s = *(Student *) a;  
29     Student t = *(Student *) b;  
30     /* Za svakog studenta iz naloga izdvajamo godinu upisa, smer i  
31        broj indeksa */  
32     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';  
33     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';  
34     char smer1 = s.nalog[1];  
35     char smer2 = t.nalog[1];  
36     int indeks1 =  
37         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +  
38         s.nalog[6] - '0';  
39     int indeks2 =  
40         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +  
41         t.nalog[6] - '0';  
42     if (godina1 != godina2)  
43         return godina1 - godina2;  
44     else if (smer1 != smer2)  
45         return smer1 - smer2;  
46     else  
47         return indeks1 - indeks2;  
48 }  
49  
50  
51 int uporedi_bsearch(const void *a, const void *b)  
52 {  
53     /* Nalog studenta koji se trazi */  
54     char *nalog = (char *) a;  
55     /* Kljuc pretrage */  
56     Student s = *(Student *) b;  
57  
58     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';  
59     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';  
60     char smer1 = nalog[1];  
61     char smer2 = s.nalog[1];  
62     int indeks1 =  
63         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] -
```

```
        '0';
65     int indeks2 =
        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
67         s.nalog[6] - '0';
    if (godina1 != godina2)
69         return godina1 - godina2;
    else if (smer1 != smer2)
71         return smer1 - smer2;
    else
73         return indeks1 - indeks2;
}

75 int main(int argc, char **argv)
77 {
    Student *nadjen = NULL;
79     char nalog_trazeni[8];
    Student niz_studenata[100];
81     int i = 0, br_studenata = 0;
    FILE *in = NULL, *out = NULL;

83
    /* Ako je broj argumenata komandne linije razlicit i od 2 i od
85     3, korisnik nije ispravno pozvao program i prijavljujemo
    gresku: */
87     if (argc != 2 && argc != 3) {
        fprintf(stderr,
89             "Greska! Program se poziva sa: ./a.out -opcija (nalog)!\n
        ");
        exit(EXIT_FAILURE);
91     }

93     /* Otvaranje datoteke za citanje */
    in = fopen("studenti.txt", "r");
95     if (in == NULL) {
        fprintf(stderr,
97             "Greska prilikom otvaranja datoteke studenti.txt!\n");
        exit(EXIT_FAILURE);
99     }

101    /* Otvaranje datoteke za pisanje */
    out = fopen("izlaz.txt", "w");
103    if (out == NULL) {
        fprintf(stderr,
105             "Greska prilikom otvaranja datoteke izlaz.txt!\n");
        exit(EXIT_FAILURE);
107    }

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

```

115     br_studenata = i;
117
118     /* Ako je student uneo opciju -p, vrsi se sortiranje po
119        poenima */
120     if (strcmp(argv[1], "-p") == 0)
121         qsort(niz_studenata, br_studenata, sizeof(Student),
122             &uporedi_poeni);
123     /* A ako je uneo opciju -n, vrsi se sortiranje po nalogu */
124     else if (strcmp(argv[1], "-n") == 0)
125         qsort(niz_studenata, br_studenata, sizeof(Student),
126             &uporedi_nalog);
127
128     /* Sortirani studenti se ispisuju u izlaznu datoteku */
129     for (i = 0; i < br_studenata; i++)
130         fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
131             niz_studenata[i].ime, niz_studenata[i].prezime,
132             niz_studenata[i].poeni);
133
134     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
135        studenta... */
136     if (argc == 3 && (strcmp(argv[1], "-n") == 0)){
137         strcpy(nalog_trazeni, argv[2]);
138
139         /* ... pronalazi se student sa tim nalogom... */
140         nadjen =
141             (Student *) bsearch(nalog_trazeni, niz_studenata,
142                 br_studenata, sizeof(Student),
143                 &uporedi_bsearch);
144
145         if (nadjen == NULL)
146             printf("Nije nadjen!\n");
147         else
148             printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
149                 nadjen->prezime, nadjen->poeni);
150     }
151
152     /* Zatvaranje datoteka */
153     fclose(in);
154     fclose(out);
155
156     return 0;
157 }

```

### Rešenje 3.37

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
   standardnog ulaza */

```

### 3 Algoritmi pretrage i sortiranja

---

```
6 void ucitaj_matricu(int **a, int n, int m)
{
8     printf("Unesite elemente matrice po vrstama:\n");
    int i, j;

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

18 /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
    kolona */
20 int zbir_vrste(int **a, int v, int m)
{
22     int i, zbir = 0;

24     for (i = 0; i < m; i++) {
        zbir += a[v][i];
26     }
    return zbir;
28 }

30 /* Funkcija koja sortira vrste matrice na osnovu zbira
    koriscenjem selection algoritma */
32 void sortiraj_vrste(int **a, int n, int m)
{
34     int i, j, min;

36     for (i = 0; i < n - 1; i++) {
        min = i;
38         for (j = i + 1; j < n; j++) {
            if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
40                 min = j;
            }
42         }
        if (min != i) {
44             int *tmp;
            tmp = a[i];
46             a[i] = a[min];
            a[min] = tmp;
48         }
    }
50 }

52 /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
    standardni izlaz */
54 void ispisi_matricu(int **a, int n, int m)
{
56     int i, j;
```



```
58     for (i = 0; i < n; i++) {
59         for (j = 0; j < m; j++) {
60             printf("%d ", a[i][j]);
61         }
62         printf("\n");
63     }
64 }

66 /* Funkcija koja alocira memoriju za matricu dimenzija nxm */
67 int **alociraj_memoriju(int n, int m)
68 {
69     int i, j;
70     int **a;

72     a = (int **) malloc(n * sizeof(int *));
73     if (a == NULL) {
74         fprintf(stderr, "Problem sa alokacijom memorije!\n");
75         exit(EXIT_FAILURE);
76     }
77     /* Za svaku vrstu ponaosob */
78     for (i = 0; i < n; i++) {
80
81         /* Alociramo memoriju */
82         a[i] = (int *) malloc(m * sizeof(int));

83         /* Proveravamo da li je doslo do greske prilikom alokacije */
84         if (a[i] == NULL) {
85             /* Ako jeste, ispisujemo poruku */
86             fprintf(stderr, "Problem sa alokacijom memorije!\n");

87             /* I oslobadjamo memoriju zauzetu do ovog koraka */
88             for (j = 0; j < i; j++) {
89                 free(a[j]);
90             }
91             free(a);
92             exit(EXIT_FAILURE);
93         }
94     }
95 }

96
97 return a;
98 }

100 /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije
101    nxm */
102 void oslobodi_memoriju(int **a, int n, int m)
103 {
104     int i;

105     for (i = 0; i < n; i++) {
106         free(a[i]);
107     }
108 }
```

```
110     free(a);
111 }
112
113
114 int main(int argc, char *argv[])
115 {
116     int **a;
117     int n, m;
118
119
120     /* Citamo dimenzije matrice */
121     printf("Unesite dimenzije matrice: ");
122     scanf("%d %d", &n, &m);
123
124     /* Alociramo memoriju */
125     a = alociraj_memoriju(n, m);
126
127
128     /* Ucitavamo elemente matrice */
129     ucitaj_matricu(a, n, m);
130
131     /* Pozivamo funkciju koja sortira vrste matrice prema zbiru */
132     sortiraj_vrste(a, n, m);
133
134     /* Ispisujemo rezultujucu matricu */
135     printf("Sortirana matrica je:\n");
136     ispisi_matricu(a, n, m);
137
138     /* Oslobadjamo memoriju */
139     oslobodi_memoriju(a, n, m);
140
141     return 0;
142 }
```

## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

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

- (a) Definirati strukturu `Cvor` koja predstavlja čvor liste.
- (b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vraća adresu kreiranog čvora.
- (c) Napisati funkciju koja dodaje novi elemenat na početak liste.
- (d) Napisati funkciju koja dodaje novi elemenat na kraj liste.
- (e) Napisati funkciju koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (f) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama `[, ]` i međusobno razdvojene zapetama.
- (g) Napisati funkciju koja proverava da li se u listi nalazi element čija se vrednost zadaje kao argument funkcije.
- (h) Napisati funkciju koja proverava da li se u listi nalazi element čija se vrednost zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenoj listi.

- (i) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja se zadaje kao argument funkcije.
- (j) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja se zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenoj listi.
- (k) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

*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.

### *Upotreba programa 1*

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 2 3 14 5 3 3 17 3 1 9
  Unosite element koji se trazi u listi: 17
Izlaz:
Lista: []
Lista: [2]
Lista: [3, 2]
Lista: [14, 3, 2]
Lista: [5, 14, 3, 2]
Lista: [3, 5, 14, 3, 2]
Lista: [3, 3, 5, 14, 3, 2]
Lista: [17, 3, 3, 5, 14, 3, 2]
Lista: [3, 17, 3, 3, 5, 14, 3, 2]
Lista: [1, 3, 17, 3, 3, 5, 14, 3, 2]
Lista: [9, 1, 3, 17, 3, 3, 5, 14, 3, 2]

  Trazeni broj 17 je u listi!
```

### *Upotreba programa 2*

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 23 14 35
  Unosite element koji se trazi u listi: 8
Izlaz:
Lista: []
Lista: [23]
Lista: [14, 23]
Lista: [35, 14, 23]

  Element nije u listi!
```

*Upotreba programa 3*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D):
  Unosite element koji se trazi u listi: 1
Izlaz:
  Lista: []

  Element nije 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.

*Upotreba programa 1*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 2 3 14 5 3 3 17 3 1 3
  Unosite element koji se briše iz liste: 3
Izlaz:
  Lista: []
  Lista: [2]
  Lista: [2, 3]
  Lista: [2, 3, 14]
  Lista: [2, 3, 14, 5]
  Lista: [2, 3, 14, 5, 3]
  Lista: [2, 3, 14, 5, 3, 3]
  Lista: [2, 3, 14, 5, 3, 3, 17]
  Lista: [2, 3, 14, 5, 3, 3, 17, 3]
  Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1]
  Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1, 3]

  Lista nakon brisanja: [2, 14, 5, 17, 1]

```

*Upotreba programa 2*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 23 14 35
  Unosite element koji se briše iz liste: 3
Izlaz:
  Lista: []
  Lista: [23]
  Lista: [23, 14]
  Lista: [23, 14, 35]

  Lista nakon brisanja: [23, 14, 35]

```

### Upotreba programa 3

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D):
  Unosite element koji se briše iz liste: 12
Izlaz:
  Lista: []

  Lista nakon brisanja: []
```

- (3) U glavnom programu se učitani celi brojevi dodaju u listu tako da je elementi budu uređeni u neopadajućem poretku. Unosi 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 i brisanja elementa liste koristiti činjenicu da je lista uređena.*

### Upotreba programa 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 2 3 14 5 3 3 17 3 1 9
  Unosite element koji se traži u listi: 5
  Unosite element koji se briše iz liste: 3
Izlaz:
  Lista: []
  Lista: [2]
  Lista: [2, 3]
  Lista: [2, 3, 14]
  Lista: [2, 3, 5, 14]
  Lista: [2, 3, 3, 5, 14]
  Lista: [2, 3, 3, 3, 5, 14]
  Lista: [2, 3, 3, 3, 5, 14, 17]
  Lista: [2, 3, 3, 3, 3, 5, 14, 17]
  Lista: [1, 2, 3, 3, 3, 3, 5, 14, 17]
  Lista: [1, 2, 3, 3, 3, 3, 5, 9, 14, 17]

  Trazeni broj 5 je u listi!
  Lista nakon brisanja: [1, 2, 5, 9, 14, 17]
```

### Upotreba programa 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D):
  Unosite element koji se traži u listi: 1
  Unosite element koji se briše iz liste: 12
Izlaz:
  Lista: []

  Element nije u listi!
  Lista nakon brisanja: []
```

*Upotreba programa 3*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 23 14 35
  Unosite element koji se trazi u listi: 8
  Unosite element koji se brise iz liste: 3
Izlaz:
  Lista: []
  Lista: [23]
  Lista: [14, 23]
  Lista: [14, 23, 35]

  Element nije u listi!
  Lista nakon brisanja: [14, 23, 35]

```

[Rešenje 4.1]

**Zadatak 4.2** Napisati biblioteku za rad sa jednosturko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekurzivno. *Napomena: koristiti iste main programe i upotrebe programa iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva, koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

- (a) Napisati funkciju koja iz prosledene liste briše element na koju pokazuje pokazivač koji se funkciji šalje kao argument.
- (b) Napisati funkciju koja ispisuje sadržaj liste od poslednjeg elementa ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. *Napomena: koristiti iste main programe i upotrebe programa iz zadatka 4.1. Ove programe dopuniti pozivom funkcije koja ispisuje listu u nazad.*

[Rešenje 4.3]

**Zadatak 4.4** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke `dat.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

### Test 1

```
Datoteka:
{[23 + 5344] * (24 - 234)} - 23
Izlaz:
  Zagrade su ispravno uparene.
```

### Test 2

```
Datoteka:
{[23 + 5] * (9 * 2)} - {23}
Izlaz:
  Zagrade su ispravno uparene.
```

### Test 3

```
Datoteka:
{[2 + 54] / (24 * 87)} + (234 + 23)
Izlaz:
  Zagrade nisu ispravno uparene.
```

### Test 3

```
Datoteka:
{(2 - 14) / (23 + 11)} * (2 + 13)
Izlaz:
  Zagrade nisu ispravno uparene.
```

### Test 4

```
Datoteka je prazna.
Izlaz:
  Zagrade su ispravno uparene.
```

### Test 5

```
Datoteka ne postoji.
Izlaz:
  Greska prilikom otvaranja
  datoteke izraz.txt!
```

[Rešenje 4.4]

**Zadatak 4.5** 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 su čvorovi HTML etikete.*

### 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:
  Ispravno uparene etikete.
```



*Test 2*

```

|| Poziv: ./a.out
|| Izlaz:
||      Greska! Program se poziva
||      sa: ./a.out datoteka.html!

```

*Test 3*

```

|| Poziv: ./a.out datoteka.html
|| Datoteka.html ne postoji.
|| Izlaz:
||      Greska prilikom otvaranja
||      datoteke datoteka.html.

```

*Test 4*

```

|| Poziv: ./a.out datoteka.html
|| Datoteka.html:
|| <html>
||   <head>
||     <title>Primer</title>
||   </head>
||   <body>
|| </html>
|| Izlaz:
||      Neispravno uparene etikete.

```

*Test 5*

```

|| Poziv: ./a.out datoteka.html
|| Datoteka.html:
|| <html>
||   <head>
||     <title>Primer</title>
||   </head>
||   <body>
|| </body>
|| Izlaz:
||      Neispravno uparene etikete.

```

*Test 6*

```

|| Poziv: ./a.out datoteka.html
|| Datoteka.html je prazna
|| Izlaz:
||      Ispravno uparene etikete.

```

[Rešenje 4.5]

**Zadatak 4.6** Napisati program kojim se simulira rad jednog šaltera na kojem se prvo kod službenika zakazuju termini, a potom službenik uslužuje korisnike. Službenik evidentira 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, tj. službeniku se 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**, tada 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 5 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.*

Milena: Ova upotreba mora da se skрати tako da stane na jednu stranu. To znaci da treba da se eliminiše bar 5 redova.

### Upotreba programa 1

Sluzbenik evidentira korisnicke zahteve unosnjem njihovog JMBG broja i opisa potrebne usluge:

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: 4567890123456  
Opis problema: Zatvaranje racuna

Novi zahtev [CTRL+D za kraj]  
JMBG: 5678901234567  
Opis problema: Podizanje kredita

Novi zahtev [CTRL+D za kraj]  
JMBG:  
Sledeci je korisnik sa JMBG brojem: 1234567890123  
sa zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG brojem: 2345678901234  
sa zahtevom: Podizanje novca  
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 3456789012345  
sa zahtevom: Reklamacija  
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG brojem: 4567890123456  
sa zahtevom: Zatvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 5678901234567  
sa zahtevom: Podizanje kredita  
Da li ga vracate na kraj reda? [Da/Ne] Da

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 1234567890123  
sa zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 3456789012345  
sa zahtevom: Reklamacija  
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 5678901234567  
sa zahtevom: Podizanje kredita  
Da li ga vracate na kraj reda? [Da/Ne] Ne

izvestaj.txt:  
JMBG: 2345678901234      Zahtev: Podizanje novca  
JMBG: 4567890123456      Zahtev: Zatvaranje racuna  
JMBG: 1234567890123      Zahtev: Otvaranje racuna  
JMBG: 3456789012345      Zahtev: Reklamacija  
JMBG: 5678901234567      Zahtev: Podizanje kredita

[Rešenje 4.6]

**Zadatak 4.7** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etiketke smestati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
{
    unsigned broj_pojavljivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;
```

*Test 1*

<pre>   Poziv: ./a.out datoteka.html    Datoteka.html:    &lt;html&gt;      &lt;head&gt;&lt;title&gt;Primer&lt;/title&gt;&lt;/head&gt;      &lt;body&gt;        &lt;h1&gt;Naslov&lt;/h1&gt;        Danas je lep i suncan dan. &lt;br&gt;        A sutra ce biti jos lepsi.        &lt;a link="http://www.google.com"&gt; Link 1&lt;/a&gt;        &lt;a link="http://www.math.rs"&gt; Link 2&lt;/a&gt;      &lt;/body&gt;    &lt;/html&gt;</pre>	<pre>Izlaz: a - 4 br - 1 h1 - 2 body - 2 title - 2 head - 2 html - 2</pre>
---	--

*Test 2*

```
|| Poziv: ./a.out
|| Izlaz:
|| Greska! Program se poziva
|| sa: ./a.out datoteka.html!
```

*Test 3*

```
|| Poziv: ./a.out datoteka.html
|| Datoteka.html ne postoji.
|| Izlaz:
|| Greska prilikom otvaranja
|| datoteke datoteka.html.
```

*Test 4*

```
|| Poziv: ./a.out datoteka.html
|| Datoteka.html je prazna.
|| Izlaz:
```

[Rešenje 4.7]

**Zadatak 4.8** Napisati program koji objedinjuje dve sortirane liste. Funkcija ne treba da kreira nove čvorove, već da samo postojeće čvorove preraspodeli. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije.

## 4 Dinamičke strukture podataka

---

Rezultujuću listu ispisati na standardni izlaz. *Napomena: koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

### Test 1

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt:
  2 4 6 10 15
dat2.txt:
  5 6 11 12 14 16
Izlaz:
  2 4 5 6 6 10 11 12 14 15 16
```

### Test 2

```
Poziv: ./a.out
Izlaz:
  Greska! Program se poziva sa:
  ./a.out dat1.txt dat2.txt!
```

### Test 3

```
Poziv: ./a.out dat1.txt
Izlaz:
  Greska! Program se poziva sa:
  ./a.out dat1.txt dat2.txt!
```

### Test 4

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt:
  2 4 6 10 15
dat2.txt ne postoji
Izlaz:
  Greska prilikom otvaranja datoteke
  dat2.txt.
```

### Test 5

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt ne postoji
dat2.txt: 2 4 6 10 15
Izlaz:
  Greska prilikom otvaranja datoteke
  dat1.txt.
```

### Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz:
  2 4 6 10 15
```

[Rešenje 4.8]

**Zadatak 4.9** Napisati funkciju koja dodaje na početak liste studenata novog studenta za koga su nam poznati broj indeksa, ime i prezime. Napisati rekurzivnu funkciju koja određuje da li neki student pripada listi ili ne. U glavnom programu učitati u listu sve podatke o studentima iz datoteke čije se ime zadaje kao argument komandne linije. U svakom redu datoteke nalaze se podaci o studentu i to broj indeksa, ime i prezime. Nakon toga se sa standardnog ulaza unose, jedan po jedan, indeksi studenata koji se traže u učitanj 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). Nakon završenog pretraživanja rekurzivnom funkcijom osloboditi memoriju koju je data lista zauzimala. *Uputstvo: Pretpostaviti da je 10 karaktera dovoljno za zapis indeks i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

Milena: lepse formulisati zadatak. Poslednji test primer je neispravan.

*Test 1*

<code>Poziv: ./a.out studenti.txt</code>		
<code>studenti.txt:</code>	<code>Ulaz:</code>	<code>Izlaz:</code>
<code>123/2014 Marko Lukic</code>	<code>3/2014</code>	<code>da: Ana Sokic</code>
<code>3/2014 Ana Sokic</code>	<code>235/2008</code>	<code>ne</code>
<code>43/2013 Jelena Ilic</code>	<code>41/2009</code>	<code>da: Marija Zaric</code>
<code>41/2009 Marija Zaric</code>		
<code>13/2010 Milovan Lazic</code>		

*Test 2*

```

Poziv: ./a.out
Izlaz:
  Greska! Program se poziva sa:
  ./a.out studenti.txt!

```

*Test 5*

```

Poziv: ./a.out studenti.txt
studenti.txt ne postoji
Izlaz:
  Greska prilikom otvaranja
  datoteke studenti.txt

```

*Test 5*

<code>Poziv: ./a.out studenti.txt</code>		
<code>studenti.txt:</code>	<code>Ulaz:</code>	<code>Izlaz:</code>
<code>prazna</code>	<code>3/2014</code>	<code>ne</code>
	<code>235/2008</code>	<code>ne</code>
	<code>41/2009</code>	<code>ne</code>

[Rešenje 4.9]

**Zadatak 4.10** Date su dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od tih lista formira novu listu L koja sadrži alternirajući raspoređene elemente lista L1 i L2 (prvi element iz L1, prvi element iz L2, drugi element L1, drugi element L2, itd). Ne formirati nove čvorove, već samo postojeće čvorove 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.

*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

```

*Test 2*

```

Poziv: ./a.out
Izlaz:
  Greska! Program se poziva sa:
  ./a.out dat1.txt dat2.txt!

```

*Test 3*

```

Poziv: ./a.out dat1.txt
Izlaz:
  Greska! Program se poziva sa:
  ./a.out dat1.txt dat2.txt!

```

*Test 4*

```

Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt ne postoji
Izlaz:
  Greska prilikom otvaranja datoteke
  dat2.txt.

```

### Test 5

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt ne postoji
dat2.txt: 2 4 6 10 15
Izlaz:
Greska prilikom otvaranja datoteke
dat1.txt.
```

### Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz:
2 4 6 10 15
```

**Zadatak 4.11** Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

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

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

### Test 1

```
Poziv: ./a.out
brojevi.txt:
43 12 15 16 4 2 8
Izlaz:
Rezultat.txt :
12 15 16
```

### Test 2

```
Poziv: ./a.out
brojevi.txt ne postoji
Izlaz:
Rezultat.txt:
Greska prilikom otvaranja datoteke
dat2.txt.
```

### Test 3

```
Poziv: ./a.out
brojevi.txt prazna
Izlaz:
Rezultat.txt je prazna.
```

### Test 4

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt:
2 4 6 10 15
Izlaz:
2 4 6 10 15
```

**Zadatak 4.12** Grupa od  $n$  plesača na kostimima imaju brojeve od 1 do  $n$ , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi  $k$ -ti plesač. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje 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 dvostruko povezanu kružnu listu.*

Milena: Dodati jos najmanje jedan test primer. Milena: Da li je ovde potrebna dvostrukopovezna lista ili moze obicna kruzna lista?

*Test 1*

```
|| Ulaz:
|| 5 3
|| Izlaz:
|| 3 1 5 2 4
```

**Zadatak 4.13** Grupa od  $n$  plesača na kostimima imaju 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 počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. *Uputstvo: Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

*Test 1*

```
|| Ulaz:
|| 5 3
|| Izlaz:
|| 3 5 4 2 1
```

*Test 2*

```
|| Ulaz:
|| 2 7
|| Izlaz:
|| n mora biti uvek vece od k, a 2 < 7!
```

## 4.2 Stabla

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

- Definisati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog stabla koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.
- Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alokira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.
- Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.

- (d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili `NULL` ukoliko takav čvor ne postoji.
- (e) Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- (f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.
- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

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



*Test 1*

```

Poziv: ./a.out
Ulaz:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
Izlaz:
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Trazi se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultuje stablo: 1 2 9 18 32

```

*Test 2*

```

Poziv: ./a.out
Ulaz:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Izlaz:
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultuje stablo: -3 -2 6 8 13 24

```

[Rešenje 4.14]

**Zadatak 4.15** Napisati program koji izračunava i na standardnom izlazu 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
Datoteka 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
Datoteka 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
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.

### *Upotreba programa 1*

```
Poziv: ./a.out
Datoteka 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
Ulaz:
  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
```

[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 (niz 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*

```

Poziv: ./a.out
Datoteka 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

```

[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 u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. 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 rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.YYYY.`

*Upotreba programa 1*

```

Poziv: a.out
Datoteka rodjendani.txt:
  Marko Markovic 12.12.1990.
  Milan Jevremovic 04.06.1989.
  Maja Agic 23.04.2000.
  Nadica Zec 01.01.1993.
  Jovana Milic 05.05.1990.
Ulaz:
  Unesite datum: 23.04.
Izlaz:
  Slavljenik: Maja Agic
Ulaz:
  Unesite datum: 01.01.
Izlaz:
  Slavljenik: Nadica Zec
Ulaz:
  Unesite datum: 01.05.
Izlaz:
  Slavljenik: Jovana Milic 05.05.
Ulaz:
  Unesite datum: CTRL+D

```

[Rešenje 4.18]

**Zadatak 4.19** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napisati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. *Napomena: Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka `stabla.h` predstavljati popis funkcija biblioteke, a datoteka `stabla.c` njihove implementacije. Programme koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

Test 1

```
Poziv: ./a.out
Ulaz:
    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
Izlaz:
    Stabla jesu identicna.
```

Test 2

```
Poziv: ./a.out
Ulaz:
    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
Izlaz:
    Stabla nisu identicna.
```

[Rešenje 4.19]

\* **Zadatak 4.20** Napisati program koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

Test 1

```
Poziv: ./a.out
Ulaz:
    Prvo stablo: 1 7 8 9 2 2
    Drugo stablo: 3 9 6 11 1
Izlaz:
    Unija: 1 1 2 2 3 6 7 8 9 9 11
    Presek: 1 9
    Razlika: 2 2 7 8
```

Test 2

```
Poziv: ./a.out
Ulaz:
    Prvo stablo: 11 2 7 5
    Drugo stablo: 4 3 3 7
Izlaz:
    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 standardnom izlazu.

*Test 1*

```
Poziv: ./a.out
Ulaz:
n: 7
a: 1 11 8 6 37 25 30
Izlaz:
1 6 8 11 25 30 37
```

*Test 2*

```
Poziv: ./a.out
Ulaz:
n: 55
Izlaz:
Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

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

- Napisati funkciju koja izračunava broj čvorova stabla.
- Napisati funkciju koja izračunava broj listova stabla.
- Napisati funkciju koja štampa pozitivne vrednosti listova stabla.
- Napisati funkciju koja izračunava zbir čvorova stabla.
- Napisati funkciju koja izračunava najveći element stabla.
- Napisati funkciju koja izračunava dubinu stabla.
- Napisati funkciju koja izračunava broj čvorova na  $i$ -tom nivou stabla.
- Napisati funkciju koja ispisuje sve elemente na  $i$ -tom nivou stabla.
- Napisati funkciju koja izračunava maksimalnu vrednost na  $i$ -tom nivou stabla.
- Napisati funkciju koja izračunava zbir čvorova na  $i$ -tom nivou stabla.
- 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.

### 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
  najveći element: 30
  dubina stabla: 5
  broj cvorova na 2. nivou: 3
  elementi na 2. nivou: 3 12 30
  maksimalni na 2. nivou: 30
  zbir na 2. nivou: 45
  zbir elemenata manjih ili jednakih od 15: 7
```

[Rešenje 4.22]

**Zadatak 4.23** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

### Test 1

```
Poziv: ./a.out
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
```

### Test 2

```
Poziv: ./a.out
Ulaz:
  6 11 8 3 -2
Izlaz:
  0.nivo: 6
  1.nivo: 3 11
  2.nivo: -2 8
```

[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.

*Test 1*

```

Poziv: ./a.out
Ulaz:
    Prvo stablo: 11 20 5 3 0
    Drugo stablo: 8 14 30 1 0
Izlaz:
    Stabla su slicna kao u ogledalu.

```

*Test 2*

```

Poziv: ./a.out
Ulaz:
    Prvo stablo: 11 20 5 3 0
    Drugo stablo: 8 20 15 0
Izlaz:
    Stabla nisu slicna kao u ogledalu.

```

**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.

*Test 1*

```

Poziv: ./a.out
Ulaz:
    10 5 15 2 11 16 1 13
Izlaz:
    7

```

*Test 2*

```

Poziv: ./a.out
Ulaz:
    16 30 40 24 10 18 45 22
Izlaz:
    6

```

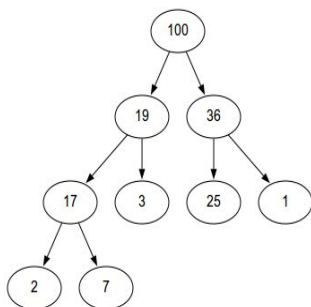
[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo se naziva HEAP ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva HEAP. Napisati zatim i glavni program koji kreira stablo kao na slici ..., poziva funkciju `heap` i ispisuje rezultat na standardnom izlazu.

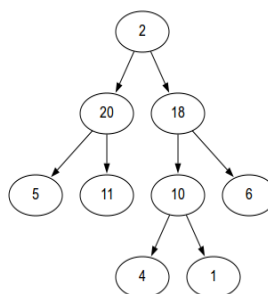
[Rešenje 4.26]

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- Napisati funkciju koja pronalazi čvor u stablu sa maksimalnim proizvodom vrednosti iz desnog podstabla.
- Napisati funkciju koja pronalazi čvor u stablu sa najmanjom sumom vrednosti iz levog podstabla.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.



Slika 4.1: zadatak 4.27



Slika 4.2: zadatak 4.23

- (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 ...

## 4.3 Rešenja

### Rešenje 4.1

```
1  #ifndef _LISTA_H
2  #define _LISTA_H
3
4  /* Struktura kojom je predstavljen cvor liste */
5  typedef struct cvor {
6      /* Podatak koji cvor sadrzi */
7      int vrednost;
8      /* Pokazivac na sledeci cvor liste */
9      struct cvor *sledeci;
10 } Cvor;
11
12 Cvor* napravi_cvor(int broj);
13
14 void oslobodi_listu(Cvor** adresa_glave) ;
15
16 void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) ;
17
18 void dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj);
19
20 Cvor * pronadji_poslednji (Cvor * glava);
21
```



```

23 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
Cvor * pronadji_mesto_umetanja(Cvor * glava, int broj ) ;
25 void dodaj_iza(Cvor * tekuci, Cvor * novi) ;
27 void dodaj_sortirano(Cvor ** adresa_glave, int broj) ;
29 Cvor * pretrazi_listu(Cvor * glava, int broj) ;
31 Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj);
33 void obrisi_element(Cvor ** adresa_glave, int broj);
35 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
37 broj);
39 void ispisi_listu(Cvor * glava);
41 #endif

```

```

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

5  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
   novog cvora inicijalizuje na broj, dok pokazivac na
7  sledeci cvor u novom cvoru postavlja na NULL.
   Funkcija vraca pokazivac na novokreirani cvor ili NULL
9  ako alokacija nije uspesno izvsena. */
Cvor * napravi_cvor(int broj)
11 {
   Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
13   if( novi == NULL )
       return NULL;

15   /* Inicijalizacija polja u novom cvoru */
17   novi->vrednost = broj;
   novi->sledeci = NULL;
19   return novi;
}

21 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
23 liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave)
25 {
   Cvor *pomocni = NULL;

27   /* Ako lista nije prazna, onda treba osloboditi memoriju. */
29   while (*adresa_glave != NULL)
       {
31       /* Potrebno je prvo zapamtiti adresu sledeceg elementa i

```

```

        onda osloboditi element koji predstavlja glavu liste */
33     pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
35     /* Sledeci element je nova glava liste */
        *adresa_glave = pomocni;
37     }
}
39
/* Funkcija proverava uspesnost alokacije memorije za cvor novi
41 i ukoliko alokacija nije bila uspesna, oslobadja se sva
42 prethodno zauzeta memorija za listu ciji pocetni cvor se
43 nalazi na adresi adresa_glave.*/
void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
45 {
    /* Ukoliko je novi NULL */
47     if ( novi == NULL )
    {
49         fprintf(stderr, "Neuspela alokacija za nov cvor\n");

        /* Oslobadjamo dinamicki alociranu memoriju i
51         prekidamo program */
53         oslobodi_listu(adresa_glave);
        exit(EXIT_FAILURE);
55     }
}

57
/* Funkcija dodaje novi cvor na pocetak liste.
59 Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje
ga na pocetak */
61 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
63     /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
alokaciji */
65     Cvor *novi = napravi_cvor(broj);
    prover_i_alokaciju(adresa_glave, novi);

67
    /* Uvezujemo novi cvor na pocetak */
69     novi->sledeci = *adresa_glave;
    /* Nov cvor je sada nova glava liste */
71     *adresa_glave = novi;
}

73
75
/* Funkcija pronalazi i vraca pokazivac na poslednji element
77 liste, ili NULL ukoliko je lista prazna. */
Cvor * pronadji_poslednji (Cvor * glava)
79 {
    /* Prazna lista nema ni poslednji cvor i u tom slucaju
81     vratamo NULL.*/
    if( glava == NULL)
83         return NULL;

```

```
85  /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
86     pomeramo pokazivac glava na taj sledeci element. Kada
87     izadjemo iz petlje, glava ce pokazivati na element liste
88     koji nema sledeceg, tj, poslednji element liste je. Zato
89     vracamo vrednost pokazivaca glava.
90     Pokazivac glava je argument funkcije i njegove promene nece
91     se odraziti na vrednost pokazivaca glava u pozivajucoj
92     funkciji. */
93  while (glava->sledeci != NULL)
94      glava = glava->sledeci;
95
96  return glava;
97  }
98
99  /* Funkcija dodaje novi cvor na kraj liste. */
100 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
101 {
102     Cvor *novi = napravi_cvor(broj);
103     prover_i_alokaciju(adresa_glave, novi);
104
105     /* U slucaju prazne liste */
106     if (*adresa_glave == NULL)
107     {
108         /* Glava nove liste je upravo novi cvor i ujedno i cela
109            lista. Azuriramo vrednost na koju pokazuje adresa_glave i
110            tako azuriramo i pokazivacku promenljivu u pozivajucoj
111            funkciji. */
112         *adresa_glave = novi;
113         return;
114     }
115
116     /* Ako lista nije prazna, pronalazimo poslednji element */
117     Cvor * poslednji = pronadji_poslednji(*adresa_glave);
118
119     /* Dodajemo novi cvor na kraj preusmeravanjem pokazivaca */
120     poslednji->sledeci = novi;
121 }
122
123 /* Pomocna funkcija pronalazi cvor u listi iza koga treba
124    umetnuti nov element sa vrednoscu broj.*/
125 Cvor * pronadji_mesto_umetanja(Cvor * glava, int broj )
126 {
127     /*Ako je lista prazna onda nema takvog mesta i vracamo NULL */
128     if(glava == NULL)
129         return NULL;
130
131     /* Krecemo se kroz listu sve dok se ne dodje do elementa ciji
132        je sledeci element veci ili jednak od novog elementa, ili
133        dok se ne dodje do poslednjeg elementa.
134
135
```

```
137     Zbog lenjog izracunavanja izraza u C-u prvi deo konjukcije
139     mora biti provera da li smo dosli do poslednjeg elementa
        liste pre nego sto proverimo vrednost njegovog sledeceg
141     elementa, jer u slucaju poslednjeg, sledeci ne postoji,
        pa ni vrednost.*/
143     while (glava->sledeci != NULL
            && glava->sledeci->vrednost < broj)
145         glava = glava->sledeci;

147     /* Iz petlje smo mogli izaci jer smo dosli do poslednjeg
        elementa ili smo se zaustavili ranije na elementu ciji
        sledeci ima vrednost vecu od broj */
149     return glava;
}

151 void dodaj_iza(Cvor * tekuci, Cvor * novi)
153 {
    /* Novi element dodajemo iza tekuceg elementa */
155     novi->sledeci = tekuci->sledeci;
    tekuci->sledeci = novi;
157 }

159 /* Funkcija dodaje novi element u sortiranu listu
    tako da nova lista ostane sortirana.*/
161 void dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
163     /* U slucaju prazne liste glava nove liste je novi cvor */
    if ( *adresa_glave == NULL ) {
165         Cvor *novi = napravi_cvor(broj);
        /* Proveravamo da li je doslo do greske prilikom
167         alokacije memorije */
        prover_i_alokaciju(adresa_glave, novi);
169         *adresa_glave = novi;
        return;
171     }

173     /* Lista nije prazna*/
    /* Ako je broj manji ili jednak vrednosti u glavi liste,
175     onda ga dodajemo na pocetak liste */
    if ( (*adresa_glave)->vrednost >= broj ) {
177         dodaj_na_pocetak_liste(adresa_glave, broj);
        return;
179     }

181     /* U slucaju da je glava liste element manji od novog elementa,
        tada pronalazimo element liste iza koga treba da se umetne
183     nov broj */
    Cvor * pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);

185     Cvor *novi = napravi_cvor(broj);
187     prover_i_alokaciju(adresa_glave, novi);
```

```

189  /* Uvezujemo novi cvor iza pomocnog */
    dodaj_iza(pomocni, novi);
191  }

193  /* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
195  broj ili NULL u slucaju da takav element ne postoji u listi.*/
    Cvor * pretrazi_listu(Cvor * glava, int broj)
197  {
        for ( ; glava != NULL; glava = glava->sledeci)
199            if (glava->vrednost == broj)
                return glava;

201        /* Nema trazenog broja u listi i vracamo NULL*/
203        return NULL;
    }
205

207  /* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
209  broj ili NULL u slucaju da takav element ne postoji u listi.
    Funkcija se u pretrazi oslanja na cinjenicu da je lista koja
211  se pretrazuje neopadajuće sortirana. */
    Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj)
213  {
        /* U konjukciji koja cini uslov ostanka u petlji,
215        bitan je redosled! */
        for ( ; glava != NULL && glava->vrednost <= broj ;
217            glava = glava->sledeci)
            if (glava->vrednost == broj)
219                return glava;

221        /* Nema trazenog broja u listi i vracamo NULL*/
        return NULL;
223    }

225
227  /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
    Funkcija azurira pokazivac na glavu liste, koji moze biti
    promenjen u slucaju da se obrise stara glava. */
229  void obrisi_element(Cvor ** adresa_glave, int broj)
    {
231        Cvor *tekuci = NULL;
        Cvor *pomocni = NULL;

233
        /* Brisemo sa pocetka liste sve cvorove koji su jednaki datom
235        broju, i azuriramo pokazivac na glavu */
        while (*adresa_glave != NULL
237            && (*adresa_glave)->vrednost == broj)
        {
239            /* Sacuvamo adresu repa liste pre oslobadjanja glave */

```

```

241     pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        *adresa_glave = pomocni;
243     }

245     /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
    if ( *adresa_glave == NULL)
247         return;

249     /* Od ovog trenutka se u svakom koraku nalazimo
        na tekucem cvoru koji je razlicit od trazenog
251     broja (kao i svi levo od njega). Poredimo
        vrednost sledeceg cvora (ako postoji) sa trazanim
253     brojem i brisemo ga ako je jednak, a prelazimo na
        sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
255     dok ne dodjemo do poslednjeg cvora. */
    tekuci = *adresa_glave;
257     while (tekuci->sledeci != NULL)
        if (tekuci->sledeci->vrednost == broj)
259            {
                /* tekuci->sledeci treba obrisati,
                zbog toga sacuvamo njegovu adresu u pomocni */
261                pomocni = tekuci->sledeci;
                /* Tekucem preusmerimo pokazivac sledeci
                tako sto preskakemo njegovog trenutnog sledeceg.
263                Njegov novi sledeci ce biti sledeci od ovog koga
                brisemo. */
                tekuci->sledeci = pomocni->sledeci;
                /* Sada mozemo da oslobodimo cvor sa vrednoscu broj. */
265                free(pomocni);
            }
267        else
            {
271                /* Ne treba brisati sledeceg. Prelazimo na sledeci. */
                tekuci = tekuci->sledeci;
            }
273        return;
275    }
277 }

279 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
281    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
    neopadajuće. Funkcija azurira pokazivac na glavu liste, koji
283    moze biti promenjen ukoliko se obrise stara glava liste. */
void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
285 broj)
{
287     Cvor *tekuci = NULL ;
    Cvor *pomocni = NULL ;

289     /* Brisemo sa pocetka liste sve eventualne cvorove koji su
291     jednaki datom broju, i azuriramo pokazivac na glavu */

```

```

293 while (*adresa_glave != NULL
      && (*adresa_glave)->vrednost == broj)
    {
295     /* Sacuvamo adresu repa liste pre oslobadjanja glave */
    pomocni = (*adresa_glave)->sledeci;
297     free(*adresa_glave);
    *adresa_glave = pomocni;
299     }

301 /* Ako je nakon toga lista ostala prazna ili glava liste sadrzi
302    vrednost koja je veca od broja, kako je lista sortirana
303    rastuce nema potrebe broj traziti u repu liste i zato
    prekidamo funkciju */
305 if ( *adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
    return;
307

309 /* Od ovog trenutka se u svakom koraku nalazimo u tekucem cvoru
    cija vrednost je manja od trazenog broja kao i svi levo od
    njega. Poredimo vrednost sledeceg cvora, ako postoji, sa
311    trazenim brojem i brisemo ga ako je jednak, a prelazimo na
    sledeci cvor ako je razlicit. Ovaj postupak ponavljamo dok
313    ne dodjemo do poslednjeg cvora ili prvog cvora cija vrednost
    je veca od trazenog broja. */
315 tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL
317           && tekuci->sledeci->vrednost <= broj)
        if (tekuci->sledeci->vrednost == broj)
319            {
                /* tekuci->sledeci treba obrisati, zbog toga cuvamo
321                   njegovu adresu u pomocni */
                pomocni = tekuci->sledeci;
323                /* Tekucem preusmerimo pokazivac sledeci tako sto
                   preskakemo njegovog trenutnog sledeceg. Njegov novi
325                   sledeci ce biti sledeci od ovog koga brisemo. */
                tekuci->sledeci = tekuci->sledeci->sledeci;
327                /* Sada mozemo da oslobodimo cvor sa vrednoscu broj */
                free(pomocni);
329            }
        else
331            {
                /* Ne treba brisati sledeceg, jer je manji od trazenog i
333                   prelazimo na sledeci */
                tekuci = tekuci->sledeci;
335            }
    return;
337 }

339

341 /* Funkcija prikazuje elemente liste pocev od glave ka kraju
    liste. Ne saljemo joj adresu promenljive koja cuva glavu
    liste, jer ova funkcija nece menjati listu, pa nema ni
343    potrebe da azuriza pokazivac na glavu liste iz pozivajuce

```

```
    funkcije. */
345 void ispisi_listu(Cvor * glava)
{
347     putchar('[');
    for ( ; glava != NULL; glava = glava->sledeci)
349     {
        printf("%d", glava->vrednost);
351         if( glava->sledeci != NULL )
            printf(", ");
353     }

355     printf("]\n");
}
```

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

4
int main()
6 {
    /* Lista je na pocetku prazna. */
8     Cvor * glava = NULL;
    Cvor * trazeni = NULL;
10     int broj;

12     /* Testiramo dodavanje na pocetak*/
    printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
14     printf("\n\tLista: ");
    ispisi_listu(glava);

16     while(scanf("%d",&broj)>0)
18     {
        dodaj_na_pocetak_liste(&glava, broj);
20         printf("\n\tLista: ");
        ispisi_listu(glava);
22     }

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

26     trazeni = pretrazi_listu(glava, broj);
28     if(trazeni == NULL)
        printf("Element NIJE u listi!\n");
30     else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
32     oslobodi_listu(&glava);

34     return 0;
36 }
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 int main()
6 {
7     Cvor * glava = NULL;
8     int broj;
9
10    /* Testiramo dodavanje na kraj liste */
11    printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
12    printf("\n\tLista: ");
13    ispisi_listu(glava);
14
15    while(scanf("%d",&broj) > 0)
16    {
17        dodaj_na_kraj_liste(&glava, broj);
18        printf("\n\tLista: ");
19        ispisi_listu(glava);
20    }
21
22    printf("\nUnesite element koji se brise iz liste: ");
23    scanf("%d", &broj);
24
25    /* Brisemo elemente iz liste cije polje vrednost je jednako
26       broju procitanom sa ulaza */
27    obrisi_element(&glava, broj);
28
29    printf("Lista nakon brisanja: ");
30    ispisi_listu(glava);
31
32    oslobodi_listu(&glava);
33
34    return 0;
35 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 int main() {
6     Cvor * glava = NULL;
7     Cvor * trazeni = NULL;
8     int broj;
9
10    /* Testiramo dodavanje u listu tako da ona bude neopadajuće
11       uredjena */
12    printf("Unosite elemente liste (za kraj unesite CTRL+D)\n");
13    printf("\n\tLista: ");
14    ispisi_listu(glava);
15 }
```

```
17 while(scanf("%d",&broj)>0)
    {
19     dodaj_sortirano(&glava, broj);
    printf("\n\tLista: ");
21     ispisi_listu(glava);
    }

23
    printf("\nUnesite element koji se trazi u listi: ");
25     scanf("%d", &broj);

27     trazeni = pretrazi_sortiranu_listu(glava, broj);
    if(trazeni == NULL)
29         printf("Element NIJE u listi!\n");
    else
31         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

33
    /* Brisemo elemente iz liste cije polje vrednost je jednako
35     broju procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
37     scanf("%d", &broj);

39     obrisi_element_sortirane_liste(&glava, broj);

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

43
    oslobodi_listu(&glava);

45     return 0;
47 }
```

### Rešenje 4.2

```
1  #ifndef _LISTA_H
   #define _LISTA_H
3
   /* Biblioteka rekurzivnih funkcije za rad
5   sa jednostruko povezanom listom celih brojeva */

7   /* Struktura kojom je predstavljen cvor liste */
   typedef struct cvor {
9       /* Podatak koji cvor sadrzi */
       int vrednost;
11      /* Pokazivac na sledeci cvor liste */
       struct cvor *sledeci;
13  } Cvor;

15  Cvor* napravi_cvor(int broj);
```

```

17 void oslobodi_listu(Cvor** adresa_glave) ;
19 int dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj);
21 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
23 int dodaj_sortirano(Cvor ** adresa_glave, int broj) ;
25 Cvor * pretrazi_listu(Cvor * glava, int broj) ;
27 Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj);
29 void obrisi_element(Cvor ** adresa_glave, int broj);
31 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
    broj);
33 void ispisi_listu(Cvor * glava);
35 void ispisi_elemente(Cvor * glava);
37
    #endif

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

/* Pomocna funkcija koja kreira cvor. Funkcija vrednost
    novog cvora inicijalizuje na broj, dok pokazivac na
    sledeci cvor u novom cvoru postavlja na NULL.
    Funkcija vraca pokazivac na novokreirani cvor ili NULL
    ako alokacija nije uspesno izvršena. */
Cvor * napravi_cvor(int broj)
{
    Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
    if( novi == NULL )
        return NULL;

    /* Inicijalizacija polja u novom cvoru */
    novi->vrednost = broj;
    novi->sledeci = NULL;
    return novi;
}

/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
    liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave)
{
    /* Lista je vec prazna */
    if( *adresa_glave == NULL )
        return;

```

```
30  /* Ako lista nije prazna, treba osloboditi memoriju. Pre
    nego oslobodimo memoriju za glavu liste, moramo osloboditi
32     rep liste. */
    oslobodi_listu( &(*adresa_glave)->sledeci);
34  /* Nakon oslobodjenog repa, oslobadjamo i glavu liste */
    free(*adresa_glave);
36  /* Azuriramo glavu u pozivajucoj funkciji tako da odgovara
    praznoj listi */
38  *adresa_glave = NULL;
}

40

42  /* Funkcija dodaje novi cvor na pocetak liste.
    Kreira novi cvor koriscenjem funkcije napravi_cvor() i
44     uvezuje ga na pocetak. Funkcija vraca 1 ukoliko je doslo do
    greske pri alokaciji, inace vraca 0. */
46  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
    {
48      /* Kreiramo nov cvor i proverimo da li je bilo greske pri
        alokaciji */
50      Cvor *novi = napravi_cvor(broj);
        if( novi == NULL)
52          return 1;

54      /* Uvezujemo novi cvor na pocetak */
        novi->sledeci = *adresa_glave;
56      /* Nov cvor je sada nova glava liste */
        *adresa_glave = novi;
58      return 0;
    }

60

62  /* Funkcija dodaje novi cvor na kraj liste.
    Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
    nov broj se dodaje u rep liste u rekurzivnom pozivu. U slucaju
64     da je u rekurzivnom pozivu doslo do greske pri alokaciji,
    funkcija vraca 1 visem rekurzivnom pozivu koji tu informaciju
66     vraca u rekurzivni poziv iznad, sve dok se ne vrati u main.
    Ako je funkcija vratila 0, onda nije bilo greske. Tek je iz
68     main funkcije moguće pristupiti pravom pocetku liste i
    osloboditi je celu, ako ima potrebe. */
70  int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
    {
72      if (*adresa_glave == NULL)
        {
74          /* Glava liste je upravo novi cvor i ujedno i cela lista.*/
            Cvor *novi = napravi_cvor(broj);
76            if( novi == NULL)
                return 1;

78            /* Azuriramo vrednost na koju pokazuje adresa_glave i tako
            azuriramo i pokazivacku promenljivu u pozivajucoj
80            funkciji. */
```

```

82     *adresa_glave = novi;
      return 0;
84 }

86 /* Ako lista nije prazna, nov element se dodaje u rep liste. */
      return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
88 }

90 /* Funkcija dodaje novi element u rastuce sortiranu listu tako da
91    nova lista ostane sortirana. Vraca 0, ako je alokacija novog
92    cvora prosla bez greske, inace vraca 1 da bi ta vrednost bila
93    propagirala nazad do prvog poziva. */
94 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
95 {
96     /* u slucaju prazne liste adresa_glave nove liste je upravo
97        novi cvor */
98     if (*adresa_glave == NULL)
99     {
100         Cvor *novi = napravi_cvor(broj);
101         if( novi == NULL)
102             return 1;
103
104         *adresa_glave = novi;
105         return 0 ;
106     }

108     /* Lista nije prazna*/
109     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda
110        u sustini dodajemo na pocetak liste. */
111     if ((*adresa_glave)->vrednost >= broj )
112     {
113         /* Vracamo informaciju o uspesnosti alokacije */
114         return dodaj_na_pocetak_liste(adresa_glave, broj);
115     }

116     /* Inace, element treba dodati u rep, tako da rep i sa novim
117        elementom bude sortirana lista */
118     return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
120 }

122

124 /* Funkcija trazi u listi element cija je vrednost jednaka datom
125    broju. Vraca pokazivac na cvor liste u kome je sadržan traženi
126    broj ili NULL u slucaju da takav element ne postoji u listi.*/
127 Cvor * pretrazi_listu(Cvor * glava, int broj)
128 {
129     /* U praznoj listi ga sigurno nema */
130     if(glava == NULL )
131         return NULL;

132     /* Ako glava liste sadrži traženi broj */

```

```
134     if(glava->vrednost == broj )
135         return glava;
136
137     /* Ako nije nijedna od prethodnih situacija, pretragu
138        nastavljamo u repu */
139     return pretrazi_listu(glava->sledeci, broj);
140 }
141
142
143
144 /* Funkcija trazi u listi element cija je vrednost jednaka datom
145    broju. Funkcija se u pretrazi oslanja na cinjenicu da je lista
146    koja se pretrazuje neopadajuće sortirana. Vraca pokazivac na
147    cvor liste u kome je sadržan traženi broj ili NULL u slučaju
148    da takav element ne postoji u listi. */
149 Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj)
150 {
151     /* U praznoj listi ga sigurno nema */
152     if(glava == NULL || glava->vrednost > broj)
153         return NULL;
154
155     /* Ako glava liste sadrži traženi broj */
156     if(glava->vrednost == broj )
157         return glava;
158
159     /* Ako nije nijedna od prethodnih situacija, pretragu
160        nastavljamo u repu */
161     return pretrazi_listu(glava->sledeci, broj);
162 }
163
164
165 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
166    Funkcija azurira pokazivac na glavu liste, koji može biti
167    promenjen u slučaju da se obriše stara glava liste. */
168 void obrisi_element(Cvor ** adresa_glave, int broj)
169 {
170     /* Ako je lista prazna nema šta da se brise, vracamo se iz
171        funkcije. */
172     if( *adresa_glave == NULL)
173         return ;
174
175     /* Pre nego proverimo situaciju sa glavom liste, obrisacemo sve
176        cvorove iz repa koji imaju vrednost bas broj */
177     obrisi_element(&(*adresa_glave)->sledeci, broj);
178
179     /* Preostaje da proverimo da li glavu treba obrisati. */
180     if ( (*adresa_glave)->vrednost == broj )
181     {
182         /* Cvor koji treba da se obriše */
183         Cvor* pomocni = *adresa_glave;
184         /* Azuriramo pokazivac na glavu da pokazuje na sledeci u
185            listi i brisemo element koji je bio glava liste. */
186     }
```

```

186     *adresa_glave = (*adresa_glave)->sledeci;
187     free(pomocni);
188 }
189
190
191
192 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
193    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
194    neopadajuće. Funkcija azurira pokazivac na glavu liste, koji
195    može biti promenjen ukoliko se obrise stara glava liste. */
196 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
197 broj)
198 {
199     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
200        veca od broja, kako je lista sortirana rastuce nema potrebe
201        broj traziti u repu liste i zato prekidamo funkciju */
202     if ( *adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
203         return;
204
205     /* Pre nego proverimo situaciju sa glavom liste, obrisacemo
206        sve cvorove iz repa koji imaju vrednost bas broj */
207     obrisi_element(&(*adresa_glave)->sledeci, broj);
208
209     /* Preostaje da proverimo da li glavu treba obrisati. */
210     if ( (*adresa_glave)->vrednost == broj )
211     {
212         /* Cvor koji treba da se obrise */
213         Cvor* pomocni = *adresa_glave;
214         /* Azuriramo pokazivac na glavu da pokazuje na sledeci u
215            listi i brisemo element koji je bio glava liste. */
216         *adresa_glave = (*adresa_glave)->sledeci;
217         free(pomocni);
218     }
219 }
220
221 /* Funkcija ispisuje samo elemente liste razdvojene zapetama */
222 void ispisi_elemente(Cvor * glava)
223 {
224     /* Prazna lista*/
225     if(glava == NULL)
226         return;
227
228     /* Ispisujemo element u glavi liste*/
229     printf(" %d",glava->vrednost);
230     /* Rekurzivni poziv za ispis svega ostalo */
231     ispisi_elemente(glava->sledeci);
232 }
233
234 /* Funkcija prikazuje elemente liste pocev od glave ka kraju
235    liste. Ne saljemo joj adresu promenljive koja cuva glavu
236    liste, jer ova funkcija nece menjati listu, pa nema ni potrebe
237    da azurira pokazivac iz pozivajuće funkcije. */

```

## 4 Dinamičke strukture podataka

---

```
238 void ispisi_listu(Cvor * glava)
{
240     putchar('[');
        ispisi_elemente(glava);
242     putchar(']');

244     putchar('\n');
}

1  /* Rekurzivne funkcije za rad sa jednostruko povezanom listom */
#include <stdio.h>
3  #include <stdlib.h>
#include "lista.h"

5
int main()
{
7     /* Lista je prazna na pocetku. */
    Cvor *glava = NULL;
9     Cvor *trazeni = NULL;
    int broj;

11

13     /* Testiramo dodavanje na pocetak */
    printf("Unosite elemente liste! (za kraj unesite CTRL+D)\n");
15     while (scanf("%d", &broj) > 0)
    {
17         /* Ako je funkcija vratila 1 onda je bilo greske pri
            alokaciji memorije za nov cvor. Listu moramo osloboditi
19         pre napustanja programa. */
        if ( dodaj_na_pocetak_liste(&glava, broj) == 1)
21         {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23             oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
25         }
        printf("\n\tLista: ");
27         ispisi_listu(glava);
    }

29     ispisi_listu(glava);

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

35     trazeni=pretrazi_listu(glava, broj);
    if( trazeni == NULL)
37         printf("Element NIJE u listi!\n");
    else
39         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

41     oslobodi_listu(&glava);

43     return 0;
```



```
}

```

```

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

int main()
{
    Cvor * glava = NULL;
    int broj;

    /* Testiramo dodavanje na kraj liste */
    printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
    printf("\n\tLista: ");
    ispisi_listu(glava);

    while(scanf("%d",&broj) > 0)
    {
        /* Ako je funkcija vratila 1 onda je bilo greske pri
           alokaciji memorije za nov cvor. Listu moramo osloboditi
           pre napustanja programa. */
        if ( dodaj_na_kraj_liste(&glava, broj) == 1)
        {
            fprintf(stderr,"Neuspela alokacija za cvor %d\n",broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
        }
        printf("\n\tLista: ");
        ispisi_listu(glava);
    }

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

    /* Brisemo elemente iz liste cije polje vrednost je jednako
       broju procitanom sa ulaza */
    obrisi_element(&glava, broj);

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

    oslobodi_listu(&glava);

    return 0;
}

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5

```

```
int main() {
7   Cvor * glava = NULL;
   Cvor * trazeni = NULL;
9   int broj;

11  /* Testiramo dodavanje u listu tako da ona bude neopadajuće
   uredjena */
13  printf("Unosite elemente liste (za kraj unesite CTRL+D)\n");
   printf("\n\tLista: ");
15  ispisi_listu(glava);

17  while(scanf("%d",&broj)>0)
   {
19      /* Ako je funkcija vratila 1 onda je bilo greske pri
       alokaciji memorije za nov cvor. Listu moramo osloboditi
       pre napustanja programa. */
21      if ( dodaj_sortirano(&glava, broj) == 1)
       {
23          fprintf(stderr,"Neuspela alokacija za cvor %d\n",broj);
           oslobodi_listu(&glava);
           exit(EXIT_FAILURE);
25          }
       printf("\n\tLista: ");
27          ispisi_listu(glava);
29      }

31  printf("\nUnosite element koji se trazi u listi: ");
33  scanf("%d", &broj);

35  trazeni = pretrazi_sortiranu_listu(glava, broj);
   if(trazeni == NULL)
37      printf("Element NIJE u listi!\n");
   else
39      printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

41  /* Brisemo elemente iz liste cije polje vrednost je jednako
   broju procitanom sa ulaza */
43  printf("\nUnosite element koji se brise iz liste: ");
45  scanf("%d", &broj);

47  obrisi_element_sortirane_liste(&glava, broj);

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

51  oslobodi_listu(&glava);

53  return 0;
55 }
```

## Rešenje 4.3

```
1  #ifndef _LISTA_H
2  #define _LISTA_H
3
4  /* Struktura kojom je predstavljen cvor liste */
5  typedef struct cvor{
6      int vrednost;
7      struct cvor *sledeci;
8      struct cvor *prethodni;
9  } Cvor;
10
11 Cvor* napravi_cvor(int broj);
12
13 void oslobodi_listu(Cvor** adresa_glave) ;
14
15 void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) ;
16
17 void dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj);
18
19 Cvor * pronadji_poslednji (Cvor * glava);
20
21 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
22
23 Cvor * pronadji_mesto_umetanja(Cvor * glava, int broj ) ;
24
25 void dodaj_iza(Cvor * tekuci, Cvor * novi) ;
26
27 void dodaj_sortirano(Cvor ** adresa_glave, int broj) ;
28
29 Cvor * pretrazi_listu(Cvor * glava, int broj) ;
30
31 Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj);
32
33 void obrisi_tekuci(Cvor** adresa_glave, Cvor* tekuci);
34
35 void obrisi_element(Cvor ** adresa_glave, int broj);
36
37 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
    broj);
38
39 void ispisi_listu(Cvor * glava);
40
41 void ispisi_listu_u_nazad(Cvor* glava) ;
42
43 #endif
44
45 #include <stdio.h>
46 #include <stdlib.h>
47 #include "lista.h"
```

```

6  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
   cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
   novom cvoru postavlja na NULL. Funkcija vraća pokazivac na
   novokreirani cvor ili NULL ako alokacija nije uspesno
   izvršena. */
10 Cvor *napravi_cvor(int broj)
   {
12     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
14         return NULL;

16     /* Inicijalizacija polja u novom cvoru. */
       novi->vrednost = broj;
18     novi->sledeci = NULL;
       return novi;
20 }

22 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
   liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
24 void oslobodi_listu(Cvor ** adresa_glave)
   {
26     Cvor *pomocni = NULL;

28     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
       while (*adresa_glave != NULL) {
30         /* Potrebno je prvo zapamtiti adresu sledeceg elementa i
           onda osloboditi element koji predstavlja glavu liste*/
32         pomocni = (*adresa_glave)->sledeci;
           free(*adresa_glave);
34         /* Sledeci element je nova glava liste */
           *adresa_glave = pomocni;
36     }
   }

38

40 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
   i ukoliko alokacija nije bila uspesna, oslobadja se sva
   prethodno zauzeta memorija za listu ciji pocetni cvor se
   nalazi na adresi adresa_glave. */
42 void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
   {
44     if (novi == NULL) {
46         fprintf(stderr, "Neuspela alokacija za nov cvor\n");

48         /* Oslobadjamo dinamicki alociranu memoriju i prekidamo
           program */
50         oslobodi_listu(adresa_glave);
           exit(EXIT_FAILURE);
52     }
   }

54

56 /* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor

```

```

koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak.*/
58 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
60     Cvor *novi = napravi_cvor(broj);
    prover_i_alokaciju(adresa_glave, novi);

62     /* Sledbenik novog cvora je glava stare liste */
64     novi->sledeci = *adresa_glave;
    /* Ako stara lista nije bila prazna, onda prethodni od glave
66     treba da bude nov cvor. */
    if (*adresa_glave != NULL)
68         (*adresa_glave)->prethodni = novi;
    /* azuriramo pokazivac na glavu u pozivajucoj funkciji jer
70     je novi od sada glava liste */
    *adresa_glave = novi;
72 }

74
/* Funkcija pronalazi i vraca pokazivac na poslednji element
76 liste, ili NULL kao je lista prazna */
Cvor *pronadji_poslednji(Cvor * glava)
78 {
    /*
80     * ako je lista prazna, nema ni poslednjeg cvor i u tom
    * slucaju vracamo NULL.
82     */
    if (glava == NULL)
84         return NULL;

86     /*
88     * Sve dok glava ne pokazuje na cvor koji nema sledeceg,
    * pomeramo pokazivac glava na taj sledeci element. Kada
    * izađemo iz petlje, glava ce pokazivati na element liste
90     * koji nema sledeceg, tj, poslednji element liste je. Zato
    * vracamo vrednost pokazivaca glava. glava je argument
92     * funkcije i njegove promene nece se odraziti na vrednost
    * pokazivaca glava u pozivajucoj funkciji.
94     */
    while (glava->sledeci != NULL)
96         glava = glava->sledeci;

98     return glava;
}

100
102 /*
    * Funkcija nov cvor dodaje na kraj liste.
104 */
void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
106 {
    Cvor *novi = napravi_cvor(broj);
108     /*

```

```
110     * Proveravamo da li je doslo do greske prilikom alokacije
    * memorije
    */
112     prover_i_alokaciju(adresa_glave, novi);

114     /*
    * ako je lista u koju dodajemo prazna. Nov cvor je jedini
116     * cvor u novoj listi i time je i glava nove liste.
    */
118     if (*adresa_glave == NULL) {
        *adresa_glave = novi;
120         return;
    }

122     /*
    * Ako lista nije prazna, pronalazimo poslednji element
124     * liste
    */
126     Cvor *poslednji = pronadji_poslednji(*adresa_glave);

128     /*
    * tada uvezujemo nov cvor na kraj, tako sto mu azuriramo
130     * pokazivac na prethodni da pokazuje na poslednjeg.
    * Sledeci od poslednjeg treba da bude nov cvor.
    */
132     poslednji->sledeci = novi;
    novi->prethodni = poslednji;
136 }

138

140
142 /*
    * Pomocna funkcija pronalazi cvor u listi iza koga treba
    * umetnuti nov element sa vrednoscu broj.
144 */
    Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
146 {
    /*
    * Ako je lista prazna onda nema takvog mesta i vracamo NULL
    */
148     if (glava == NULL)
        return NULL;

150     /*
    * Krecemo se kroz listu sve dok se ne dodje do elementa
    * ciji je sledeci element veci ili jednak od novog
152     * elementa, ili dok se ne dodje do poslednjeg elementa.
    *
    * Zbog lenjog izracunavanja izraza u C-u prvi deo
154     * konjukcije mora biti prover a da li smo dosli do
    * poslednjeg elementa liste pre nego sto proverimo vrednost
160
```

```

162     * njegovog sledeceg elementa, jer u slucaju poslednjeg,
163     * sledeci ne postoji, pa ni vrednost.
164     */
165     while (glava->sledeci != NULL
166            && glava->sledeci->vrednost < broj)
167         glava = glava->sledeci;
168
169     /*
170     * Iz petlje smo mogli izaci jer smo dosli do poslednjeg
171     * elementa ili smo se zaustavili ranije na elementu ciji
172     * sledeci ima vrednost vecu od broj
173     */
174     return glava;
175 }
176
177 void dodaj_iza(Cvor * tekuci, Cvor * novi)
178 {
179     /*
180     * Novi element dodajemo iza tekuceg elementa
181     */
182     novi->sledeci = tekuci->sledeci;
183     novi->prethodni = tekuci;
184
185     /*
186     * Ako tekuci ima sledeceg, onda upravo dodajemo njemu
187     * prethodnika i potrebno je i njemu da postavimo pokazivace
188     * na ispravne adrese
189     */
190     if (tekuci->sledeci != NULL)
191         tekuci->sledeci->prethodni = novi;
192     tekuci->sledeci = novi;
193 }
194
195
196
197 /*
198 * Funkcija dodaje u listu nov cvor na odgovarajuce mesto, tako
199 * sto pronalazi cvor u listi iza kod treba uvezati nov cvor.
200 */
201 void dodaj_sortirano(Cvor ** adresa_glave, int broj)
202 {
203     /*
204     * Ako je lista prazna, glava nove liste je novi cvor
205     */
206     if (*adresa_glave == NULL) {
207         Cvor *novi = napravi_cvor(broj);
208         prover_i_alokaciju(adresa_glave, novi);
209         *adresa_glave = novi;
210         return;
211     }
212 }

```

```
214     /*
215      * Lista nije prazna
216      */
217     /*
218      * Ukoliko je vrednost glave liste veca od nove vrednosti
219      * onda nov cvor treba staviti na pocetak liste.
220      */
221     if ((*adresa_glave)->vrednost >= broj) {
222         dodaj_na_pocetak_liste(adresa_glave, broj);
223         return;
224     }
225
226     Cvor *novi = napravi_cvor(broj);
227     prover_i_alokaciju(adresa_glave, novi);
228
229     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
230     /*
231      * Uvezujemo novi cvor iza pomocnog
232      */
233     dodaj_iza(pomocni, novi);
234 }
235
236 /*
237 * Funkcija trazi u listi element cija je vrednost jednaka datom
238 * broju. Vraca pokazivac na cvor liste u kome je sadržan
239 * traženi broj ili NULL u slučaju da takav element ne postoji u
240 * listi.
241 */
242 Cvor *pretrazi_listu(Cvor * glava, int broj)
243 {
244     for (; glava != NULL; glava = glava->sledeci)
245         if (glava->vrednost == broj)
246             return glava;
247
248     /*
249      * Nema traženog broja u listi i vraćamo NULL
250      */
251     return NULL;
252 }
253
254 /*
255 * Funkcija trazi u listi element cija je vrednost jednaka datom
256 * broju. Funkcija se u pretrazi oslanja na činjenicu da je lista
257 * koja se pretražuje neopadajuće sortirana. Vraca pokazivac na
258 * cvor liste u kome je sadržan traženi broj ili NULL u slučaju da
259 * takav element ne postoji u listi.
260 */
261 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
262 {
263     /*
264      * U konjukciji koja cini uslov ostanka u petlji, bitan je
```



```

266     * redosled!
267     */
268     for (; glava != NULL && glava->vrednost <= broj;
269           glava = glava->sledeci)
270         if (glava->vrednost == broj)
271             return glava;
272
273     /*
274     * Nema trazenog broja u listi i vracamo NULL
275     */
276     return NULL;
277 }
278
279 /*
280 * Funkcija brise u listi na koju pokazuje pokazivac glava onaj
281 * cvor na koji pokazuje pokazivac tekuci. Obratiti paznju da je
282 * kod dvostruke liste ovo mnogo lakse uraditi jer cvor tekuci
283 * sadrzi pokazivace na svog sledbenika i prethodnika u listi.
284 * Pre nego sto fizicki obrisemo tekuci obavezno moramo azurirati
285 * sve pokazivace sledbenika i prethodnika.
286 */
287 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
288 {
289     /*
290     * Ako je tekuci NULL pokazivac nema sta da se brise.
291     */
292     if (tekuci == NULL)
293         return;
294
295     /*
296     * Ako postoji prethodnik od tekuceg onda se postavlja da
297     * njegov sledeci bude sledeci od tekuceg.
298     */
299     if (tekuci->prethodni != NULL)
300         tekuci->prethodni->sledeci = tekuci->sledeci;
301
302     /*
303     * Ako postoji sledbenik tekuceg (cvora koji bismo obrisali)
304     * onda njegov prethodnik treba da bude prethodnik tekuceg.
305     */
306     if (tekuci->sledeci != NULL)
307         tekuci->sledeci->prethodni = tekuci->prethodni;
308
309     /*
310     * Ako je glava element koji se brise, glava nove liste ce
311     * biti sledbenik od stare glave.
312     */
313     if (tekuci == *adresa_glave)
314         *adresa_glave = tekuci->sledeci;
315
316     /*

```

```
318     * Oslobadjamo dinamički alocirani prostor za cvor tekuci.
319     */
320     free(tekuci);
321 }
322
323
324 /*
325  * Funkcija brise iz liste sve cvorove koji sadrze dati broj.
326  * Funkcija azurira pokazivac na glavu liste, koji moze biti
327  * promenjen u slucaju da se obrise stara glava.
328  */
329 void obrisi_element(Cvor ** adresa_glave, int broj)
330 {
331     Cvor *tekuci = *adresa_glave;
332
333     while ((tekuci =
334             pretrazi_listu(*adresa_glave, broj)) != NULL)
335         obrisi_tekuci(adresa_glave, tekuci);
336 }
337
338
339 /*
340  * Funkcija brise iz liste sve cvorove koji sadrze dati broj,
341  * oslanjajuci se na cinjenicu da je prosledjena lista sortirana
342  * neopadajuće. Funkcija azurira pokazivac na glavu liste, koji
343  * moze biti promenjen ukoliko se obrise stara glava liste.
344  */
345 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
346                                     broj)
347 {
348     Cvor *tekuci = *adresa_glave;
349
350     while ((tekuci =
351             pretrazi_sortiranu_listu(*adresa_glave,
352                                     broj)) != NULL)
353         obrisi_tekuci(adresa_glave, tekuci);
354 }
355
356
357 /*
358  * Funkcija prikazuje elemente liste pocev od glave ka kraju
359  * liste. Ne saljemo joj adresu promenljive koja cuva glavu
360  * liste, jer ova funkcija nece menjati listu, pa nema ni
361  * potrebe da azurira pokazivac na glavu liste iz pozivajuće
362  * funkcije.
363  */
364 void ispisi_listu(Cvor * glava)
365 {
366     putchar('[');
367     for (; glava != NULL; glava = glava->sledeci) {
368         printf("%d", glava->vrednost);
```

```

        if (glava->sledeci != NULL)
370         printf(", ");
    }
372
    printf("]\n");
374 }
376
377 /*
378  * Funkcija prikazuje elemente liste pocev od kraja ka glavi
379  * liste. Kod dvostruko povezane to je jako jednostavno jer
380  * svaki cvor ima pokazivac na prethodni element u listi.
381  */
382 void ispisi_listu_u_nazad(Cvor * glava)
383 {
384     putchar('[');
385     if (glava == NULL) {
386         printf("]\n");
387         return;
388     }
389
390     glava = pronadji_poslednji(glava);
391
392     for (; glava != NULL; glava = glava->prethodni) {
393         printf("%d", glava->vrednost);
394         if (glava->prethodni != NULL)
395             printf(", ");
396     }
397     printf("]\n");
398 }

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  int main()
6  {
7      /*
8       * Lista je na pocetku prazna.
9       */
10     Cvor *glava = NULL;
11     Cvor *trazeni = NULL;
12     int broj;
13
14     /*
15     * Testiramo dodavanje na pocetak
16     */
17     printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
18     printf("\n\tLista: ");
19     ispisi_listu(glava);
20
21     while (scanf("%d", &broj) > 0) {
22         dodaj_na_pocetak_liste(&glava, broj);
23     }
24 }

```

```
23     printf("\n\tLista: ");
24     ispisi_listu(glava);
25 }
26
27 printf("\nUnesite element koji se trazi u listi: ");
28 scanf("%d", &broj);
29
30 trazeni = pretrazi_listu(glava, broj);
31 if (trazeni == NULL)
32     printf("Element NIJE u listi!\n");
33 else
34     printf("Trazeni broj %d je u listi!\n",
35           trazeni->vrednost);
36
37 printf("\nLista ispisana u nazad: ");
38 ispisi_listu_u_nazad(glava);
39
40 oslobodi_listu(&glava);
41
42 return 0;
43 }
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  int main()
6  {
7      Cvor *glava = NULL;
8      int broj;
9
10     /*
11     * Testiramo dodavanje na kraj liste
12     */
13     printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
14     printf("\n\tLista: ");
15     ispisi_listu(glava);
16
17     while (scanf("%d", &broj) > 0) {
18         dodaj_na_kraj_liste(&glava, broj);
19         printf("\n\tLista: ");
20         ispisi_listu(glava);
21     }
22
23     printf("\nUnesite element koji se brise iz liste: ");
24     scanf("%d", &broj);
25
26     /*
27     * Brisemo elemente iz liste cije polje vrednost je jednako
28     * broju procitanom sa ulaza
29     */
30     obrisi_element(&glava, broj);
31 }
```

```
31     printf("Lista nakon brisanja: ");
32     ispisi_listu(glava);
33
34     printf("\nLista ispisana u nazad: ");
35     ispisi_listu_u_nazad(glava);
36
37
38     oslobodi_listu(&glava);
39
40     return 0;
41 }

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5
6  int main()
7  {
8      Cvor *glava = NULL;
9      Cvor *trazeni = NULL;
10     int broj;
11
12     /*
13      * Testiramo dodavanje u listu tako da ona bude neopadajuće
14      * uređjena
15      */
16     printf
17         ("\nUnosite elemente liste (za kraj unesite CTRL+D)\n");
18     printf("\n\tLista: ");
19     ispisi_listu(glava);
20
21     while (scanf("%d", &broj) > 0) {
22         dodaj_sortirano(&glava, broj);
23         printf("\n\tLista: ");
24         ispisi_listu(glava);
25     }
26
27     printf("\nUnosite element koji se traži u listi: ");
28     scanf("%d", &broj);
29
30     trazeni = pretraži_sortiranu_listu(glava, broj);
31     if (trazeni == NULL)
32         printf("Element NIJE u listi!\n");
33     else
34         printf("Traženi broj %d je u listi!\n",
35             trazeni->vrednost);
36
37
38     /*
39      * Brisemo elemente iz liste čije polje vrednost je jednako

```

## 4 Dinamičke strukture podataka

```
40     * broju procitanom sa ulaza
41     */
42     printf("\nUnesite element koji se brise iz liste: ");
43     scanf("%d", &broj);
44
45     obrisi_element_sortirane_liste(&glava, broj);
46
47     printf("Lista nakon brisanja: ");
48     ispisi_listu(glava);
49
50     printf("\nLista ispisana u nazad: ");
51     ispisi_listu_u_nazad(glava);
52
53     oslobodi_listu(&glava);
54
55     return 0;
56 }
```

### Rešenje 4.4

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

2
typedef struct cvor {
    char z;
    struct cvor *sledeci;
} Cvor;

8
int main()
{
10     Cvor *stek = NULL;
11     FILE *in = NULL;
12     char pom;
13     Cvor *tmp = NULL;
14
15     in = fopen("dat.txt", "r");
16     if (in == NULL) {
17         fprintf(stderr,
18             "Greska prilikom otvaranja datoteke dat.txt!\n");
19         exit(EXIT_FAILURE);
20     }
21
22     while ((pom = fgetc(in)) != EOF) {
23         /* Ako je učitana otvorena zagrada stavljamo je na stek */
24         if (pom == '(' || pom == '{' || pom == '[') {
25             Cvor *tmp = (Cvor *) malloc(sizeof(Cvor));
26             if (tmp == NULL) {
27                 fprintf(stderr, "Greska prilikom alokacije memorije!\n");
28                 return 1;
29             }
30             tmp->z = pom;

```

```

32     tmp->sledeci = stek;
    stek = tmp;
34 }
/* Ako je učitana zatvorena zagrada proveravamo da stek nije
36 prazan i da li se na vrhu steka nalazi njegova
odgovarajuća otvorena zagrada. */
38 else {
    if (pom == ')' || pom == '}' || pom == ']') {
40         if (stek != NULL && ((stek->z == '(' && pom == ')')
                                || (stek->z == '{' && pom == '}')
                                || (stek->z == '[' && pom == ']')))) {
42             /* Sa vrha steka uklanjamo otvorenu zgradu. */
44             Cvor *tmp = stek->sledeci;
            free(stek);
46             stek = tmp;
        } else {
48             /* Zgrade u aritmetickom izrazu nisu ispravno
uparene. */
50             break;
        }
52     }
}
54 }

56 /* Ako je na kraju stek prazan i procitali smo datoteku,
zgrade su ispravno uparene, u suprotnom, nisu. */
58 if (stek == NULL && pom == EOF)
    printf("Zgrade su ispravno uparene.\n");
60 else {
    printf("Zgrade nisu ispravno uparene.\n");
62
    while (stek != NULL) {
64         /* Oslobadjamo memoriju koja je ostala na steku, u slucaju
neispravnog uparivanja. */
66         Cvor *tmp = stek->sledeci;
        free(stek);
68         stek = tmp;
    }
70 }

72 fclose(in);
    return 0;
74 }

```

### Rešenje 4.5

```

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

```

```

6  #define MAX 100

8  #define OTVORENA 1
   #define ZATVORENA 2

10

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

14

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

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

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

42

44  /* Funkcija prazni stek */
   void oslobodi_stek(Cvor ** adresa_vrha)
   {
46  Cvor *pomocni;
   while (*adresa_vrha != NULL) {
48  pomocni = *adresa_vrha;
   *adresa_vrha = (*adresa_vrha)->sledeci;
50  free(pomocni);
   }
52  }

54  /* Funkcija proverava uspesnost alokacije memorije za cvor novi
   i ukoliko alokacija nije bila uspesna, oslobadja se sva
56  prethodno zauzeta memorija za listu cija pocetni cvor se
   nalazi na adresi adresa_vrha. */

```



```

58 void prover_i_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
{
60     if (novi == NULL) {
        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
62         oslobodi_stek(adresa_vrha);
        exit(EXIT_FAILURE);
64     }
}

66 /* Funkcija postavlja na vrh steka novu etiketu. */
68 void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
{
70     Cvor *novi = napravi_cvor(etiketa);
    prover_i_alokaciju(adresa_vrha, novi);
72     novi->sledeci = *adresa_vrha;
    *adresa_vrha = novi;
74 }

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

86     /* Pokušavamo da skinemo vrednost sa vrha praznog steka i
        imamo gresku. */
88     if (*adresa_vrha == NULL)
        return 0;

90     /* Ako adresa na koju želimo da smestamo etiketu nije NULL
        kopiramo tamo etiketu sa vrha steka. */
92     if (etiketa != NULL)
        strcpy(etiketa, (*adresa_vrha)->etiketa);

94     /* Skidamo element sa vrha steka. */
    pomocni = *adresa_vrha;
    *adresa_vrha = (*adresa_vrha)->sledeci;
    free(pomocni);

100     return 1;
102 }

104 /* Funkcija vraca pokazivac na string koji sadrži etiketu na
    vrhu steka. Ukoliko je stek prazan, vraca NULL. */
106 char *vrh_steka(Cvor * vrh)
{
108     if (vrh == NULL)
        return NULL;

```

```
110     return vrh->etiketa;
111 }
112
113 /* Funkcija prikazuje stek pocev od vrha prema dnu. */
114 void prikazi_stek(Cvor * vrh)
115 {
116     for (; vrh != NULL; vrh = vrh->sledeci)
117         printf("< %s> \n", vrh->etiketa);
118 }
119
120 /* Funkcija iz fajla na koji pokazuje f cita sledecu etiketu, i
121    njeno ime upisuje u niz na koji pokazuje pokazivac etiketa.
122    Funkcija vraca EOF u slucaju da se dodje do kraja fajla pre
123    nego sto se procita etiketa, vraca OTVORENA ako je procitana
124    otvorena etiketa, odnosno ZATVORENA ako je procitana
125    zatvorena etiketa. */
126 int uzmi_etiketu(FILE * f, char *etiketa)
127 {
128     int c;
129     int i = 0;
130
131     /* Stanje predstavlja informaciju dokle smo stali sa citanjem
132        etikete inicijalno smo ga postavili na vrednost VAN_ETIKETE
133        jer jos uvek nismo poceli da citamo. Tip predstavlja
134        informaciju o tipu etikete uzima vrednosti OTVORENA ili
135        ZATVORENA. */
136     int stanje = VAN_ETIKETE;
137     int tip;
138
139     /* HTML je neosetljiv na razliku izmedju malih i velikih
140        slova. U HTML-u etikete BODY i body imaju isto znacenje,
141        dok to u C-u ne vazi. Zato cemo sve etikete prevoditi u
142        zapis samo malim slovima. */
143     while ((c = fgetc(f)) != EOF) {
144         switch (stanje) {
145             case VAN_ETIKETE:
146                 if (c == '<')
147                     stanje = PROCITANO_MANJE;
148                 break;
149             case PROCITANO_MANJE:
150                 if (c == '/') {
151                     /* Citamo zatvarac */
152                     tip = ZATVORENA;
153                 } else {
154                     if (isalpha(c)) {
155                         /* Citamo otvarac */
156                         tip = OTVORENA;
157                         etiketa[i++] = tolower(c);
158                     }
159                 }
160             }
161         /* Sada citamo etiketu i zato menjamo stanje. */
162     }
```

```

162     stanje = U_ETIKETI;
163     break;
164 case U_ETIKETI:
165     if (isalpha(c) && i < MAX - 1) {
166         /* Ako je procitani karakter slovo i nismo premasili
167            maksimalnu dozvoljenu duzinu za etiketu, smestamo
168            procitani karakter u etiketu. */
169         etiketa[i++] = tolower(c);
170     } else {
171         stanje = VAN_ETIKETE;
172         /* U suprotnom, prestajemo sa citanjem etikete i menjamo
173            stanje. */
174         etiketa[i] = '\0';
175         /* Zavrшили smo sa citanjem etikete i vracamo tip
176            etikete koju smo procitali, a ona nam je sacuvana u
177            nisci etiketa. */
178         return tip;
179     }
180     break;
181 }
182 }

184 /* Dosli smo do kraja datoteke pre nego sto smo završili sa
185    citanjem etikete, stoga imamo gresku i vracamo EOF. */
186 return EOF;
187 }

188 int main(int argc, char **argv)
189 {
190     /* Stek nam je prazan na pocetku. */
191     Cvor *vrh = NULL;
192     char etiketa[MAX];
193     int tip;
194     /* Na pocetku su nam etikete upare, jer nismo nijednu jos
195        procitali. */
196     int uparene = 1;
197     FILE *f = NULL;
198     /* Ime datoteke dobijamo iz komandne linije. */
199     if (argc < 2) {
200         fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n",
201             argv[0]);
202         exit(0);
203     }
204 }

206 /* Otvaramo datoteku. */
207 if ((f = fopen(argv[1], "r")) == NULL) {
208     fprintf(stderr, "fopen() greska!\n");
209     exit(1);
210 }
211 /* Dokle god ima etiketa, uzimamo ih jednu po jednu sa ulaza. */
212 while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
213     /* Ako je otvorena etiketa, dodajemo je na stek. Izuzetak su

```

```
214     etikete <br>, <hr> i <meta> koje nemaju sadržaj, tako da
216     ih nije potrebno zatvoriti. NAPOMENA: U html-u postoje
218     jos neke etikete koje nemaju sadržaj (npr link).
218     Pretpostavimo da njih nema u dokumentu, zbog
218     jednostavnosti. */
218     if (tip == OTVORENA) {
220         if (strcmp(etiketa, "br") != 0
220             && strcmp(etiketa, "hr") != 0
222             && strcmp(etiketa, "meta") != 0)
222             potisni_na_stek(&vrh, etiketa);
224     }
224     /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti
226     da je u pitanju zatvaranje etikete koja je poslednja
226     otvorena, a jos uvek nije zatvorena. Ova etiketa se mora
228     nalaziti na vrhu steka. Ako je taj uslov ispunjen, tada
228     je skidamo sa steka, jer je zatvorena. U suprotnom,
230     obavestavamo korisnika da etikete nisu pravilno uparene. */
230     else if (tip == ZATVORENA) {
232         if (vrh_steka(vrh) != NULL
232             && strcmp(vrh_steka(vrh), etiketa) == 0)
234             skini_sa_steka(&vrh, NULL);
234         else {
236             printf(vrh_steka(vrh) !=
236                 NULL ? "Etikete nisu pravilno uparene\n(nadjena\
238 etiketa </%s> a poslednja otvorena etiketa je <%s>)\n" : "Etikete
238     nisu pravilno uparene\n(nadjena etiketa </%s> koja nije\
238     otvorena)\n", etiketa, vrh_steka(vrh));
240             uparene = 0;
240             break;
242         }
242     }
244 }
244 /* Zatvaramo datoteku. */
246 fclose(f);
246 /* Ako nismo pronasli pogresno uparivanje do sada, stek treba
248 da bude prazan. Ako nije, tada znaci da postoje jos neke
248 etikete koje su otvorene ali nisu bile zatvorene. */
250 if (uparene) {
252     if (vrh_steka(vrh) == NULL)
252         printf("Etikete su pravilno uparene!\n");
252     else
254         printf("Etikete nisu pravilno uparene\n(etiketa <%s>
254 nije zatvorena)\n", vrh_steka(vrh));
256 }
258 /* Oslobadjamo memoriju alociranu za stek, ukoliko vec nije. */
258 oslobodi_stek(&vrh);
260 return 0;
}
```

### Rešenje 4.6

```

1  #ifndef _RED_H
   #define _RED_H
3
   #include <stdio.h>
5  #include <stdlib.h>
7
   #define MAX 1000
   #define JMBG_DUZINA 14
9
   /* Struktura kojom predstavljamo zahtev korisnika, obuhvata JMBG
11  korisnika i opis njegovog zahteva. */
   typedef struct {
13     char jmbg[JMBG_DUZINA];
       char opis[MAX];
15 } Zahtev;
17
   /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
       korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
       Zahtev nalog;
21     struct cvor *sledeci;
   } Cvor;
23
   Cvor * napravi_cvor( Zahtev * zahtev);
25
   void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
27
   void prover_i_alokaciju(Cvor ** adresa_pocetka,
29                        Cvor ** adresa_kraja, Cvor* novi) ;
31
   void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
       Zahtev* zahtev);
33
   int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
35                  Zahtev *zahtev);
37
   Zahtev * pocetak_reda(Cvor * pocetak) ;
39
   void prikazi_red(Cvor * pocetak);
41 #endif

```

```

1  #include "red.h"
3
   /* Funkcija kreira novi cvor, inicijalizuje polje nalog na
       zahtev sa poslate adrese i vraca adresu novog cvora ili NULL
5  ako je doslo do greske pri alokaciji. Ako je doslo do greske,
       trebalo bi osloboditi ceo red. Ostavljamo da to uradi funkcija
7  koja je pozvala funkciju napravi_cvor, a gresku signaliziramo
       saljuci joj NULL. Funkciji se prosledjuje pokazivac na zahtev
9  koji treba smestiti u nov cvor zbog smestanja manjeg podatka

```

```

11     na sistemski stek. Pokazivac na strukturu Zahtev je manje
    velicine u bajtovima(B) u odnosu na strukturu Zahtev. */
13 Cvor *napravi_cvor(Zahtev * zahtev)
14 {
15     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
16     if (novi == NULL)
17         return NULL;
18
19     novi->nalog = *zahtev;
20     novi->sledeci = NULL;
21     return novi;
22 }
23
24 /* Funkcija prazni red */
25 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
26 {
27     Cvor * pomocni = NULL;
28
29     while (*pocetak != NULL) {
30         pomocni = *pocetak;
31         *pocetak = (*pocetak)->sledeci;
32         free(pomocni);
33     }
34     *kraj = NULL;
35 }
36
37 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
    i ukoliko alokacija nije bila uspesna, oslobadja se sva
    prethodno zauzeta memorija za listu cija pocetni cvor se
    nalazi na adresi adresa_pocetka. */
39 void prover_i_alokaciju(Cvor ** adresa_pocetka,
40                         Cvor ** adresa_kraja, Cvor * novi)
41 {
42     if (novi == NULL) {
43         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
44         oslobodi_red(adresa_pocetka, adresa_kraja);
45         exit(EXIT_FAILURE);
46     }
47 }
48
49
51 /* Funkcija dodaje na kraj reda novi fajl. */
52 void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
53                 Zahtev * zahtev)
54 {
55     Cvor * novi = napravi_cvor(zahtev);
56     prover_i_alokaciju(adresa_pocetka, adresa_kraja, novi);
57
58     /* U red se uvek dodaje na kraj, ali zbog postojanja
59        pokazivaca na kraj, dodavanje na kraj je podjednako
60        efikasno kao dodavanje na pocetak. */
61     if (*adresa_kraja != NULL) {

```

```

        (*adresa_kraja)->sledeci = novi;
63     *adresa_kraja = novi;
    } else {
65         /* Ako je red bio ranije prazan */
        *adresa_pocetka = novi;
67         *adresa_kraja = novi;
    }
69 }

71 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji
    argument pokazivac razlicit od NULL, tada se u strukturu na
73 koju on pokazuje upisuje zahtev koji je upravo skinut sa reda
    dok u suprotnom ne upisuje nista. Funkcija vraca 0 ako je red
75 bio prazan ili 1 u suprotnom. */
int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
77                 Zahtev * zahtev)
{
79     Cvor *pomocni = NULL;

81     if (*adresa_pocetka == NULL)
        return 0;

83     if (zahtev != NULL)
85         *zahtev = (*adresa_pocetka)->nalog;

87     pomocni = *adresa_pocetka;
    *adresa_pocetka = (*adresa_pocetka)->sledeci;
89     free(pomocni);

91     if (*adresa_pocetka == NULL)
        *adresa_kraja = NULL;

93     return 1;
95 }

97
99 /* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev
    korisnika na pocetku reda. Ukoliko je red prazan, vraca NULL.
    */
101 Zahtev * pocetak_reda(Cvor * pocetak)
    {
103         if (pocetak == NULL)
            return NULL;

105         return &(pocetak->nalog);
107     }

109 /* Funkcija prikazuje red. */
void prikazi_red(Cvor * pocetak)
111 {
    for (; pocetak != NULL; pocetak = pocetak->sledeci)
113         printf("%s %s\n",

```

```

115         (pocetak->nalog).jmbg, (pocetak->nalog).opis);
117     printf("\n");
}

1  #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     FILE *izlaz = fopen("izvestaj.txt", "a");
17
18     if (izlaz == NULL) {
19         fprintf(stderr, "Neuspesno otvaranje datoteke \
izvestaj.txt\n");
21         exit(EXIT_FAILURE);
22     }
23
24     /* Sluzbenik evidentira korisnicke zahteve. */
25     printf("Sluzbenik evidentira korisnicke zahteve unosenjem \
njihovog JMBG broja i opisa potrebne usluge:\n[CTRL+D za \
kraj]\n");
27
28     /* Neophodan je poziv funkcije getchar da bi se i nov red
29     nakon JMBG broja procitao i da bi fgets nakon toga
30     procitala ispravan red sa opisom zahteva. */
31     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
32     while (scanf("%s", nov_zahtev.jmbg) != EOF) {
33         getchar();
34         printf("\tOpis problema: ");
35         fgets(nov_zahtev.opis, MAX - 1, stdin);
36         /* Ako je poslednji karakter nov red, eliminisemo ga. */
37         if (nov_zahtev.opis[strlen(nov_zahtev.opis)-1] == '\n')
38             nov_zahtev.opis[strlen(nov_zahtev.opis)-1] = '\0';
39         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
40         printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
41     }
42
43     /* Dokle god ima korisnika u redu, usluzujemo ih. */
44     while (1) {
45         sledeci = pocetak_reda(pocetak);
46         /* Ako nema nikog vise u redu. */

```



```

49     if (sledeci == NULL)
        break;

51     printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
        sledeci->jmbg);
53     printf("sa zahtevom: %s\n", sledeci->opis);

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

57     broj_usluzenih++;

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

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

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

71             if (strcmp(odgovor, "Da") == 0)
73                 break;
            else
75                 broj_usluzenih = 0;
        }
77     }

79     fclose(izlaz);

81     /* Oslobadjamo red ukoliko je sluzbenik prekinuo sa radom
        mozda je bilo jos neusluzenih korisnika. */
83     oslobodi_red(&pocetak, &kraj);

85     return 0;
}

```

```

/* Alternativno resenje bez koriscenja funkcije za skidanje
2   zahteva sa pocetak reda. */

4   #include <stdio.h>
   #include <stdlib.h>
6   #include <string.h>
   #include "red.h"

8   #define VREME_ZA_PAUZU 5

10  int main(int argc, char **argv)
12  {

```

```
14      /* Red je prazan. */
15      Cvor *pocetak = NULL, *kraj = NULL;
16      Zahtev nov_zahtev;
17      Zahtev *sledeci = NULL;
18      char odgovor[3];
19      int broj_usluzenih = 0;
20      FILE *izlaz = fopen("izvestaj.txt", "a");
21
22      if (izlaz == NULL) {
23          fprintf(stderr, "Neuspesno otvaranje datoteke \
24      izvestaj.txt\n");
25          exit(EXIT_FAILURE);
26      }
27
28      /* Sluzbenik evidentira korisnicke zahteve. */
29      printf("Sluzbenik evidentira korisnicke zahteve unosnjem \
30      njihovog JMBG broja i opisa potrebne usluge:\n");
31      /* Neophodan je poziv funkcije getchar da bi se i nov red
32      nakon JMBG broja procitao i da bi fgets nakon toga
33      procitala ispravan red sa opisom zahteva. */
34      printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
35      while (scanf("%s", nov_zahtev.jmbg) != EOF) {
36          getchar();
37          printf("\tOpis problema: ");
38          fgets(nov_zahtev.opis, MAX - 1, stdin);
39          /* Ako je poslednji karakter nov red, eliminisemo ga. */
40          if (nov_zahtev.opis[strlen(nov_zahtev.opis)-1] == '\n')
41              nov_zahtev.opis[strlen(nov_zahtev.opis)-1] = '\0';
42          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
43          printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
44      }
45
46      /* Dokle god ima korisnika u redu, usluujemo ih. */
47      while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
48          printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
49              nov_zahtev.jmbg);
50          printf("sa zahtevom: %s\n", nov_zahtev.opis);
51
52          broj_usluzenih++;
53
54          printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
55          scanf("%s", odgovor);
56
57          if (strcmp(odgovor, "Da") == 0)
58              dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
59          else
60              fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
61                  nov_zahtev.jmbg, nov_zahtev.opis);
62
63          if (broj_usluzenih == VREME_ZA_PAUZU) {
64              printf("\nDa li je kraj smene? [Da/Ne] ");
65              scanf("%s", odgovor);
```

```

66         if (strcmp(odgovor, "Da") == 0)
67             break;
68         else
69             broj_usluzenih = 0;
70     }
71 }
72
73 fclose(izlaz);
74
75 /* Oslobadjamo red ukoliko je sluzbenik prekinuo sa radom
76    mozda je bilo jos neusluzenih korisnika. */
77 oslobodi_red(&pocetak, &kraj);
78
79 return 0;
80 }

```

### Rešenje 4.7

```

1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  #define MAX_DUZINA 20
5
6  typedef struct _Element {
7      unsigned broj_pojavljivanja;
8      char etiketa[20];
9      struct _Element *sledeci;
10 } Element;
11
12 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi
13    cvor ili NULL ako alokacija nije uspesno izvrшена. */
14 Element *napravi_cvor(unsigned br, char *etiketa)
15 {
16     Element *novi = (Element *) malloc(sizeof(Element));
17     if (novi == NULL)
18         return NULL;
19
20     novi->broj_pojavljivanja = br;
21     strcpy(novi->etiketa, etiketa);
22     novi->sledeci = NULL;
23     return novi;
24 }
25
26 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
27    liste. */
28 void oslobodi_listu(Element ** glava)
29 {
30     Element *pomocni = NULL;
31
32     while (*glava != NULL) {

```

```

33     pomocni = (*glava)->sledeci;
34     free(*glava);
35     *glava = pomocni;
36 }
37 }

39 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
40 i ukoliko alokacija nije bila uspesna, oslobadja se sva
41 prethodno zauzeta memorija za listu cija pocetni cvor se
42 nalazi na adresi glava. */
43 void proveraj_alokacije(Element * novi, Element ** glava)
44 {
45     if (novi == NULL) {
46         fprintf(stderr, "malloc() greska u funkciji
47 napravi_cvor()!\n");
48         oslobodi_listu(glava);
49         exit(EXIT_FAILURE);
50     }
51 }

53 /* Funkcija dodaje novi cvor na pocetak liste. */
54 void dodaj_na_pocetak_liste(Element ** glava, unsigned br,
55                             char *etiketa)
56 {
57     Element *novi = napravi_cvor(br, etiketa);
58     proveraj_alokacije(novi, glava);
59     novi->sledeci = *glava;
60     *glava = novi;
61 }

63 /* Funkcija vraća cvor koji kao vrednost sadrži traženu etiketu.
64 (NULL u suprotnom) */
65 Element *pretrazi_listu(Element * glava, char etiketa[])
66 {
67     Element *tekuci;
68     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
69         if (strcmp(tekuci->etiketa, etiketa) == 0)
70             return tekuci;
71     return NULL;
72 }

73 /* Funkcija ispisuje sadržaj liste */
74 void ispisi_listu(Element * glava)
75 {
76     for (; glava != NULL; glava = glava->sledeci)
77         printf("%s - %u\n", glava->etiketa,
78             glava->broj_pojavljivanja);
79 }

81 int main(int argc, char **argv)
82 {
83     if (argc != 2) {

```

```
85     fprintf(stderr, "Greska! Program se poziva sa: ./a.out
datoteka.html!\n");
87     exit(EXIT_FAILURE);
}

89     FILE *in = NULL;
91     in = fopen(argv[1], "r");
93     if (in == NULL) {
        fprintf(stderr,
95         "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
        exit(EXIT_FAILURE);
    }

97
99     char c;
100     int i = 0;
101     char a[MAX_DUZINA];

102     Element *glava = NULL;
103     Element *trazeni = NULL;

104     while ((c = fgetc(in)) != EOF) {
105
106         if (c == '<') {
107             /* Citamo zatvarac */
108             if ((c = fgetc(in)) == '/') {
109                 i = 0;
110                 while ((c = fgetc(in)) != '>')
111                     a[i++] = c;
112             }
113
114             /* Citamo otvarac */
115             else {
116                 i = 0;
117                 a[i++] = c;
118                 while ((c = fgetc(in)) != ' ' && c != '>')
119                     a[i++] = c;
120             }
121             a[i] = '\0';
122
123             /* Ispitujemo da li medju do sada formiranim cvorovima
124              postoji cvor sa ucitanom etiketom. Ukoliko ne postoji,
125              dodajemo novi cvor za ucitanu etiketu (broj
126              pojavljivanja postavljamo na 1), inace uvecavamo broj
127              pojavljivanja. */
128             trazeni = pretrazi_listu(glava, a);
129             if (trazeni == NULL)
130                 dodaj_na_pocetak_liste(&glava, 1, a);
131             else
132                 trazeni->broj_pojavljivanja++;
133         }
134     }
135 }
```

```
137     ispisi_listu(glava);
      oslobodi_listu(&glava);
139
      fclose(in);
141     return 0;
  }
```

### Rešenje 4.8

```
1  #include<stdio.h>
   #include<stdlib.h>
3  #include "601/lista.h"

5  Cvor *objedini(Cvor ** glava1, Cvor ** glava2)
   {
7     Cvor *l3 = NULL;
      Cvor **tek = &l3;

9
      if (*glava1 == NULL && *glava2 == NULL)
11         return NULL;

13     /* Ako je prva lista prazna, onda je rezultat druga lista. */
      if (*glava1 == NULL)
15         return *glava2;

17     /* Ako je druga lista prazna, onda je rezultat prva lista. */
      if (*glava2 == NULL)
19         return *glava1;

21     /* l3 pokazuje na pocetak nove liste, tj. na manji od brojeva
      sadrzanih u cvorovima na koje pokazuju glava1 i glava2. */
23     l3 = ((*glava1)->vrednost < (*glava2)->vrednost) ? *glava1 :
      *glava2;

25

27     while (*glava1 != NULL && *glava2 != NULL) {
          if ((*glava1)->vrednost < (*glava2)->vrednost) {
29             *tek = *glava1;
              *glava1 = (*glava1)->sledeci;
31         } else {
              *tek = *glava2;
              *glava2 = (*glava2)->sledeci;
33         }
          tek = &((*tek)->sledeci);
35     }

37

39     /* Ukoliko smo izašli iz petlje zato što smo stigli do kraja
      prve liste onda na rezultujuću listu nadovezujemo ostatak
      druge liste. */
41     if (*glava1 == NULL)
        *tek = *glava2;
```

```
43     else if (*glava2 == NULL)
44         *tek = *glava1;
45
46     return l3;
47 }
48
49 int main(int argc, char **argv)
50 {
51     if (argc != 3) {
52         fprintf(stderr,
53             "Greska! Program se poziva sa: ./a.out dat1.txt dat2.txt
54             !\n");
55         exit(EXIT_FAILURE);
56     }
57
58     FILE *in1 = NULL;
59     in1 = fopen(argv[1], "r");
60     if (in1 == NULL) {
61         fprintf(stderr,
62             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
63         exit(EXIT_FAILURE);
64     }
65
66     FILE *in2 = NULL;
67     in2 = fopen(argv[2], "r");
68     if (in2 == NULL) {
69         fprintf(stderr,
70             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
71         exit(EXIT_FAILURE);
72     }
73
74     int broj;
75     Cvor *glava1 = NULL;
76     Cvor *glava2 = NULL;
77     Cvor *l3 = NULL;
78
79     /* Ucitavamo liste. */
80     while (fscanf(in1, "%d", &broj) != EOF)
81         dodaj_na_kraj_liste(&glava1, broj);
82     while (fscanf(in2, "%d", &broj) != EOF)
83         dodaj_na_kraj_liste(&glava2, broj);
84
85     /* Objedinjujemo ih u jednu listu. */
86     l3 = objedini(&glava1, &glava2);
87
88     /* Ispisujemo rezultujucu listu. */
89     ispisi_listu(l3);
90     oslobodi_listu(&l3);
91
92     fclose(in1);
93     fclose(in2);
```

```
    return 0;
95 }
```

### Rešenje 4.9

```
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  typedef struct _Cvor {
9      char broj_indeksa[MAX_INDEKS];
10     char ime[MAX_IME_PREZIME];
11     char prezime[MAX_IME_PREZIME];
12     struct _Cvor *sledeci;
13 } Cvor;
14
15 /* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac
16    na nov cvor ili NULL ukoliko alokacija nije prosla. */
17 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
18 {
19     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
20     if (novi == NULL)
21         return NULL;
22     strcpy(novi->broj_indeksa, broj_indeksa);
23     strcpy(novi->ime, ime);
24     strcpy(novi->prezime, prezime);
25     novi->sledeci = NULL;
26     return novi;
27 }
28
29 /* Funkcija oslobadja memoriju zauzetu za elemente liste. */
30 void oslobodi_listu(Cvor ** glava)
31 {
32     if (*glava == NULL)
33         return;
34     oslobodi_listu(&(*glava)->sledeci);
35     free(*glava);
36     *glava = NULL;
37 }
38
39 void prover_i_alokaciju(Cvor ** glava, Cvor * novi)
40 {
41     if (novi == NULL) {
42         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
43         oslobodi_listu(glava);
44         exit(EXIT_FAILURE);
45     }
46 }
```



```

47  /* Funkcija dodaje novi cvor na pocetak liste. */
49  void dodaj_na_pocetak_liste(Cvor ** glava, char *broj_indeksa,
                                char *ime, char *prezime)
51  {
    Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
53  proveri_alokaciju(glava, novi);
    novi->sledeci = *glava;
55  *glava = novi;
    }

57  void ispisi_listu(Cvor * glava)
59  {
    for (; glava != NULL; glava = glava->sledeci)
61      printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
                glava->prezime);
63  }

65  /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu,
    u suprotnom vraca NULL. */
67  Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
    {
69      if (glava == NULL)
          return NULL;
71      if (!strcmp(glava->broj_indeksa, broj_indeksa))
          return glava;
73      return pretrazi_listu(glava->sledeci, broj_indeksa);
    }

75  int main(int argc, char **argv)
77  {
    if (argc != 2) {
79      fprintf(stderr, "Greska! Program se poziva sa: ./a.out \
studenti.txt!\n");
81      exit(EXIT_FAILURE);
    }

83
    FILE *in = NULL;
85    in = fopen(argv[1], "r");
    if (in == NULL) {
87        fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
89        exit(EXIT_FAILURE);
    }

91
    char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
93    char broj_indeksa[MAX_INDEKS];
    Cvor *glava = NULL;
95    Cvor *trazeni = NULL;

97    /* Ucitavamo listu sa standardnog ulaza. */
    while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) !=

```

```
99         EOF)
        dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);
101
102     while (scanf("%s", broj_indeksa) != EOF) {
103         trazeni = pretrazi_listu(glava, broj_indeksa);
104         if (trazeni == NULL)
105             printf("ne\n");
106         else
107             printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
108     }
109
110     oslobodi_listu(&glava);
111     fclose(in);
112     return 0;
113 }
```

Rešenje 4.10

Rešenje 4.11

Rešenje 4.12

Rešenje 4.13

Rešenje 4.14

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* a) Struktura kojom se predstavlja cvor binarnog
   pretrazivackog stabla */
5 typedef struct cvor {
6     int broj;
7     struct cvor *levo;
8     struct cvor *desno;
9 } Cvor;
10
11 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
12 Cvor *napravi_cvor(int broj)
13 {
14     /* Alociramo memoriju */
15     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
16     if (novi == NULL)
17         return NULL;
18
19     /* Inicijalizujemo polja novog cvora. */
20 }
```

```
22     novi->broj = broj;
23     novi->levo = NULL;
24     novi->desno = NULL;

26     /* Vracamo adresu novog cvora. */
27     return novi;
28 }

30
31 /* Funkcija koja proverava uspesnost kreiranja novog cvora
32    stabla */
33 void prover_i_alokaciju(Cvor * novi_cvor)
34 {
35     /* Ukoliko je cvor neuspesno kreiran */
36     if (novi_cvor == NULL) {

38         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
39            programa */
40         fprintf(stderr, "Malloc greska za novi cvor!\n");
41         exit(EXIT_FAILURE);
42     }
43 }

44

46 /* c) Funkcija koja dodaje zadati broj u stablo */
47 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
48 {
49     /* Ako je stablo prazno */
50     if (*adresa_korena == NULL) {

52         /* Kreiramo novi cvor */
53         Cvor *novi = napravi_cvor(broj);
54         prover_i_alokaciju(novi);

56         /* I proglašavamo ga korenom stabla */
57         *adresa_korena = novi;
58         return;
59     }

60     /* U suprotnom trazimo odgovarajucu poziciju za zadati broj */

62     /* Ako je zadata vrednost manja od vrednosti korena */
63     if (broj < (*adresa_korena)->broj)

65         /* Dodajemo broj u levo podstablo */
66         dodaj_u_stablo(&(*adresa_korena)->levo, broj);

68     else

70         /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa
71            ga dodajemo u desno podstablo */
72         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
73 }
```

```
74 |
76 | /* d) Funkcija koja proverava da li se zadati broj nalazi u
    | stablu */
78 | Cvor *pretrazi_stablo(Cvor * koren, int broj)
    | {
80 |     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
    |     */
82 |     if (koren == NULL)
    |         return NULL;
84 |
    |     /* Ako je trazena vrednost sadrazana u korenu */
86 |     if (koren->broj == broj) {
    |
88 |         /* Prekidamo pretragu */
    |         return koren;
90 |     }
    |
92 |     /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
    |     if (broj < koren->broj)
94 |
    |         /* Pretragu nastavljamo u levom podstablu */
96 |         return pretrazi_stablo(koren->levo, broj);
    |
98 |     else
    |         /* U suprotnom, pretragu nastavljamo u desnom podstablu */
100 |         return pretrazi_stablo(koren->desno, broj);
    | }
102 |
104 | /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost u
    | stablu */
106 | Cvor *pronadji_najmanji(Cvor * koren)
    | {
108 |     /* Ako je stablo prazno, prekidamo pretragu */
    |     if (koren == NULL)
110 |         return NULL;
    |
112 |     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze
    |     se levo od njega */
114 |
    |     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
    |     najmanju vrednost */
116 |     if (koren->levo == NULL)
    |         return koren;
118 |
    |     /* Inace, pretragu treba nastaviti u levom podstablu */
120 |     return pronadji_najmanji(koren->levo);
    | }
122 |
124 | /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u
```

```
126     stablu */
127 Cvor *pronadji_najveci(Cvor * koren)
128 {
129     /* Ako je stablo prazno, prekidamo pretragu */
130     if (koren == NULL)
131         return NULL;
132
133     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze
134        se desno od njega */
135
136     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
137        najveću vrednost */
138     if (koren->desno == NULL)
139         return koren;
140
141     /* Inace, pretragu treba nastaviti u desnom podstablu */
142     return pronadji_najveci(koren->desno);
143 }
144
145 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
146 void obrisi_element(Cvor ** adresa_korena, int broj)
147 {
148     Cvor *pomocni_cvor = NULL;
149
150     /* ako je stablo prazno, brisanje nije primenljivo pa mozemo
151        prekinuti rad funkcije */
152     if (*adresa_korena == NULL)
153         return;
154
155     /* Ako je vrednost koju treba obrisati manja od vrednosti u
156        korenu stabla, ona se eventualno nalazi u levom podstablu,
157        pa treba rekurzivno primeniti postupak na levo podstablo.
158        Koren ovako modifikovanog stabla je nepromenjen. */
159     if (broj < (*adresa_korena)->broj) {
160         obrisi_element(&(*adresa_korena)->levo, broj);
161         return;
162     }
163
164     /* Ako je vrednost koju treba obrisati veca od vrednosti u
165        korenu stabla, ona se eventualno nalazi u desnom podstablu
166        pa treba rekurzivno primeniti postupak na desno podstablo.
167        Koren ovako modifikovanog stabla je nepromenjen. */
168     if ((*adresa_korena)->broj < broj) {
169         obrisi_element(&(*adresa_korena)->desno, broj);
170         return;
171     }
172
173     /* Slede podslucajevi vezani za slucaj kada je vrednost u
174        korenu jednaka broju koji se brise (tj. slucaj kada treba
175        obrisati koren) */
```

```
178  /* Ako koren nema sinova, tada se on prosto brise, i rezultat
    je prazno stablo (vracamo NULL) */
180  if ((*adresa_korena)->levo == NULL
    && (*adresa_korena)->desno == NULL) {
182      free(*adresa_korena);
    *adresa_korena = NULL;
184      return;
    }
186
188  /* Ako koren ima samo levog sina, tada se brisanje vrshi tako
    sto obrisemo koren, a novi koren postaje levi sin */
190  if ((*adresa_korena)->levo != NULL
    && (*adresa_korena)->desno == NULL) {
192      pomocni_cvor = (*adresa_korena)->levo;
    free(*adresa_korena);
    *adresa_korena = pomocni_cvor;
194      return;
    }
196
198  /* Ako koren ima samo desnog sina, tada se brisanje vrshi tako
    sto obrisemo koren, a novi koren postaje desni sin */
200  if ((*adresa_korena)->desno != NULL
    && (*adresa_korena)->levo == NULL) {
202      pomocni_cvor = (*adresa_korena)->desno;
    free(*adresa_korena);
    *adresa_korena = pomocni_cvor;
204      return;
    }
206
208  /* Slucaj kada koren ima oba sina. Tada se brisanje vrshi na
    sledeci nacin: - najpre se potrazi sledbenik korena (u
    smislu poretka) u stablu. To je upravo po vrednosti
    najmanji cvor u desnom podstablu. On se moze pronaci npr.
    funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
    vrednost tog cvora, a u taj cvor se smesti vrednost korena
    (tj. broj koji se brise). - Onda se prosto rekursivno
    pozove funkcija za brisanje na desno podstablo. S obzirom
    da u njemu treba obrisati najmanji element, a on zasigurno
    ima najvise jednog potomka, jasno je da ce njegovo brisanje
    biti obavljeno na jedan od jednostavnijih nacina koji su
    gore opisani. */
218  pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
    (*adresa_korena)->broj = pomocni_cvor->broj;
    pomocni_cvor->broj = broj;
222  obrisi_element(&(*adresa_korena)->desno, broj);
    }
224
226  /* h) Funkcija ispisuje stablo u infiksnoj notaciji ( Levo
    postablo - Koren - Desno podstablo ) */
228  void ispisi_stablo_infiksno(Cvor * koren)
    {
```

```
230  /* Ako stablo nije prazno */
    if (koren != NULL) {
232
        /* Prvo ispisujemo sve cvorove levo od korena */
234        ispisi_stablo_infiksno(koren->levo);

        /* Ispisujemo vrednost u korenu */
236        printf("%d ", koren->broj);

238        /* Na kraju ispisujemo cvorove desno od korena */
240        ispisi_stablo_infiksno(koren->desno);
    }
242 }

244
/* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren -
246     Levo podstablo - Desno podstablo ) */
void ispisi_stablo_prefiksno(Cvor * koren)
248 {
    /* Ako stablo nije prazno */
250    if (koren != NULL) {

        /* Prvo ispisujemo vrednost u korenu */
252        printf("%d ", koren->broj);

254        /* Ispisujemo sve cvorove levo od korena */
256        ispisi_stablo_prefiksno(koren->levo);

258        /* Na kraju ispisujemo sve cvorove desno od korena */
        ispisi_stablo_prefiksno(koren->desno);
    }
260 }
262

264 /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo
     podstablo - Desno postablo - Koren ) */
266 void ispisi_stablo_postfiksno(Cvor * koren)
    {
268        /* Ako stablo nije prazno */
        if (koren != NULL) {

270            /* Prvo ispisujemo sve cvorove levo od korena */
272            ispisi_stablo_postfiksno(koren->levo);

274            /* Ispisujemo sve cvorove desno od korena */
            ispisi_stablo_postfiksno(koren->desno);

276            /* Na kraju ispisujemo vrednost u korenu */
278            printf("%d ", koren->broj);
        }
280    }
```

```

282  /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
284  void oslobodi_stablo(Cvor ** adresa_korena)
286  {
288      /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
290      if (*adresa_korena == NULL)
292          return;

294      /* U suprotnom rekurzivno oslobadjamo memoriju koje zauzima
296      najpre levo, a zatim i desno podstablo */
298      oslobodi_stablo(&(*adresa_korena)->levo);
299      oslobodi_stablo(&(*adresa_korena)->desno);

301      /* Oslobadjamo memoriju koju zauzima koren */
302      free(*adresa_korena);

304      /* I proglašavamo stablo praznim */
305      *adresa_korena = NULL;
306  }

308  int main()
309  {
310      Cvor *koren;
311      int n;
312      Cvor *trazeni_cvor;

314      /* Proglašavamo stablo praznim */
315      koren = NULL;

317      /* Dodajemo vrednosti u stablo */
318      printf("Unesite brojeve (CTRL+D za kraj unosa): ");
319      while (scanf("%d", &n) != EOF) {
320          dodaj_u_stablo(&koren, n);
321      }

323      /* Generisemo trazene ispise: */
324      printf("\nInfiksni ispisi: ");
325      ispisi_stablo_infiksno(koren);
326      printf("\nPrefiksni ispisi: ");
327      ispisi_stablo_prefiksno(koren);
328      printf("\nPostfiksni ispisi: ");
329      ispisi_stablo_postfiksno(koren);

331      /* Demonstriramo rad funkcije za pretragu */
332      printf("\nTraži se broj: ");
333      scanf("%d", &n);
334      trazeni_cvor = pretrazi_stablo(koren, n);
335      if (trazeni_cvor == NULL)
336          printf("Broj se ne nalazi u stablu!\n");
337      else
338          printf("Broj se nalazi u stablu!\n");

```



```

334  /* Demonstriramo rad funkcije za brisanje */
    printf("Brise se broj: ");
336  scanf("%d", &n);
    obrisi_element(&koren, n);
338  printf("Rezultujuce stablo: ");
    ispisi_stablo_infiksno(koren);
340  printf("\n");

342  /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);

344

    /* Prekidamo sa programom */
346  return 0;
}

```

### Rešenje 4.15

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <ctype.h>
5
   #define MAX 50
7
   /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
9   pojavljivanja i redom pokazivace na levo i desno podstablo */
   typedef struct cvor {
11     char *rec;
       int brojac;
13     struct cvor *levo;
       struct cvor *desno;
15 } Cvor;

17 /* Funkcija koja kreira novi cvora stabla */
   Cvor *napravi_cvor(char *rec)
19 {
       /* Alociramo memoriju za novi cvor */
21     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
       if (novi_cvor == NULL)
23         return NULL;

25     /* Alociramo memoriju za zadatu rec: potrebno je rezervisati
       memoriju za svaki karakter reci ukljucujuci i terminirajucu
27     nulu */
       novi_cvor->rec =
29         (char *) malloc((strlen(rec) + 1) * sizeof(char));
       if (novi_cvor->rec == NULL) {
31         free(novi_cvor);
           return NULL;
33     }
}

```

```

35  /* Inicijalizujemo polja u novom cvoru */
    strcpy(novi_cvor->rec, rec);
37  novi_cvor->brojac = 1;
    novi_cvor->levo = NULL;
39  novi_cvor->desno = NULL;

41  /* Vracamo adresu novog cvora */
    return novi_cvor;
43 }

45 /* Funkcija koja proverava uspesnost kreiranja novog cvora
    stabla */
47 void prover_i_alokaciju(Cvor * novi_cvor)
{
49  /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
51      /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
        programa */
53      fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
55  }
}

57 /* Funkcija koja dodaje novu rec u stablo. */
59 void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
{
61  /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
63      /* Kreiramo novi cvor */
        Cvor *novi = napravi_cvor(rec);
65      prover_i_alokaciju(novi);

67      /* i proglašavamo ga korenom stabla */
        *adresa_korena = novi;
69      return;
    }

71  /* U suprotnom trazimo odgovarajucu poziciju za novu rec */

73  /* Ako je rec leksikografski manju od reci u korenu ubacujemo
    je u levo podstablo */
75  if (strcmp(rec, (*adresa_korena)->rec) < 0)
77      dodaj_u_stablo(&(*adresa_korena)->levo, rec);

79  else
81      /* Ako je rec leksikografski veca od reci u korenu ubacujemo
        je u desno podstablo */
        if (strcmp(rec, (*adresa_korena)->rec) > 0)
83            dodaj_u_stablo(&(*adresa_korena)->desno, rec);

85  else
        /* Ako je rec jednaka reci u korenu, uvecavamo njen broj

```

```

87         pojavljivanja */
      (*adresa_korena)->brojac++;
89   }

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

97     /* Inace ... */
99     /* Oslobadjamo memoriju zauzetu levim podstablom */
     oslobodi_stablo(&(*adresa_korena)->levo);

101    /* Oslobadjamo memoriju zauzetu desnim podstablom */
103    oslobodi_stablo(&(*adresa_korena)->desno);

105    /* Oslobadjamo memoriju zauzetu korenom */
     free((*adresa_korena)->rec);
107     free(*adresa_korena);

109     /* Proglasavamo stablo praznim */
     *adresa_korena = NULL;
111   }

113   /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec
115     (rec sa najvećim brojem pojavljivanja) */
   Cvor *nadj_i_najfrekventniju_rec(Cvor * koren)
117   {
      Cvor *max, *max_levo, *max_desno;

119     /* Ako je stablo prazno, prekidamo sa pretragom */
121     if (koren == NULL)
        return NULL;

123     /* Pronalazimo najfrekventniju reci u levom podstablu */
125     max_levo = nadj_i_najfrekventniju_rec(koren->levo);

127     /* Pronalazimo najfrekventniju reci u desnom podstablu */
     max_desno = nadj_i_najfrekventniju_rec(koren->desno);

129     /* Trazimo maksimum vrednosti pojavljivanja reci iz levog
131       podstabla, korena i desnog podstabla */
     max = koren;
133     if (max_levo != NULL && max_levo->brojac > max->brojac)
        max = max_levo;
135     if (max_desno != NULL && max_desno->brojac > max->brojac)
        max = max_desno;

137     /* Vracamo adresu cvora sa najvećim brojacem */

```

```
139     return max;
140 }
141
142 /* Funkcija koja ispisuje reci iz stabla u leksikografskom
143    poretku pracene brojem pojavljivanja */
144 void prikazi_stablo(Cvor * koren)
145 {
146     /* Ako je stablo prazno, završavamo sa ispisom */
147     if (koren == NULL)
148         return;
149
150     /* Zbog leksikografskog poretka, prvo ispisujemo sve reci iz
151        levog podstabla */
152     prikazi_stablo(koren->levo);
153
154     /* Zatim ispisujemo rec iz korena */
155     printf("%s: %d\n", koren->rec, koren->brojac);
156
157     /* I nastavljamo sa ispisom reci iz desnog podstabla */
158     prikazi_stablo(koren->desno);
159 }
160
161
162 /* Funkcija ucitava sledecu rec iz zadate datoteke i upisuje je
163    u niz rec. Maksimalna duzina reci je odredjena argumentom
164    max. Funkcija vraca EOF ako nema vise reci ili 0 u suprotnom.
165    Rec je niz malih ili velikih slova. */
166 int procitaj_rec(FILE * f, char rec[], int max)
167 {
168     /* karakter koji citamo */
169     int c;
170
171     /* indeks pozicije na koju se smesta procitani karakter */
172     int i = 0;
173
174     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle god
175        nismo stigli do kraja datoteke... */
176     while (i < max - 1 && (c = fgetc(f)) != EOF) {
177         /* Proveravamo da li je procitani karakter slovo */
178         if (isalpha(c))
179
180             /* Ako jeste, smestamo ga u niz - pritom vrsimo konverziju
181                u mala slova jer program treba da bude neosetljiv na
182                razliku izmedju malih i velikih slova */
183             rec[i++] = tolower(c);
184
185         else
186             /* Ako nije, proveravamo da li smo procitali barem jedno
187                slovo nove rece */
188             /* Ako jesmo prekidamo sa citanjem */
189             if (i > 0)
```

```
191     break;

193     /* U suprotnom idemo na sledecu iteraciju */
194 }

195     /* Dodajemo na rec terminirajucu nulu */
197     rec[i] = '\0';

199     /* Vracamo 0 ako smo procitali rec, EOF u suprotnom */
200     return i > 0 ? 0 : EOF;
201 }

203 int main(int argc, char **argv)
204 {
205     Cvor *koren = NULL, *max;
206     FILE *f;
207     char rec[MAX];

209     /* Proveravamo da li je navedeno ime datoteke prilikom
210        pokretanja programa */
211     if (argc < 2) {
212         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
213         exit(EXIT_FAILURE);
214     }

215     /* Otvaramo datoteku iz koje citamo reci */
217     if ((f = fopen(argv[1], "r")) == NULL) {
218         fprintf(stderr, "fopen() greska pri otvaranju %s\n",
219             argv[1]);
220         exit(EXIT_FAILURE);
221     }

223     /* Ucitavamo reci iz datoteke i smestamo u binarno stablo
224        pretrage. */
225     while (procitaj_rec(f, rec, MAX) != EOF)
226         dodaj_u_stablo(&koren, rec);

227     /* Posto smo zavrшили sa citanjem reci zatvaramo datoteku */
229     fclose(f);

231     /* Prikazujemo sve reci iz teksta i brojeve njihovih
232        pojavljivanja. */
233     prikazi_stablo(koren);

235     /* Pronalazimo najfrekventniju rec */
236     max = nadji_najfrekventniju_rec(koren);

237     /* Ako takve reci nema... */
239     if (max == NULL)

241         /* Ispisujemo odgovarajuće obavestenje */
242         printf("U tekstu nema reci!\n");
```

```
243
else
245     /* Inace, ispisujemo broj pojavljivanja reci */
    printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
247         max->rec, max->brojac);

249     /* Oslobadjamo dinamicki alociran prostor za stablo */
    oslobodi_stablo(&koren);

251     /* Završavamo sa programom */
253     return 0;
}
```

### Rešenje 4.16

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

6 #define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
8 #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
    broj telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
    char ime_i_prezime[MAX_IME_I_PREZIME];
14     char telefon[MAX_CIFARA];
    struct cvor *levo;
16     struct cvor *desno;
} Cvor;

18

/* Funkcija koja kreira novi cvor stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
22     /* Alociramo memoriju za novi cvor */
    Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
24     if (novi_cvor == NULL)
        return NULL;

26     /* Inicijalizujemo polja u novom cvoru */
28     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
    strcpy(novi_cvor->telefon, telefon);
30     novi_cvor->levo = NULL;
    novi_cvor->desno = NULL;

32     /* Vracamo adresu novog cvora */
34     return novi_cvor;
}

36
```

```
38 /* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
40 void prover_i_alokaciju(Cvor * novi_cvor)
41 {
42     /* Ukoliko je cvor neuspesno kreiran */
43     if (novi_cvor == NULL) {
44         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
           programa */
45         fprintf(stderr, "Malloc greska za novi cvor!\n");
46         exit(EXIT_FAILURE);
47     }
48 }
49
50
51
52 /* Funkcija koja dodaje novu osobu i njen broj telefona u
   stablo. */
53 void
54 dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
55               char *telefon)
56 {
57     /* Ako je stablo prazno */
58     if (*adresa_korena == NULL) {
59         /* Kreiramo novi cvor */
60         Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
61         prover_i_alokaciju(novi);
62
63         /* i proglašavamo ga korenom stabla */
64         *adresa_korena = novi;
65         return;
66     }
67
68     /* U suprotnom trazimo odgovarajucu poziciju za novi unos */
69     /* Kako pretragu treba vrsiti po imenu i prezimenu, stablo
       treba da bude pretrazivacko po ovom polju */
70     /* Ako je zadato ime i prezime leksikografski manje od imena i
       prezimena sadržanog u korenu, podatke dodajemo u levo
       podstablo */
71     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
72         < 0)
73         dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
74                       telefon);
75
76     else
77         /* Ako je zadato ime i prezime leksikografski vece od imena
           i prezimena sadržanog u korenu, podatke kodajemo u desno
           podstablo */
78         if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
79             dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
80                           telefon);
81 }
82
83
84
85
86
87
88
```

```
90 /* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
92 {
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
94     if (*adresa_korena == NULL)
        return;
96
    /* Inace ... */
98     /* Oslobadjamo memoriju zauzetu levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);
100
    /* Oslobadjamo memoriju zauzetu desnim podstablom */
102     oslobodi_stablo(&(*adresa_korena)->desno);
104
    /* Oslobadjamo memoriju zauzetu koren timerom */
    free(*adresa_korena);
106
    /* Proglasavamo stablo praznim */
108     *adresa_korena = NULL;
}
110

112 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
/* Napomena: ova funkcija nije trazena u zadatku ali se moze
114 koristiti za proveru da li je stablo lepo kreirano ili ne */
void prikazi_stablo(Cvor * koren)
116 {
    /* Ako je stablo prazno, završavamo sa ispisom */
118     if (koren == NULL)
        return;
120
    /* Zbog leksikografskog poretka, prvo ispisujemo podatke iz
122     levog podstabla */
    prikazi_stablo(koren->levo);
124
    /* Zatim ispisujemo podatke iz korena */
126     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
128
    /* I nastavljamo sa ispisom podataka iz desnog podstabla */
    prikazi_stablo(koren->desno);
130 }
132

134 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
    ime i prezime i broj telefona u odgovarajuće nizove.
    Maksimalna dužina imena i prezimena određena je konstantom
136 MAX_IME_PREZIME, a maksimalna dužina broja telefona
    konstantom MAX_CIFARA. Funkcija vraća EOF ako nema više
    kontakata ili 0 u suprotnom. */
138 int procitaj_kontakt(FILE * f, char *ime_i_prezime,
140                     char *telefon)
```



```

142 {
143     int c;
144     int i = 0;
145
146     /* Linije datoteke koje obradjujemo su formata Ime Prezime
147        BrojTelefona */
148     /* Preskacemo eventualne praznine sa pocetka linije datoteke */
149     while ((c = fgetc(f)) != EOF && isspace(c));
150
151     /* Prvo procitano slovo upisujemo u ime i prezime */
152     if (!feof(f))
153         ime_i_prezime[i++] = c;
154
155     /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
156        cifre, tako da cemo citanje forsirati sve dok ne naidjemo
157        na cifru. Pri tom cemo voditi racuna da li ima dovoljno
158        mesta za smestanje procitanog karaktera i da slucajno ne
159        dodjemo do kraja datoteke */
160     while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
161         if (!isdigit(c))
162             ime_i_prezime[i++] = c;
163
164         else if (i > 0)
165             break;
166     }
167
168     /* Upisujemo terminirajucu nulu na mesto poslednjeg procitanog
169        blanko karaktera */
170     ime_i_prezime[--i] = '\0';
171
172     /* I pocinjemo sa citanjem broja telefona */
173     i = 0;
174
175     /* Upisujemo cifru koju smo vec procitali */
176     telefon[i++] = c;
177
178     /* I citamo peostale cifre - naznaka kraja ce nam biti pojava
179        karaktera cije prisustvo nije dozvoljeno u broju telefona */
180     while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
181         if (c == '/' || c == '-' || isdigit(c))
182             telefon[i++] = c;
183
184         else
185             break;
186
187     /* Upisujemo terminirajucu nulu */
188     telefon[i] = '\0';
189
190     /* Vracamo 0 ako smo procitali kontakt, EOF u suprotnom */
191     return !feof(f) ? 0 : EOF;
192 }

```

```
194 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
    prezimenom */
196 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
{
198     /* Ako je imenik prazan, završavamo sa pretragom */
    if (koren == NULL)
200         return NULL;

202     /* Ako je trazeno ime i prezime sadržano u korenu, takodje
        završavamo sa pretragom */
204     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
        return koren;

206     /* Ako je zadato ime i prezime leksikografski manje od
        vrednosti u korenu pretragu nastavljamo levo */
208     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
210         return pretrazi_imenik(koren->levo, ime_i_prezime);

212     else
        /* u suprotnom, pretragu nastavljamo desno */
214         return pretrazi_imenik(koren->desno, ime_i_prezime);
}

216 int main(int argc, char **argv)
{
218     char ime_datoteke[MAX_IME_DATOTEKE];
220     Cvor *koren = NULL;
    Cvor *trazeni;
222     FILE *f;
    char ime_i_prezime[MAX_IME_I_PREZIME];
224     char telefon[MAX_CIFARA];
    char c;
226     int i;

228     /* Učitavamo ime datoteke i pripremamo je za citanje */
    printf("Unesite ime datoteke: ");
230     scanf("%s", ime_datoteke);
    if ((f = fopen(ime_datoteke, "r")) == NULL) {
232         fprintf(stderr, "fopen() greska prilikom otvaranja
        %s\n", ime_datoteke);
234         exit(EXIT_FAILURE);
    }

236     /* Učitavamo podatke iz datoteke i smestamo kontakte u binarno
        stablo pretrage. */
238     while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
240         dodaj_u_stablo(&koren, ime_i_prezime, telefon);

242     /* Posto smo završili sa citanjem podataka zatvaramo datoteku */
    fclose(f);
244 }
```

```

246  /* Omogucavamo pretragu imenika */
    while (1) {
248      /* Ucitavamo ime i prezime */
      printf("Unesite ime i prezime: ");
      i = 0;
250      while ((c = getchar()) != '\n')
        ime_i_prezime[i++] = c;
252      ime_i_prezime[i] = '\0';

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

258      /* Inace, ispisujemo rezultat pretrage */
260      trazen_i = pretrazi_imenik(koren, ime_i_prezime);
      if (trazen_i == NULL)
262        printf("Broj nije u imeniku!\n");

264      else
        printf("Broj je: %s \n", trazen_i->telefon);
266    }

268    /* Oslobadjamo memoriju zauzetu imenikom */
    oslobodi_stablo(&koren);

270    /* Završavamo sa programom */
272    return 0;
}

```

### Rešenje 4.17

```

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

4
#define MAX 51

6
/* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8  studenta, ukupan uspsih, uspeh iz matematike, uspeh iz
   maternjeg jezika i redom pokazivace na levo i desno podstablo
10 */
typedef struct cvor_stabla {
12   char ime[MAX];
   char prezime[MAX];
14   double uspeh;
   double matematika;
16   double jezik;
   struct cvor_stabla *levo;
18   struct cvor_stabla *desno;
} Cvor;

```

```
20
21 /* Funkcija kojom se kreira cvor stabla */
22 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
23                   double matematika, double jezik)
24 {
25     /* Alociramo memoriju za novi cvor */
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;
29
30     /* Inicijalizujemo 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     /* Vracamo adresu kreiranog cvora */
40     return novi;
41 }
42
43 /* Funkcija kojom se proverava uspesnost alociranja memorije */
44 void proveri_alokaciju(Cvor * novi_cvor)
45 {
46     /* Ako alokacije nije uspesna */
47     if (novi_cvor == NULL) {
48         /* Ispisujemo poruku i prekidamo sa izvršavanjem */
49         fprintf(stderr, "Malloc greska za novi cvor!\n");
50         exit(EXIT_FAILURE);
51     }
52 }
53
54 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
55 void oslobodi_stablo(Cvor ** koren)
56 {
57     /* Ako je stablo prazno, nema potrebe za oslobadjanjem
58        memorije */
59     if (*koren == NULL)
60         return;
61
62     /* oslobadjamo memoriju zauzetu levim podstablom */
63     oslobodi_stablo(&(*koren)->levo);
64
65     /* oslobadjamo memoriju zauzetu desnim podstablom */
66     oslobodi_stablo(&(*koren)->desno);
67
68     /* oslobadjamo memoriju zauzetu korenom */
69     free(*koren);
70 }
```

```

72  /* proglašavamo stablo praznim */
    *koren = NULL;
74  }

76  /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
78                  double uspeh, double matematika,
                      double jezik)
80  {
    /* Ako je stablo prazno */
82    if (*koren == NULL) {
        /* Kreiramo novi cvor */
84        Cvor *novi =
            napravi_cvor(ime, prezime, uspeh, matematika, jezik);
86        proveri_alokaciju(novi);

88        /* I proglašavamo ga korenom stabla */
        *koren = novi;
90
92        return;
    }

94    /* Inace, dodajemo cvor u stablo tako da bude sortiran po
        ukupnom broju poena */
96    if (uspeh + matematika + jezik >
        (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
98        dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
                        matematika, jezik);
100    else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
102                      matematika, jezik);
104    }

106  /* Funkcija ispisuje sadrzaj stabla - ukoliko je vrednost
        argumenta polozili jednaka 0 ispisuju se informacije o
108    uenicima koji nisu polozili prijemni, a ako je vrednost
        argumenta razlicita od nule, ispisuju se informacije o
110    uenicima koji su polozili prijemni */
void stampa_j(Cvor * koren, int polozili)
112  {
    /* Stablo je prazno - prekidamo sa ispisom */
114    if (koren == NULL)
        return;
116
    /* Stampamo informacije iz levog podstabla */
118    stampa_j(koren->levo, polozili);

    /* Stampamo informacije iz korenog cvora */
120    if (polozili && koren->matematika + koren->jezik >= 10)
122        printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
                koren->prezime, koren->uspeh, koren->matematika,

```

```

124         koren->jezik,
            koren->uspeh + koren->matematika + koren->jezik);
126     else if (!polozili && koren->matematika + koren->jezik < 10)
        printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
128            koren->prezime, koren->uspeh, koren->matematika,
            koren->jezik,
130            koren->uspeh + koren->matematika + koren->jezik);

132     /* Stampamo informacije iz desnog podstabla */
    stampaj(koren->desno, polozili);
134 }

136
138 /* Funkcija koja odredjuje koliko studenata nije polozilo
    prijemni ispit */
140 int nisu_polozili(Cvor * koren)
{
142     /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
    if (koren == NULL)
        return 0;

144     /* Pretragu vrsimo i u levom i u desnom podstablu - ako uslov
        za polaganje nije ispunjen za koreni cvor, broj studenata
        uvecavamo za 1 */
146     if (koren->matematika + koren->jezik < 10)
        return 1 + nisu_polozili(koren->levo) +
150            nisu_polozili(koren->desno);

152     return nisu_polozili(koren->levo) +
        nisu_polozili(koren->desno);
154 }

156 int main(int argc, char **argv)
{
158     FILE *in;
    Cvor *koren;
160     char ime[MAX], prezime[MAX];
    double uspeh, matematika, jezik;

162     /* Otvaramo datoteku sa rezultatima sa prijemnog za citanje */
    in = fopen("prijemni.txt", "r");
164     if (in == NULL) {
        fprintf(stderr, "Greska prilikom citanja podataka!\n");
        exit(EXIT_FAILURE);
166     }

168 }

170 /* Citamo podatke i dodajemo ih u stablo */
    koren = NULL;
172     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
        &matematika, &jezik) != EOF) {
174         dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
            jezik);

```

```

176     }

178     /* Zatvaramo datoteku */
    fclose(in);

180     /* Stampamo prvo podatke o ucenicima koji su polozili prijemni
182     */
    stampaj(koren, 1);

184     /* Liniju iscrtavamo samo ako postoje učenici koji nisu
186     polozili prijemni */
    if (nisu_polozili(koren) != 0)
188         printf("-----\n");

190     /* Stampamo podatke o ucenicima koji nisu polozili prijemni */
    stampaj(koren, 0);

192     /* Oslobadjamo memoriju zauzetu stablom */
194     oslobodi_stablo(&koren);

196     /* Završavamo sa programom */
    return 0;

198 }

```

### Rešenje 4.18

```

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

5  #define MAX_NISKA 51
   #define MAX_DATUM 3

7

9  /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i
   prezime osobe, dan, mesec i godinu rođenja i redom
   pokazivace na levo i desno podstablo */
11 typedef struct cvor_stabla {
    char ime[MAX_NISKA];
13    char prezime[MAX_NISKA];
    int dan;
15    int mesec;
    int godina;
17    struct cvor_stabla *levo;
    struct cvor_stabla *desno;
19 } Cvor;

21 /* Funkcija koja kreira novi cvor */
Cvor *napravi_cvor(char ime[], char prezime[], int dan,
23                  int mesec, int godina)
{
25     /* Alociramo memoriju */

```

```
27 Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;
29
    /* Inicijalizujemo polja strukture */
31    strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
33    novi->dan = dan;
    novi->mesec = mesec;
35    novi->godina = godina;
    novi->levo = NULL;
37    novi->desno = NULL;
39
    /* Vracamo adresu novog cvora */
    return novi;
41 }
43
/* Funkcija koja proverava uspesnost alokacije */
void proveri_alokaciju(Cvor * novi_cvor)
45 {
    /* Ako memorija nije uspesno alocirana */
47    if (novi_cvor == NULL) {
        /* Ispisujemo poruku i prekidamo izvršavanje programa */
49        fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
51    }
}
53
/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** koren)
55 {
    /* Stablo je prazno */
57    if (*koren == NULL)
59        return;
61
    /* Oslobadjamo memoriju zauzetu levim podstablom (ako postoji)
    */
63    if ((*koren)->levo)
        oslobodi_stablo(&(*koren)->levo);
65
    /* Oslobadjamo memoriju zauzetu desnim podstablom (ako
    postoji) */
67    if ((*koren)->desno)
69        oslobodi_stablo(&(*koren)->desno);
71
    /* Oslobadjamo memoriju zauzetu korenom */
    free(*koren);
73
    /* Proglasavamo stablo praznim */
75    *koren = NULL;
    }
77
```



```

79  /* Funkcija koja dodaje novi cvor u stablo - stablo treba da
    bude uredjeno po datumu */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
81                      int dan, int mesec, int godina)
{
83  /* Ako je stablo prazno */
    if (*koren == NULL) {
85
86      /* Kreiramo novi cvor */
87      Cvor *novi_cvor =
            napravi_cvor(ime, prezime, dan, mesec, godina);
89      prover_i_alokaciju(novi_cvor);

91      /* I proglašavamo ga korenom */
      *koren = novi_cvor;
93
94      return;
95  }

97  /* Kako se ne unosi godina za pretragu, stablo uredjujemo samo
    po mesecu (i danu u okviru istog meseca) */
99  if (mesec < (*koren)->mesec)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
101                      godina);
    else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
103        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
                        godina);
    else
105        dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec,
                        godina);
107 }

109 /* Funkcija vrši pretragu stabla i vraća cvor sa traženim
    datumom (null ako takav ne postoji). u promenljivu pom ce
    biti smesten prvi datum (dan i mesec) veci od traženog datuma
    (null ako takav ne postoji)

111 */
Cvor *pretrazi(Cvor * koren, int dan, int mesec)
117 {
    /* Stablo je prazno, obustavljamo pretragu */
119    if (koren == NULL)
        return NULL;

121
    /* Nasli smo traženi datum u stablu */
123    if (koren->dan == dan && koren->mesec == mesec)
        return koren;

125
    /* Ako je mesec traženog datuma manji od meseca sadržanog u
    korenu ili ako su meseci isti ali je dan traženog datuma
    manji od aktuelnog datuma, pretražujemo levo podstablo -
    pre toga svakako proveravamo da li leva grana postoji - ako
127
129

```

```
131     ne postoji treba da vratimo prvi sledeci, a to je bas
    vrednost uocenog korena */
133     if (mesec < koren->mesec
        || (mesec == koren->mesec && dan < koren->dan)) {
135         if (koren->levo == NULL)
            return koren;
        else
137             return pretrazi(koren->levo, dan, mesec);
    }

139     /* inace, nastavljamo pretragu u desnom delu */
141     return pretrazi(koren->desno, dan, mesec);
143 }

145 int main(int argc, char **argv)
146 {
147     FILE *in;
148     Cvor *koren;
149     Cvor *slavljenik;
150     char ime[MAX_NISKA], prezime[MAX_NISKA];
151     int dan, mesec, godina;

153     /* Proveravamo da li je zadato ime ulazne datoteke */
154     if (argc < 2) {
155         /* Ako nije, ispisujemo poruku i prekidamo sa izvršavanjem
            programa */
156         printf("Nedostaje ime ulazne datoteke!\n");
157         return 0;
158     }

159     /* Inace, pripremamo datoteku za citanje */
160     in = fopen(argv[1], "r");
161     if (in == NULL) {
162         fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
163         exit(EXIT_FAILURE);
164     }

165     /* I popunjavamo podacima stablo */
166     koren = NULL;
167     while (fscanf
168         (in, "%s %s %d.%d.%d.", ime, prezime, &dan, &mesec,
169          &godina) != EOF)
170         dodaj_u_stablo(&koren, ime, prezime, dan, mesec, godina);

171     /* I zatvaramo datoteku */
172     fclose(in);

173     /* Omogucavamo pretragu podataka */
174     while (1) {
175
176         /* Ucitavamo novi datum */
177     }
```

```

183     printf("Unesite datum: ");
184     if (scanf("%d.%d.", &dan, &mesec) == EOF)
185         break;
186
187     /* Pretražujemo stablo */
188     slavljenik = pretrazi(koren, dan, mesec);
189
190     /* Ispisujemo pronadjene podatke */
191     if (slavljenik == NULL) {
192         printf("Nema podataka o ovim ni o sledecem rođjendanu.\n");
193         continue;
194     }
195
196     /* Slučaj kada smo pronasli prave podatke */
197     if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
198         printf("Slavljenik: %s %s\n", slavljenik->ime,
199             slavljenik->prezime);
200         continue;
201     }
202
203     /* Slučaj kada smo pronasli podatke o prvom sledecem
204        rođjendanu */
205     printf("Slavljenik: %s %s %d.%d.\n", slavljenik->ime,
206         slavljenik->prezime, slavljenik->dan,
207         slavljenik->mesec);
208 }
209
210 /* Oslobadjamo memoriju zauzetu stablom */
211 oslobodi_stablo(&koren);
212
213 /* Prekidamo sa izvršavanjem programa */
214 return 0;
215 }

```

### Rešenje 4.19

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura kojom se predstavlja cvor binarnog pretraživackog
5     stabla */
6  typedef struct cvor {
7     int broj;
8     struct cvor *levo, *desno;
9 } Cvor;
10
11 /* Funkcija koja alokira memoriju za novi cvor stabla,
12    inicijalizuje polja strukture i vraca pokazivac na novi cvor */
13 Cvor *napravi_cvor(int broj);
14
15 /* Funkcija koja proverava uspesnost kreiranja novog cvora

```

```
16     stabla. */
17 void proveri_alokaciju(Cvor * novi_cvor);
18
19 /* Funkcija koja dodaje zadati broj u stablo */
20 void dodaj_u_stablo(Cvor ** adresa_korena, int broj);
21
22 /* Funkcija koja proverava da li se zadati broj nalazi u stablu */
23 Cvor *pretrazi_stablo(Cvor * koren, int broj);
24
25 /* Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
26    stablu */
27 Cvor *pronadji_najmanji(Cvor * koren);
28
29 /* Funkcija koja pronalazi cvor koji sadrzi najveći vrednost u
30    stablu */
31 Cvor *pronadji_najveci(Cvor * koren);
32
33 /* Funkcija koja brise cvor stabla koji sadrzi zadati broj */
34 void obrisi_element(Cvor ** adresa_korena, int broj);
35
36 /* Funkcija koja ispisuje sadržaj stabla u infiksnoj notaciji
37    (levo podstablo - koren - desno podstablo) */
38 void prikazi_stablo(Cvor * koren);
39
40 /* Funkcija koja oslobadja memoriju zauzetu stablom */
41 void oslobodi_stablo(Cvor ** adresa_korena);
42
43 #endif
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"
4
5  /* Funkcija kojom se kreira novi cvor stabla koji sadrzi zadatu
6     vrednost */
7  Cvor *napravi_cvor(int broj)
8  {
9      /* Alociramo memoriju za novi cvor */
10     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
11     if (novi == NULL)
12         return NULL;
13     /* Inicijalizujemo polja cvora */
14     novi->broj = broj;
15     novi->levo = NULL;
16     novi->desno = NULL;
17     /* Vracamo adresu novog cvora */
18     return novi;
19 }
20
21 /* Funkcija koja proverava uspesnost kreiranja novog cvora
22    stabla */
23 void proveri_alokaciju(Cvor * novi_cvor)
```

```

{
25  /* Ukoliko je cvor neuspesno kreiran */
   if (novi_cvor == NULL) {
27      /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
        programa */
29      fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
31  }
   }

33

/* Funkcija koja dodaje novi broj u stablo. */
35 void dodaj_u_stablo(Cvor ** koren, int broj)
{
37     /* Ako je stablo prazno */
   if (*koren == NULL) {
39         /* Kreiramo novi cvor */
        Cvor *novi = napravi_cvor(broj);
41         prover_i_alokaciju(novi);
        /* i proglašavamo ga korenom stabla */
43         *koren = novi;
        return;
45     }
   /* U suprotnom trazimo odgovarajucu poziciju za novi broj */
47   /* Ako je broj manji od vrednosti sadrzane u korenu, ubacujemo
     ga u levo podstablo */
49   if (broj < (*koren)->broj)
       dodaj_u_stablo(&(*koren)->levo, broj);
51   else
       /* Inace, ubacujemo broj u desno podstablo */
53   dodaj_u_stablo(&(*koren)->desno, broj);
   }

55
/* Funkcija koja oslobadja memoriju zauzetu stablom */
57 void oslobodi_stablo(Cvor ** koren)
{
59     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
   if (*koren == NULL)
61         return;
   /* Inace ... */
63   /* Oslobadjamo memoriju zauzetu levom podstablom */
   if ((*koren)->levo)
65       oslobodi_stablo(&(*koren)->levo);
   /* Oslobadjamo memoriju zauzetu desnom podstablom */
67   if ((*koren)->desno)
       oslobodi_stablo(&(*koren)->desno);
69   /* Oslobadjamo memoriju zauzetu korenom */
   free(*koren);
71   /* Proglašavamo stablo praznim */
   *koren = NULL;
73 }

75 Cvor *pronadji_najmanji(Cvor * koren)

```

```

{
77  /* ako je stablo prazno, prekidamo pretragu */
   if (koren == NULL)
79     return NULL;
   /* vrednosti koje su manje od vrednosti u korenu stabla nalaze
81    se levo od njega */
   /* ako je koren cvor koji nema levo podstablo, onda on sadrzi
83     najmanju vrednost */
   if (koren->levo == NULL)
85     return koren;
   /* inace, pretragu treba nastaviti u levom podstablu */
87   return pronadji_najmanji(koren->levo);
}

89 Cvor *pronadji_najveci(Cvor * koren)
90 {
91   /* ako je stablo prazno, prekidamo pretragu */
92   if (koren == NULL)
93     return NULL;
94   /* vrednosti koje su vece od vrednosti u korenu stabla nalaze
95    se desno od njega */
96   /* ako je koren cvor koji nema desno podstablo, onda on sadrzi
97     najveću vrednost */
98   if (koren->desno == NULL)
99     return koren;
100  /* inace, pretragu treba nastaviti u desnom podstablu */
101  return pronadji_najveci(koren->desno);
102 }

103
104 /* Funkcija brise element iz stabla ciji je broj upravo jednak
105    broju n. Funkcija azurira koren stabla u pozivajućoj
106    funkciji, jer u ovoj funkciji koren može biti promenjen u
107    funkciji. */
108 void obrisi_element(Cvor ** adresa_korena, int n)
109 {
110   Cvor *pomocni = NULL;
111   /* Izlaz iz rekurzije: ako je stablo prazno, nema sta da se
112    brise */
113   if (*adresa_korena == NULL)
114     return;
115   /* Ako je vrednost broja veca od vrednosti u korenu stabla,
116    tada se broj eventualno nalazi u desnom podstablu, pa treba
117    rekurzivno primeniti postupak na desno podstablo. Koren
118    ovako modifikovanog stabla je nepromenjen. */
119   if ((*adresa_korena)->broj < n) {
120     obrisi_element(&(*adresa_korena)->desno, n);
121     return;
122   }
123   /* Ako je vrednost broja manja od vrednosti korena, tada se
124    broj eventualno nalazi u levom podstablu, pa treba
125    rekurzivno primeniti postupak na levo podstablo. Koren
126    ovako modifikovanog stabla je nepromenjen. */

```

```

129     if ((*adresa_korena)->broj > n) {
        obrisi_element(&(*adresa_korena)->levo, n);
        return;
131     }
    /* Slede podslucajevi vezani za slucaj kada je vrednost u
133     korenu jednaka broju koji se brise (tj. slucaj kada treba
        obrisati koren) */
135     /* Ako koren nema sinova, tada se on prosto brise, i rezultat
        je prazno stablo (vracamo NULL) */
137     if ((*adresa_korena)->levo == NULL
        && (*adresa_korena)->desno == NULL) {
139         free(*adresa_korena);
        *adresa_korena = NULL;
141         return;
    }
143     /* Ako koren ima samo levog sina, tada se brisanje vrshi tako
        sto obrisemo koren, a novi koren postaje levo sin */
145     if ((*adresa_korena)->levo != NULL
        && (*adresa_korena)->desno == NULL) {
147         pomocni = (*adresa_korena)->levo;
        free(*adresa_korena);
149         *adresa_korena = pomocni;
        return;
151     }
    /* Ako koren ima samo desnog sina, tada se brisanje vrshi tako
153     sto obrisemo koren, a novi koren postaje desno sin */
    if ((*adresa_korena)->desno != NULL
155         && (*adresa_korena)->levo == NULL) {
        pomocni = (*adresa_korena)->desno;
157         free(*adresa_korena);
        *adresa_korena = pomocni;
159         return;
    }
161     /* Slucaj kada koren ima oba sina. Tada se brisanje vrshi na
        sledeci nacin: - najpre se potrazi sledbenik korena (u
163         smislu poretka) u stablu. To je upravo po vrednosti
        najmanji cvor u desnom podstablu. On se moze pronaci npr.
165         funkcijom pronadji_najmanji(). - Nakon toga se u koren
        smesti vrednost tog cvora, a u taj cvor se smesti vrednost
167         korena (tj. broj koji se brise). - Onda se prosto
        rekursivno pozove funkcija za brisanje na desno podstablo.
169         S obzirom da u njemu treba obrisati najmanji element, a on
        definitivno ima najvise jednog potomka, jasno je da ce
171         njegovo brisanje biti obavljeno na jedan od jednostavnijih
        nacina koji su gore opisani. */
173     pomocni = pronadji_najmanji((*adresa_korena)->desno);
    (*adresa_korena)->broj = pomocni->broj;
175     pomocni->broj = n;
    obrisi_element(&(*adresa_korena)->desno, n);
177 }

179 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje

```

```
    elemente u rastucem poretku) */
181 void prikazi_stablo(Cvor * koren)
182 {
183     /* izlaz iz rekurzije */
184     if (koren == NULL)
185         return;
186     prikazi_stablo(koren->levo);
187     printf("%d ", koren->broj);
188     prikazi_stablo(koren->desno);
189 }

191 Cvor *pretrazi_stablo(Cvor * koren, int broj)
192 {
193     /* ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
194     */
195     if (koren == NULL)
196         return NULL;
197     /* ako je trazena vrednost sadrazana u korenu */
198     if (koren->broj == broj) {
199         /* prekidamo pretragu */
200         return koren;
201     }
202     /* inace, ako je broj manji od vrednosti sadrzane u korenu */
203     if (broj < koren->broj)
204         /* pretragu nastavljamo u levom podstablu */
205         return pretrazi_stablo(koren->levo, broj);
206     else
207         /* u suprotnom, pretragu nastavljamo u desnom podstablu */
208         return pretrazi_stablo(koren->desno, broj);
209 }
```

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključujemo biblioteku za rad sa stablima - pogledati uvodni
5   zadatak ove glave */
6  #include "stabla.h"
7
8
9  /* Funkcija koja proverava da li su dva stabla koja sadrže cele
10   brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
11   odnosno 0 ako nisu */
12 int identitet(Cvor * koren1, Cvor * koren2)
13 {
14     /* Ako su oba stabla prazna, jednaka su */
15     if (koren1 == NULL && koren2 == NULL)
16         return 1;
17
18     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
19     jednaka */
20     if (koren1 == NULL || koren2 == NULL)
21         return 0;
```



```
23  /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
    vrednosti, mozemo da zakljucimo da se razlikuju */
25  if (koren1->broj != koren2->broj)
    return 0;
27
    /* inace, proveravamo da li vazi i jednakost u levih
    podstabala i desnih podstabala */
29  return (identitet(koren1->levo, koren2->levo)
    && identitet(koren1->desno, koren2->desno));
31 }
33
int main()
35 {
    int broj;
    Cvor *koren1, *koren2;
37
    koren1 = NULL;
    /* učitavamo elemente prvog stabla */
41  printf("Prvo stablo: ");
    scanf("%d", &broj);
43  while (broj != 0) {
        dodaj_u_stablo(&koren1, broj);
45  scanf("%d", &broj);
    }
47
    koren2 = NULL;
    /* učitavamo elemente drugog stabla */
49  printf("Drugo stablo: ");
    scanf("%d", &broj);
51  while (broj != 0) {
        dodaj_u_stablo(&koren2, broj);
53  scanf("%d", &broj);
    }
55
57  /* pozivamo funkciju koja ispituje identitet stabala */
    if (identitet(koren1, koren2))
59  printf("Stabla jesu identicna.\n");
    else
61  printf("Stabla nisu identicna.\n");
63
    /* oslobadjamo memoriju zauzetu stablima */
    oslobodi_stablo(&koren1);
65  oslobodi_stablo(&koren2);
67
    /* završavamo sa radom programa */
    return 0;
69 }
```

## Rešenje 4.20

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Uklucujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija kreira novo stablo identicno stablu koje je dato
8   korenom. */
9  void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
10 {
11     /* Izlaz iz rekurzije: ako je stablo prazno nema sta da se
12        kopira */
13     if (koren == NULL) {
14         *duplikat = NULL;
15         return;
16     }
17
18     /* Dupliramo koren stabla i postavljamo ga da bude koren novog
19        stabla */
20     *duplikat = napravi_cvor(koren->broj);
21     prover_i_alokaciju(*duplikat);
22
23     /* Rekurzivno dupliramo levo podstablo i njegovu adresu cuvamo
24        u pokazivacu na levo podstablo korena duplikata. */
25     kopiraj_stablo(koren->levo, &(*duplikat)->levo);
26
27     /* Rekurzivno dupliramo desno podstablo i njegovu adresu
28        cuvamo u pokazivacu na desno podstablo korena duplikata. */
29     kopiraj_stablo(koren->desno, &(*duplikat)->desno);
30 }
31
32 /* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
33    dobija modifikacijom prvog stabla */
34 void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
35 {
36     /* Ako drugo stablo nije prazno */
37     if (koren2 != NULL) {
38         /* dodajemo njegov koren u prvo stablo */
39         dodaj_u_stablo(adresa_korena1, koren2->broj);
40
41         /* rekurzivno racunamo uniju levog i desnog podstabla drugog
42            stabla sa prvim stablom */
43         kreiraj_uniju(adresa_korena1, koren2->levo);
44         kreiraj_uniju(adresa_korena1, koren2->desno);
45     }
46 }
47
48 /* Funkcija izracunava presek dva stabla - rezultujuce stablo se
49    dobija modifikacijom prvog stabla */
50 void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
51 {
```

```

52  /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
    */
54  if (*adresa_korena1 == NULL)
    return;
56
58  /* Kreiramo presek levog i desnog podstabla sa drugim stablom,
    tj. iz levog i desnog podstabla prvog stabla brisemo sve
    one elemente koji ne postoje u drugom stablu */
60  kreiraj_presek(&(*adresa_korena1)->levo, koren2);
    kreiraj_presek(&(*adresa_korena1)->desno, koren2);
62
64  /* Ako se koren prvog stabla ne nalazi u drugom stablu tada ga
    uklanjamo iz prvog stabla */
    if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
66        obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
    }
68
    /* Funkcija izracunava razliku dva stabla - rezultujuce stablo
    se dobija modifikacijom prvog stabla */
70  void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
72  {
74      /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
        */
        if (*adresa_korena1 == NULL)
76            return;
78
        /* Kreiramo razliku levog i desnog podstabla sa drugim
        stablom, tj. iz levog i desnog podstabla prvog stabla
        brisemo sve one elemente koji postoje i u drugom stablu */
80        kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
82        kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
84
        /* Ako se koren prvog stabla nalazi i u drugom stablu tada ga
        uklanjamo iz prvog stabla */
86        if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
            obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
88    }
90  int main()
    {
92      Cvor *koren1;
        Cvor *koren2;
94      Cvor *pomocni = NULL;
        int n;
96
        /* Ucitavamo elemente prvog stabla: */
98        koren1 = NULL;
        printf("Prvo stablo: ");
100        while (scanf("%d", &n) != EOF) {
            dodaj_u_stablo(&koren1, n);
102        }
    }

```

```
104  /* Ucitavamo elemente drugog stabla: */
    koren2 = NULL;
106  printf("Drugo stablo: ");
    while (scanf("%d", &n) != EOF) {
108      dodaj_u_stablo(&koren2, n);
    }

110
    /* Kreiramo uniju stabala: prvo napravimo kopiju prvog stabla
112     kako bi mogli da ga iskoristimo i za preostale operacije */
    kopiraj_stablo(koren1, &pomocni);
114     kreiraj_uniju(&pomocni, koren2);
    printf("Unija: ");
116     prikazi_stablo(pomocni);
    putchar('\n');

118
    /* Oslobadjamo stablo za rezultatom operacije */
120     oslobodi_stablo(&pomocni);

122
    /* Kreiramo presek stabala: prvo napravimo kopiju prvog stabla
124     kako bi mogli da ga iskoristimo i za preostale operacije; */
    kopiraj_stablo(koren1, &pomocni);
126     kreiraj_presek(&pomocni, koren2);
    printf("Presek: ");
128     prikazi_stablo(pomocni);
    putchar('\n');

130
    /* Oslobadjamo stablo za rezultatom operacije */
    oslobodi_stablo(&pomocni);

132
    /* Kreiramo razliku stabala: prvo napravimo kopiju prvog
134     stabla kako bi mogli da ga iskoristimo i za preostale
        operacije; */
136     kopiraj_stablo(koren1, &pomocni);
    kreiraj_razliku(&pomocni, koren2);
138     printf("Razlika: ");
    prikazi_stablo(pomocni);
140     putchar('\n');

142
    /* Oslobadjamo stablo za rezultatom operacije */
    oslobodi_stablo(&pomocni);

144
    /* Oslobadjamo i polazna stabla */
146     oslobodi_stablo(&koren2);
    oslobodi_stablo(&koren1);

148
    /* Završavamo sa programom */
150     return 0;
}
```

### Rešenje 4.21

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Uključujemo biblioteku 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
11  vrednosti koje su smestene u niz. */
   int kreiraj_niz(Cvor * koren, int a[])
13 {
   int r, s;
15
   /* Stablo je prazno - u niz je smesteno 0 elemenata */
17   if (koren == NULL)
       return 0;
19
   /* Dodajemo u niz elemente 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 mozemo odrediti indeks novog
25   elementa */
27
   /* Smestamo vrednost iz korena */
   a[r] = koren->broj;
29
   /* Dodajemo elemente iz desnog podstabla */
31   s = kreiraj_niz(koren->desno, a + r + 1);
33
   /* Racunamo indeks na koji treba smestiti naredni element */
   return r + s + 1;
35 }
37
   /* Funkcija sortira niz tako sto najpre elemente niza smesti u
      stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva
39   u desno.
41
      Ovaj nacin sortiranja primer sortiranja koje nije "u mestu "
      kao sto je to slucaj sa ostalim prethodno opisanim
43   algoritmima sortiranja, jer se sortiranje vrši u pomocnoj
      dinamičkoj strukturi, a ne razmenom elemenata niza. */
45 void sortiraj(int a[], int n)
   {
47     int i;
     Cvor *koren;
49
     /* Kreiramo stablo smestanjem elemenata iz niza u stablo */
51     koren = NULL;
```

```
53     for (i = 0; i < n; i++)
        dodaj_u_stablo(&koren, a[i]);

55     /* Infiksnim obilaskom stabla elemente iz stabla prepisujemo u
        niz a */
57     kreiraj_niz(koren, a);

59     /* Vise nam stablo nije potrebno i oslobadjamo memoriju */
        oslobodi_stablo(&koren);
61 }

63 int main()
{
65     int a[MAX];
        int n, i;

67     /* Ucitavamo dimenziju i elemente niza */
69     printf("n: ");
        scanf("%d", &n);
71     if (n < 0 || n > MAX) {
        printf("Greska: pogresna dimenzija niza!\n");
73         return 0;
    }

75     printf("a: ");
77     for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

79     /* Pozivamo funkciju za sortiranje */
81     sortiraj(a, n);

83     /* Ispisujemo rezultat */
85     for (i = 0; i < n; i++)
        printf("%d ", a[i]);
        printf("\n");

87     /* Prekidamo sa programom */
89     return 0;
}
```

### Rešenje 4.22

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

3
  /* Uključujemo biblioteku za rad sa stablima */
5 #include "stabla.h"

7 /* a) Funkcija koja izracunava broj cvorova stabla */
int broj_cvorova(Cvor * koren)
9 {
```

```
11  /* Ako je stablo prazno, broj cvorova je nula */
12  if (koren == NULL)
13      return 0;
14
15  /* U suprotnom je broj cvorova stabla jednak zbiru broja
16     cvorova u levom podstablu i broja cvorova u desnom
17     podstablu - 1 dodajemo zato sto treba racunati i koren */
18  return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) +
19      1;
20 }
21
22 /* b) Funkcija koja izracunava broj listova stabla */
23 int broj_listova(Cvor * koren)
24 {
25     /* Ako je stablo prazno, broj listova je nula */
26     if (koren == NULL)
27         return 0;
28
29     /* Proveravamo da li je tekuci cvor list */
30     if (koren->levo == NULL && koren->desno == NULL)
31         /* i ako jeste vracamo 1 - to ce kasnije zbog rekurzivnih
32            poziva uvecati broj listova za 1 */
33         return 1;
34
35     /* U suprotnom prebrojavamo listove koje se nalaze u
36        podstablama */
37     return broj_listova(koren->levo) + broj_listova(koren->desno);
38 }
39
40 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
41 void pozitivni_listovi(Cvor * koren)
42 {
43     /* Slucaj kada je stablo prazno */
44     if (koren == NULL)
45         return;
46
47     /* Ako je cvor list i sadrzi pozitivnu vrednost */
48     if (koren->levo == NULL && koren->desno == NULL
49         && koren->broj > 0)
50         /* Stampamo ga */
51         printf("%d ", koren->broj);
52
53     /* Nastavljamo sa stampanjem pozitivnih listova u podstablama */
54     pozitivni_listovi(koren->levo);
55     pozitivni_listovi(koren->desno);
56 }
57
58 /* d) Funkcija koja izracunava zbir cvorova stabla */
59 int zbir_cvorova(Cvor * koren)
60 {
61     /* Ako je stablo prazno, zbir cvorova je 0 */
62     if (koren == NULL)
```

```
        return 0;

63
    /* Inace, zbir cvorova stabla izracunavamo kao zbir korena i
65     svih elemenata u podstablina */
    return koren->broj + zbir_cvorova(koren->levo) +
67         zbir_cvorova(koren->desno);
}

69
/* e) Funckija koja izracunava najveći element stabla. */
71 Cvor *najveci_element(Cvor * koren)
{
73     /* Ako je stablo prazno, obustavljamo pretragu */
    if (koren == NULL)
75         return NULL;

77     /* Zbog prirode pretrazivackog stabla, sigurni smo da su
       vrednosti veće od korena u desnom podstablu */

79
    /* Ako desnog podstabla nema */
    if (koren->desno == NULL)
81         /* Najveća vrednost je koren */
83         return koren;

85     /* Inace, najveću vrednost trazimo jos desno */
    return najveci_element(koren->desno);
87 }

89 /* f) Funckija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
91 {
    /* Dubina praznog stabla je 0 */
93     if (koren == NULL)
        return 0;

95
    /* Izracunavamo dubinu levog podstabla */
97     int dubina_levo = dubina_stabla(koren->levo);

99     /* Izracunavamo dubinu desnog podstabla */
    int dubina_desno = dubina_stabla(koren->desno);

101
    /* dubina stabla odgovara vecoj od dubina podstabala - 1
103     dodajemo jer racunamo i koren */
    return dubina_levo >
105         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

107
/* g) Funckija koja izracunava broj cvorova na i-tom nivou */
109 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
{
111     /* ideja je da ste spustamo kroz stablo sve dok ne stignemo do
       trazenog nivoa */
113
```



```
115  /* Ako nema vise cvorova, ne mozemo da se spustamo niz stablo */
    if (koren == NULL)
        return 0;
117
    /* Ako smo stigli do trazenog nivoa, vracamo 1 - to ce kasnije
119     zbog rekurzivnih poziva uvecati broj pojavljivanja za 1 */
    if (i == 0)
121         return 1;

    /* inace, spustamo se jedan nivo nize i u levom i u desnom
123     postablu */
    return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
125         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
127 }

129 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispis_nivo(Cvor * koren, int i)
131 {
    /* ideja je slicna ideji iz prethodne funkcije */
133     /* nema vise cvorova, ne mozemo da se spustamo kroz stablo */
    if (koren == NULL)
135         return;

    /* ako smo na trazenom nivou - ispisujemo vrednost */
137     if (i == 0) {
        printf("%d ", koren->broj);
139         return;
    }
141
    /* inace, spustamo se jedan nivo nize i u levom i u desnom
143     podstablu */
    ispis_nivo(koren->levo, i - 1);
145     ispis_nivo(koren->desno, i - 1);
    }

147
149 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom
    nivou stabla */
Cvor *max_nivo(Cvor * koren, int i)
151 {
    /* Ako je stablo prazno, obustavljamo pretragu */
153     if (koren == NULL)
        return NULL;
155

    /* Ako smo na trazenom nivou, takodje prekidamo pretragu */
157     if (i == 0)
        return koren;
159

    /* Pronalazimo maksimum sa i-tog nivoa levog podstabla */
161     Cvor *a = max_nivo(koren->levo, i - 1);

    /* Pronalazimo maksimum sa i-tog nivoa desnog podstabla */
163     Cvor *b = max_nivo(koren->desno, i - 1);
165
```

```
167  /* Trazimo i vracamo maksimum izracunatih vrednosti */
169  if (a == NULL && b == NULL)
171      return NULL;
173  if (a == NULL)
175      return b;
177  if (b == NULL)
179      return a;
181  return a->broj > b->broj ? a : b;
183 }
185
187 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
189 int zbir_nivo(Cvor * koren, int i)
191 {
193     /* Ako je stablo prazno, zbir je nula */
195     if (koren == NULL)
197         return 0;
199
201     /* Ako smo na trazenom nivou, vracamo vrednost */
203     if (i == 0)
205         return koren->broj;
207
209     /* Inace, spustamo se jedan nivo nize i trazimo sume iz levog
211     i desnog podstabla */
213     return zbir_nivo(koren->levo, i - 1) + zbir_nivo(koren->desno,
215                                                         i - 1);
217 }
219
221 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje
223 su manje ili jednake od date vrednosti x */
225 int suma(Cvor * koren, int x)
227 {
229     /* Ako je stablo prazno, zbir je nula */
231     if (koren == NULL)
233         return 0;
235
237     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
239     prirode pretrazivackog stabla treba obici i levo i desno
241     podstablo */
243     if (koren->broj < x)
245         return koren->broj + suma(koren->levo,
247                                     x) + suma(koren->desno, x);
249
251     /* Inace, racunamo samo sumu vrednosti iz levog podstabla jer
253     medju njima jedino moze biti onih koje zadovoljavaju uslov */
255     return suma(koren->levo, x);
257 }
259
261 int main(int argc, char **argv)
263 {
265     /* Analiziramo argumente komandne linije */
267     if (argc != 3) {
```

```

219     fprintf(stderr,
        "Greska! Program se poziva sa: ./a.out nivo
        broj_zapretragu\n");
        exit(EXIT_FAILURE);
221     }
        int i = atoi(argv[1]);
223     int x = atoi(argv[2]);

225     /* Kreiramo stablo */
        Cvor *koren = NULL;
227     int broj;
        while (scanf("%d", &broj) != EOF)
229         dodaj_u_stablo(&koren, broj);

231     /* ispisujemo rezultat rada funkcija */
        printf("broj cvorova: %d\n", br_cvorova(koren));
233     printf("broj listova: %d\n", br_listova(koren));
        printf("pozitivni listovi: ");
235     pozitivni_listovi(koren);
        printf("zbir cvorova: %d\n", suma_cvorova(koren));
237

        if (najveci_element(koren) == NULL)
239             printf("najveci element: ne postoji\n");
        else
241             printf("najveci element: %d\n",
                najveci_element(koren)->broj);
243

        printf("dubina stabla: %d\n", dubina_stabla(koren));
245     printf("\n");
        printf("broj cvorova na %d. nivou: %d\n", i,
247             cvorovi_nivo(koren, i));
        printf("elementi na %d. nivou: ", i);
249     ispis_nivo(koren, i);
        printf("\n");
251     if (max_nivo(koren, i) == NULL)
            printf("Nema elemenata na %d. nivou!\n", i);
253     else
        printf("maksimalni na %d. nivou: %d\n", i,
255             max_nivo(koren, i)->broj);

257     printf("zbir na %d. nivou: %d\n", i, zbir_nivo(koren, i));
        printf("zbir elemenata manjih ili jednakih od %d: %d\n", x,
259             suma(koren, x));

261     /* Oslobadjamo memoriju zauzetu stablom */
        oslobodi_stablo(&koren);
263

        /* Prekidamo izvršavanje programa */
265     return 0;
    }

```

### Rešenje 4.23

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 /* Uključujemo biblioteku 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     /* Izracunavamo dubinu levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
16
17     /* Izracunavamo dubinu desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);
19
20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1
21        dodajemo jer racunamo i koren */
22     return dubina_levo >
23         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }
25
26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
27 void ispisi_nivo(Cvor * koren, int i)
28 {
29     /* Ideja je slicna ideji iz prethodne funkcije */
30
31     /* Nema vise cvorova, ne mozemo da se spustamo kroz stablo */
32     if (koren == NULL)
33         return;
34
35     /* Ako smo na trazenom nivou - ispisujemo vrednost */
36     if (i == 0) {
37         printf("%d ", koren->broj);
38         return;
39     }
40     /* Inace, spustamo se jedan nivo nize i u levom i u desnom
41        podstablu */
42     ispisi_nivo(koren->levo, i - 1);
43     ispisi_nivo(koren->desno, i - 1);
44 }
45
46 /* Funkcija koja ispisuje stablo po nivoima */
47 void ispisi_stablo_po_nivoima(Cvor * koren)
48 {
49     int i;
```

```

51  /* Prvo izracunavamo dubinu stabla */
    int dubina;
53  dubina = dubina_stabla(koren);

55  /* Ispisujemo nivo po nivo stabla */
    for (i = 0; i < dubina; i++) {
57      printf("%d. nivo: ", i);
        ispisi_nivo(koren, i);
59      printf("\n");
    }
61 }

63 int main(int argc, char **argv)
{
65     Cvor *koren;
    int broj;

67     /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
    koren = NULL;
69     while (scanf("%d", &broj) != EOF) {
71         dodaj_u_stablo(&koren, broj);
    }

73     /* Ispisujemo stablo po nivoima */
    ispisi_stablo_po_nivoima(koren);

75     /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);

77     /* Prekidamo izvršavanje programa */
79     return 0;
81 }

```

### Rešenje 4.24

### Rešenje 4.25

```

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

3

    /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  /* Funkcija koja izracunava dubinu stabla */
    int dubina_stabla(Cvor * koren)
9  {
    /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
        return 0;
13

```

```
15  /* Izracunavamo dubinu levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

17  /* Izracunavamo dubinu desnog podstabla */
    int dubina_desno = dubina_stabla(koren->desno);

19
    /* Dubina stabla odgovara vecoj od dubina podstabala - 1
    dodajemo jer racunamo i koren */
21    return dubina_levo >
23           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
    }

25  /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za
27     AVL stablo */
    int avl(Cvor * koren)
29  {
        int dubina_levo, dubina_desno;

31
        /* Ako je stablo prazno, zaustavljamo brojanje */
33        if (koren == NULL) {
            return 0;
35        }

37        /* Izracunavamo dubinu levog podstabla korena */
        dubina_levo = dubina_stabla(koren->levo);

39
        /* Izracunavamo dubinu desnog podstabla korena */
41        dubina_desno = dubina_stabla(koren->desno);

43
        /* Ako je uslov za AVL stablo ispunjen */
        if (abs(dubina_desno - dubina_levo) <= 1) {
45            /* Racunamo broj avl cvorova u levom i desnom podstablu i
            uvecavamo za jedan iz razloga sto koren ispunjava uslov */
47            return 1 + avl(koren->levo) + avl(koren->desno);
        } else {
49            /* Inace, racunamo 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;

59        /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
        koren = NULL;
61        while (scanf("%d", &broj) != EOF) {
            dodaj_u_stablo(&koren, broj);
63        }

65        /* Racunamo i ispisujemo broj AVL cvorova */
```

```

printf("%d\n", avl(koren));
67
/* Oslobadjamo memoriju zauzetu stablom */
69 oslobodi_stablo(&koren);

71 /* Prekidamo izvršavanje programa */
return 0;
73 }

```

### Rešenje 4.26

```

1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  /* Funkcija proverava da li je zadato binarno stablo celih
   pozitivnih brojeva heap. Ideja koju ćemo implementirati u
9  osnovi ima pronalazenje maksimalne vrednosti levog i
   maksimalne vrednosti desnog podstabla - ako je vrednost u
11 korenu veća od izračunatih vrednosti uoceni fragment stabla
   zadovoljava uslov za heap. Zato će funkcija vratiti
13 maksimalne vrednosti iz uocenog podstabala ili vrednost -1
   ukoliko zaključimo da stablo nije heap. */
15 int heap(Cvor * koren)
   {
17     int max_levo, max_desno;

19     /* Prazno sablo je heap. */
   if (koren == NULL) {
21         /* posto je 0 najmanji pozitivan broj, može nam poslužiti
           kao indikator */
23         return 0;

25     }
   /* Ukoliko je stablo list ... */
27     if (koren->levo == NULL && koren->desno == NULL) {
       /* ... vraćamo njegovu vrednost */
29         return koren->broj;
   }

31     /* Proveravamo svojstvo za levo podstablo. */
33     max_levo = heap(koren->levo);

35     /* Proveravamo svojstvo za desno podstablo. */
   max_desno = heap(koren->desno);
37     /* Ako levo ili desno podstablo uocenog cvora nije heap, onda
       nije ni celo stablo. */
39     if (max_levo == -1 || max_desno == -1) {
       return -1;
   }

```

```
41 }

43 /* U suprotnom proveravamo da li svojstvo vazii za uoceni
   cvor. */
45 if (koren->broj > max_levo && koren->broj > max_desno) {
46     /* ako vazii, vracamo vrednost korena */
47     return koren->broj;
48 }

49 /* u suprotnom zakljucujemo da stablo nije heap */
51 return -1;
52 }

53 int main(int argc, char **argv)
54 {
55     Cvor *koren;
56     int heap_indikator;

57     /* Kreiramo stablo koje sadrzi brojeve 100 19 36 17 3 25 1 2 7
       */
61     koren = NULL;
62     koren = napravi_cvor(100);
63     koren->levo = napravi_cvor(19);
64     koren->levo->levo = napravi_cvor(17);
65     koren->levo->levo->levo = napravi_cvor(2);
66     koren->levo->levo->desno = napravi_cvor(7);
67     koren->levo->desno = napravi_cvor(3);
68     koren->desno = napravi_cvor(36);
69     koren->desno->levo = napravi_cvor(25);
70     koren->desno->desno = napravi_cvor(1);

71     /* pozivamo funkciju kojom proveravamo da li je stablo heap */
72     heap_indikator = heap(koren);

73     /* i ispisujemo rezultat */
74     if (heap_indikator == -1) {
75         printf("Zadato stablo nije heap\n");
76     } else {
77         printf("Zadato stablo je heap!\n");
78     }

81     /* Oslobadjamo memoriju zauzetu stablom. */
82     oslobodi_stablo(&koren);

83     /* Završavamo sa programom */
84     return 0;
85 }
86 }
```

Rešenje 4.27



## Glava 5

# Ispitni rokovi

### 5.1 Programiranje 2, praktični deo ispita, jun 2015.

#### Zadatak 5.1

Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost -1 i prekinuti izvršavanje programa.

#### *Test 1*

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
Izlaz:
Ispit
Matematika
Programiranje
```

#### *Test 2*

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
2
maksimalano
poena
Izlaz:
```

### Test 3

```
Poziv: ./a.out ulaz.txt
Problem: datoteka
ulaz.txt ne postoji
Izlaz:
-1
```

### Test 4

```
Poziv: ./a.out
Izlaz:
-1
```

[Rešenje 5.1]

### Zadatak 5.2

Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na  $n$ -tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj  $n$ , a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

### Test 1

```
Ulaz:
2 8 10 3 6 14 13 7 4 0
Izlaz:
20
```

### Test 2

```
Ulaz:
0 50 14 5 2 4 56 8 52 7 1 0
Izlaz:
50
```

[Rešenje 5.2]

**Zadatak 5.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

<p><i>Test 1</i></p> <pre> Ulaz:  4 5  1 2 3 4 5 -1 2 -3 4 -5 -5 -4 -3 -2 1 -1 0 0 0 0 Izlaz: 0         </pre>	<p><i>Test 2</i></p> <pre> Ulaz:  2 3  0 0 -5  1 2 -4 Izlaz:  2         </pre>	<p><i>Test 3</i></p> <pre> Ulaz: -2 Izlaz (na stderr): -1         </pre>
--	--	--

[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.

<p><i>Test 1</i></p> <pre> Poziv: ./a.out ulaz.txt test ulaz.txt: Ovo je test primer.           U njemu se rec test javlja           vise puta. testtesttest Izlaz: 5         </pre>	<p><i>Test 2</i></p> <pre> Poziv: ./a.out Izlaz (na stderr): -1         </pre>
<p><i>Test 3</i></p> <pre> Poziv: ./a.out ulaz.txt foo ulaz.txt: (ne postoji) Izlaz (na stderr): -1         </pre>	<p><i>Test 4</i></p> <pre> Poziv: ./a.out ulaz.txt . ulaz.txt: (prazna) Izlaz: 0         </pre>

[Rešenje 5.4]

**Zadatak 5.5 Jelena:** Uključeno rešenje prethodnog zadatka. Dodati rešenje ovog zadatka. Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva

## 5 Ispitni rokovi

trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

<p><i>Test 1</i></p> <pre>   Datoteka: 4              0 0 0 1.2 1 0              0.3 0.3 0.5 0.5 0.9 1              -2 0 0 0 0 1              2 0 2 2 -1 -1    Izlaz:     2 0 2 2 -1 -1              -2 0 0 0 0 1              0 0 0 1.2 1 0              0.3 0.3 0.5 0.5 0.9 1</pre>	<p><i>Test 2</i></p> <pre>   Datoteka: 3              1.2 3.2 1.1 4.3    Izlaz:     -1</pre>
<p><i>Test 3</i></p> <pre>   Datoteka: (nema datoteke)    Izlaz:     -1</pre>	<p><i>Test 4</i></p> <pre>   Datoteka: 0    Izlaz:</pre>

[Rešenje 5.5]

**Zadatak 5.6** Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeba. Napisati funkciju

```
int f3(Cvor *koren, int n)
```

koja u datom stablu prebrojava čvorove na  $n$ -tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou 0. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

<p><i>Test 1</i></p> <pre>   Ulaz:    1 5 3 6 1 4 7 9    Izlaz:    1</pre>	<p><i>Test 2</i></p> <pre>   Ulaz:    2 5 3 6 1 0 4 7 9    Izlaz:    2</pre>	<p><i>Test 3</i></p> <pre>   Ulaz:    0 4 2 5    Izlaz:    0</pre>
<p><i>Test 4</i></p> <pre>   Ulaz:    3    Izlaz:    0</pre>	<p><i>Test 5</i></p> <pre>   Ulaz:    -1 4 5 1 7    Izlaz:    0</pre>	

[Rešenje 5.6]

## 5.3 Rešenja

### Rešenje 5.1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #define MAX 50
5
6  void greska()
7  {
8      printf("-1\n");
9      exit(EXIT_FAILURE);
10 }
11
12 int main(int argc, char *argv[])
13 {
14
15     FILE *ulaz;
16     char **linije;
17     int i, j, n;
18
19     /* Proveravamo argumente komandne linije. */
20     if (argc != 2) {
21         greska();
22     }
23
24     /* Otvaramo datoteku cije ime je navedeno kao argument
25        komandne linije neposredno nakon imena programa koji se
26        poziva. */
27     ulaz = fopen(argv[1], "r");
28     if (ulaz == NULL) {
29         greska();
30     }
31
32     /* Ucitavamo broj linija. */
33     fscanf(ulaz, "%d", &n);
34
35     /* Alociramo memoriju na osnovu ucitanog broja linija. */
36     linije = (char **) malloc(n * sizeof(char *));
37     if (linije == NULL) {
38         greska();
39     }
40     for (i = 0; i < n; i++) {
41         linije[i] = malloc(MAX * sizeof(char));
42         if (linije[i] == NULL) {
43             for (j = 0; j < i; j++) {
44                 free(linije[j]);
45             }
46             free(linije);
47         }
48     }
```

```
47     greska();
48     }
49 }

51 /* Ucitavamo svih n linija iz datoteke. */
52 for (i = 0; i < n; i++) {
53     fscanf(ulaz, "%s", linije[i]);
54 }

55
56 /* Ispisujemo u odgovarajucem poretку ucitane linije koje
57    zadovoljavaju kriterijum. */
58 for (i = n - 1; i >= 0; i--) {
59     if (isupper(linije[i][0])) {
60         printf("%s\n", linije[i]);
61     }
62 }

63
64 /* Oslobadjamo memoriju koju smo dinamicki alocirali. */
65 for (i = 0; i < n; i++) {
66     free(linije[i]);
67 }

68
69 free(linije);

71 /* Zatvaramo datoteku. */
72 fclose(ulaz);

73
74 /* Zavrsavamo sa programom. */
75 return 0;
76
77 }
```

### Rešenje 5.2

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

3
4
5  int sumirajN (Cvor * koren, int n){
6      if(koren==NULL){
7          return 0;
8      }

9
10     if(n==0){
11         return koren->broj;
12     }

13
14     return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
15 }
16
```

```

18 int main(){
    Cvor* koren=NULL;
20     int n;
    int nivo;

22     /* Citamo vrednost nivoa */
24     scanf("%d", &nivo);

26

    while(1){

28         /* Citamo broj sa standardnog ulaza */
30         scanf("%d", &n);

32         /* Ukoliko je korisnik uneo 0, prekidamo dalje citanje. */
        if(n==0){
34             break;
        }

36         /* A ako nije, dodajemo procitani broj u stablo. */
38         dodaj_u_stablo(&koren, n);

40     }

42     /* Ispisujemo rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren,nivo));

44     /* Oslobadjamo memoriju */
46     oslobodi_stablo(&koren);

48

    /* Prekidamo izvršavanje programa */
50     return 0;
}

```

```

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

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

9

    /* Inicijalizacija polja novog Cvora */
11    novi->broj = b;
    novi->levo = NULL;
13    novi->desno = NULL;

15    return novi;
17 }

```

```

19 void oslobodi_stablo(Cvor** adresa_korena) {
20     /* Prazno stablo i nema sta da se oslobadja */
21     if( *adresa_korena == NULL)
22         return;
23
24     /* Rekurzivno oslobadjamo najpre levo, a onda i desno podstablo*/
25     if( (*adresa_korena)->levo )
26         oslobodi_stablo(&(*adresa_korena)->levo);
27     if( (*adresa_korena)->desno)
28         oslobodi_stablo(&(*adresa_korena)->desno);
29
30     free(*adresa_korena);
31     *adresa_korena =NULL;
32 }
33
34 void prover_i_alokaciju( Cvor* novi) {
35     if( novi == NULL) {
36         fprintf(stderr, "Malloc greska za nov cvor!\n");
37         exit(EXIT_FAILURE);
38     }
39 }
40
41 void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
42     /* Postojece stablo je prazno*/
43     if( *adresa_korena == NULL){
44         Cvor* novi = napravi_cvor(broj);
45         prover_i_alokaciju(novi);
46         *adresa_korena = novi; /* Kreirani Cvor novi ce biti od
47             sada koren stabla*/
48         return;
49     }
50
51     /* Brojeve smestamo u uredjeno binarno stablo, pa
52     ako je broj koji ubacujemo manji od broja koji je u korenu */
53     if( broj < (*adresa_korena)->broj)
54         /* Dodajemo u levo podstablo */
55         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
56     /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
57     dodajemo nov Cvor desno od korena */
58     else
59         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
60 }

```

```

1 #ifndef __STABLA_H__
2 #define __STABLA_H__ 1
3
4 /* Struktura kojom se predstavlja Cvor stabla */
5 typedef struct dcvor{
6     int broj;
7     struct dcvor* levo, *desno;
8 }

```



```

8 } Cvor;

10 /* Funkcija alocira prostor za novi Cvor stabla, inicijalizuje polja
   strukture i vraća pokazivac na nov Cvor */
12 Cvor* napravi_cvor(int b );

14 /* Oslobadjamo dinamički alociran prostor za stablo
   * Nakon oslobadjanja se u pozivajućoj funkciji koren
16 * postavlja NULL, jer je stablo prazno */
void oslobodi_stablo(Cvor** adresa_korena);

18

20 /* Funkcija proverava da li je novi Cvor ispravno alociran,
   * i nakon toga prekida program */
22 void prover_i_alokaciju( Cvor* novi);

24

26 /* Funkcija dodaje nov Cvor u stablo i
   * azurira vrednost korena stabla u pozivajućoj funkciji.
   */
28 void dodaj_u_stablo(Cvor** adresa_korena, int broj);

30 #endif

```

### Rešenje 5.3

```

#include <stdio.h>
2 #define MAX 50

4

int main()
6 {
    int m[MAX][MAX];
    int v, k;
    int i, j;
    int max_broj_negativnih, max_indeks_kolone;
    int broj_negativnih;

    /* Ucitavamo dimenzije matrice */
14 scanf("%d", &v);
    if (v < 0 || v > MAX) {
        fprintf(stderr, "-1\n");
16         return 0;
    }

20     scanf("%d", &k);
    if (k < 0 || k > MAX) {
        fprintf(stderr, "-1\n");
22         return 0;
    }

24 }

```

```

26  /* Ucitavamo elemente matrice */
    for (i = 0; i < v; i++) {
28      for (j = 0; j < k; j++) {
          scanf("%d", &m[i][j]);
30      }
    }

32
    /* Pronalazimo kolonu koja sadrzi najveći broj negativnih
34     elemenata */
    max_indeks_kolone = 0;

36
    max_broj_negativnih = 0;
38    for (i = 0; i < v; i++) {
        if (m[i][0] < 0) {
40            max_broj_negativnih++;
        }

42    }

44
    for (j = 0; j < k; j++) {
46        broj_negativnih = 0;
        for (i = 0; i < v; i++) {
48            if (m[i][j] < 0) {
                broj_negativnih++;
50            }
            if (broj_negativnih > max_broj_negativnih) {
52                max_indeks_kolone = j;
            }

54        }

56    }

58    /* Ispisujemo traženi rezultat */
    printf("%d\n", max_indeks_kolone);

60
    /* Završavamo program */
62    return 0;
}

```

### Rešenje 5.4

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #define MAX 128

5
int main(int argc, char **argv)
7 {
    FILE *f;
9    int brojac = 0;
    char linija[MAX], *p;

```

```

11  if (argc != 3) {
13      fprintf(stderr, "-1\n");
14      exit(EXIT_FAILURE);
15  }

17  if ((f = fopen(argv[1], "r")) == NULL) {
18      fprintf(stderr, "-1\n");
19      exit(EXIT_FAILURE);
20  }

21  while (fgets(linija, MAX, f) != NULL) {
22      p = linija;
23      while (1) {
24          p = strstr(p, argv[2]);
25          if (p == NULL)
26              break;
27          brojac++;
28          p = p + strlen(argv[2]);
29      }
30  }

31  fclose(f);

32  printf("%d\n", brojac);

33  return 0;
34  }

```

### Rešenje 5.5

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  typedef struct _trougao {
6      double xa, ya, xb, yb, xc, yc;
7  } trougao;
8
9  double duzina(double x1, double y1, double x2, double y2) {
10     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
11 }
12
13 double povrsina(trougao t) {
14     double a = duzina(t.xb, t.yb, t.xc, t.yc);
15     double b = duzina(t.xa, t.ya, t.xc, t.yc);
16     double c = duzina(t.xa, t.ya, t.xb, t.yb);
17     double s = (a + b + c) / 2;
18     return sqrt(s * (s - a) * (s - b) * (s - c));
19 }
20

```

```
int poredi(const void *a, const void *b) {
22   trougao x = *(trougao*)a;
   trougao y = *(trougao*)b;
24   double xp = povrsina(x);
   double yp = povrsina(y);
26   if (xp < yp)
       return 1;
28   if (xp > yp)
       return -1;
30   return 0;
}

32
int main() {
34   FILE *f;
   int n, i;
36   trougao *niz;

38   if ((f = fopen("trouglovi.txt", "r")) == NULL) {
       fprintf(stderr, "-1\n");
40       exit(EXIT_FAILURE);
   }

42
   if (fscanf(f, "%d", &n) != 1) {
44       fprintf(stderr, "-1\n");
       exit(EXIT_FAILURE);
46   }

48   if ((niz = malloc(n * sizeof(trougao))) == NULL) {
       fprintf(stderr, "-1\n");
50       exit(EXIT_FAILURE);
   }

52
   for (i = 0; i < n; i++) {
54       if (fscanf(f, "%lf%lf%lf%lf%lf%lf",
                   &niz[i].xa, &niz[i].ya,
56                   &niz[i].xb, &niz[i].yb,
                   &niz[i].xc, &niz[i].yc) != 6) {
58           fprintf(stderr, "-1\n");
           exit(EXIT_FAILURE);
60       }
   }

62
   qsort(niz, n, sizeof(trougao), &poredi);

64
   for (i = 0; i < n; i++)
66       printf("%g %g %g %g %g %g\n",
              niz[i].xa, niz[i].ya,
68              niz[i].xb, niz[i].yb,
              niz[i].xc, niz[i].yc);
70
   free(niz);
72   fclose(f);
}
```

```
74     return 0;
75 }

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

5  Cvor *napravi_cvor(int broj)
6  {
7
8      /* Dinamicki kreiramo cvor */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

11     /* U slucaju greske ... */
12     if (novi == NULL) {
13         fprintf(stderr, "-1\n");
14         exit(1);
15     }

17     /* Inicijalizacija */
18     novi->vrednost = broj;
19     novi->levi = NULL;
20     novi->desni = NULL;

21
22     /* Vracamo adresu novog cvora */
23     return novi;
24 }

25 void dodaj_u_stablo(Cvor **koren, int broj)
26 {
27
28     /* Izlaz iz rekurzije: ako je stablo bilo prazno,
29        novi koren je upravo novi cvor */
30     if (*koren == NULL) {
31         *koren = napravi_cvor(broj);
32         return;
33     }

34
35     /* Ako je stablo neprazno, i koren sadrzi manju vrednost
36        od datog broja, broj se umece u desno podstablo,
37        rekurzivnim pozivom */
38     if ((*koren)->vrednost < broj)
39         dodaj_u_stablo(&(*koren)->desni, broj);
40
41     /* Ako je stablo neprazno, i koren sadrzi vecu vrednost
42        od datog broja, broj se umece u levo podstablo,
43        rekurzivnim pozivom */
44     else if ((*koren)->vrednost > broj)
45         dodaj_u_stablo(&(*koren)->levi, broj);
46
47 }
```

```
49 void prikazi_stablo(Cvor * koren)
50 {
51     /* Izlaz iz rekurzije */
52     if (koren == NULL)
53         return;
54
55     prikazi_stablo(koren->levi);
56     printf("%d ", koren->vrednost);
57     prikazi_stablo(koren->desni);
58 }
59
60 Cvor* ucitaj_stablo() {
61     Cvor *koren = NULL;
62     int x;
63     while (scanf("%d", &x) == 1)
64         dodaj_u_stablo(&koren, x);
65     return koren;
66 }
67
68 void oslobodi_stablo(Cvor **koren)
69 {
70
71     /* Izlaz iz rekurzije */
72     if (*koren == NULL)
73         return;
74
75     oslobodi_stablo(&(*koren)->levi);
76     oslobodi_stablo(&(*koren)->desni);
77     free(*koren);
78
79     *koren = NULL;
80 }
```

```
1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura koja predstavlja cvor stabla */
5  typedef struct cvor {
6      int vrednost;    /* Vrednost koja se cuva */
7      struct cvor *levi;    /* Pokazivac na levo podstablo */
8      struct cvor *desni;   /* Pokazivac na desno podstablo */
9  } Cvor;
10
11 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira
12    dinamicki, funkcijom malloc(). U slucaju greske program
13    se prekida i ispisuje se poruka o gresci. U slucaju
14    uspeha inicijalizuje se vrednost datim brojem, a pokazivaci
15    na podstabla se inicijalizuju na NULL. Funkcija vraća
16    adresu novokreiranog cvora */
17 Cvor *napravi_cvor(int broj);
18
19 /* Funkcija dodaje novi cvor u stablo sa datim korenom.
```

```

21     Ukoliko broj vec postoji u stablu, ne radi nista.
    Cvor se kreira funkcijom napravi_cvor(). */
void dodaj_u_stablo(Cvor **koren, int broj);
23
/* Funkcija prikazuje stablo s leva u desno (tj.
25    prikazuje elemente u rastucem poretku) */
void prikazi_stablo(Cvor * koren);
27
/* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
    vraca
29    pokazican na njegov koren */
Cvor* ucitaj_stablo();
31
/* Funkcija oslobadja prostor koji je alociran za
33    cvorove stabla. */
void oslobodi_stablo(Cvor **koren);
35
#endif

```

### Rešenje 5.6

```

#include <stdio.h>
#include "stabla.h"
4
int f3(Cvor * koren, int n)
{
6     if (koren == NULL || n < 0)
        return 0;
8     if (n == 0) {
        if (koren->levi == NULL && koren->desni != NULL)
10         return 1;
        if (koren->levi != NULL && koren->desni == NULL)
12         return 1;
        return 0;
14     }
    return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
16 }

18 int main()
{
20     Cvor *koren;
    int n;
22
    scanf("%d", &n);
24     koren = ucitaj_stablo();

26     printf("%d\n", f3(koren, n));

28     oslobodi_stablo(&koren);

30     return 0;

```

## ***5 Ispitni rokovi***

---

}
---