

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 . . . . .	11
1.4	Rešenja . . . . .	19
<b>2</b>	<b>Pokazivači</b>	<b>63</b>
2.1	Pokazivačka aritmetika . . . . .	63
2.2	Višedimenzioni nizovi . . . . .	66
2.3	Dinamička alokacija memorije . . . . .	71
2.4	Pokazivači na funkcije . . . . .	75
2.5	Rešenja . . . . .	76
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>115</b>
3.1	Pretraživanje . . . . .	115
3.2	Sortiranje . . . . .	120
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	130
3.4	Rešenja . . . . .	135
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>211</b>
4.1	Liste . . . . .	211
4.2	Stabla . . . . .	225
4.3	Rešenja . . . . .	235
<b>5</b>	<b>Razno</b>	<b>311</b>
5.1	Zadaci . . . . .	311
5.2	Rešenja . . . . .	311

<b>6</b>	<b>Ispitni rokovi</b>	<b>313</b>
6.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	313
6.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	314
6.3	Rešenja . . . . .	316



# 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 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, dok test program 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 broj 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. *Uputstvo: redosled navođenja koeficijenata u nizu treba da bude takav da se 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), ispisuje se polinom  $p$  u odgovarajućem obliku. Zatim se od korisnika traži da unese tačka u kojoj se računa vrednost tog polinoma i ispisuje izračunata vrednost zaokružena na dve decimale. Nakon toga unosi se 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):
3 1 2 3 4
1.00x^3+2.00x^2+3.00x+4.00
Unesite tacku 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 1.00x^3+2.00x^2+3.00x+4.00 i 1.00x^2+1.00 polinoma je : 1.00x^3+3.00x^2+3.00x+5.00
Zbir polinoma 1.00x^3+2.00x^2+3.00x+4.00 i 1.00x^2+1.00 polinoma je : 1.00x^5+2.00x^4+4.00x^3+6.00x^2+3.00x+4
Unosite izvod polinoma koji zelite:
2
2. izvod polinoma 1.00x^3+2.00x^2+3.00x+4.00 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.
- (g) Napisati funkcije koje sabiraju, oduzimaju, množe i dele dva razlomka.

## 1 Uvodni zadaci

---

Napisati program koji testira prethodne funkcije tako što se sa standardnog ulaza unose dva razlomka `r1` i `r2` i na standardni izlaz se ispisuju skraćeni vrednosti razlomaka koji predstavljaju zbir, razliku, proizvod i količnik razlomka `r1` i recipročne vrednosti razlomka `r2`.

*Test 1*

```
|| Ulaz:   Unesite imenilac i brojilac: 1 2
||         Unesite imenilac i brojilac: 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 celog broja  $x$ . Napisati program koja testira funkciju `print_bits` za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

*Test 1*

```
|| Ulaz:   0x7F
|| Izlaz:   0000 0000 0000 0000 0000 0000 0111 1111
```

*Test 2*

```
|| Ulaz:   0x80
|| Izlaz:   0000 0000 0000 0000 0000 0000 1000 0000
```

*Test 3*

```
|| Ulaz:   0x00FF00FF
|| Izlaz:   0000 0000 1111 1111 0000 0000 1111 1111
```

*Test 4*

```
|| Ulaz:   0xFFFFFFFF
|| Izlaz:   1111 1111 1111 1111 1111 1111 1111 1111
```

*Test*

```
|| Ulaz:   0xABCDE123
|| Izlaz:   1010 1011 1100 1101 1110 0001 0010 0011
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkciju koja broji bitove postavljene na 1 u zapisu 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.

<p><i>Test 1</i></p> <pre> Ulaz:  0x7F Izlaz:   Broj bitova u zapisu je  7         </pre>	<p><i>Test 2</i></p> <pre> Ulaz:  0x80 Izlaz:   Broj bitova u zapisu je  1         </pre>	<p><i>Test 3</i></p> <pre> Ulaz:  0x00FF00FF Izlaz:   Broj bitova u zapisu je 16         </pre>
<p><i>Test 4</i></p> <pre> Ulaz:  0xFFFFFFFF Izlaz:   Broj bitova u zapisu je 32         </pre>	<p><i>Test 4</i></p> <pre> Ulaz:  0xABCDE123 Izlaz:   Broj bitova u zapisu je 17         </pre>	

[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.

<p><i>Test 1</i></p> <pre> Ulaz:  0x7F Izlaz:   Najveci:     1111 1110 0000 0000 0000 0000 0000 0000   Najmanji:     0000 0000 0000 0000 0000 0000 0111 1110         </pre>	<p><i>Test 2</i></p> <pre> Ulaz:  0x80 Izlaz:   Najveci:     1000 0000 0000 0000 0000 0000 0000 0000   Najmanji:     0000 0000 0000 0000 0000 0000 0000 0001         </pre>
---	---

### Test 3

```
Ulaz:  0x00FF00FF
Izlaz:
Najveci:
1111 1111 1111 1111 0000 0000 0000 0000
Najmanji:
0000 0000 0000 0000 1111 1111 1111 1111
```

### Test 4

```
Ulaz:  0xFFFFFFFF
Izlaz:
Najveci:
1111 1111 1111 1111 1111 1111 1111 1111
Najmanji:
1111 1111 1111 1111 1111 1111 1111 1111
```

### Test 4

```
Ulaz:  0xABCDE123
Izlaz:
Najveci:
1111 1111 1111 1111 1000 0000 0000 0000
Najmanji:
0000 0000 0000 0001 1111 1111 1111 1111
```

[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 kada se  $n$  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:   255  
| Izlaz:  
|      000000000000000000000000000000001001010101  
|      1010101001000000000000000000000000
```

[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:  111111115
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:  111111115
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>
<code>Poziv: ./a.out 1 2</code>	<code>Poziv: ./a.out 1 2</code>	<code>Poziv: ./a.out 12 12</code>
<code>Ulaz: 11</code>	<code>Ulaz: 1024</code>	<code>Ulaz: 12345</code>
<code>Izlaz: 13</code>	<code>Izlaz: 1024</code>	<code>Izlaz: 12345</code>

**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>
<code>Ulaz: 11</code>	<code>Ulaz: 1024</code>	<code>Ulaz: 12345</code>
<code>Izlaz: 0000000B</code>	<code>Izlaz: 00000400</code>	<code>Izlaz: 00003039</code>

[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>
<code>Ulaz: 123 10</code>	<code>Ulaz: 3251 0</code>	<code>Ulaz: 12541 1024</code>
<code>Izlaz: 4294967285</code>	<code>Izlaz: 4294967295</code>	<code>Izlaz: 4294966271</code>

**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>
<code>Ulaz: 123</code>	<code>Ulaz: 3245</code>	<code>Ulaz: 100328</code>
<code>Izlaz: 0</code>	<code>Izlaz: 2</code>	<code>Izlaz: 1</code>

## 1.3 Rekurzija



## 1 Uvodni zadaci

---

**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.

[Rešenje 1.17]

*Test 1*

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

**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.

*Test 1*

```
|| Ulaz: 11
|| Izlaz: Uneti broj ima paran broj cifara
```

*Test 2*

```
|| Ulaz: 123
|| Izlaz: Uneti broj ima neparan broj cifara
```

[Rešenje 1.18]

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

*Test 1*

```
|| Ulaz: 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:  2 3 5
|| Izlaz: 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]

**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:
|| Suma elemenata je 21
```

### Test 2

```
|| Ulaz: 2 -1 0 -5 -10
|| Izlaz:
|| Suma elemenata je 2
```

### Test 3

```
|| Ulaz: 1 11 3 5 8 1
|| Izlaz:
|| Suma elemenata je 11
```

### Test 4

```
|| Ulaz: 5
|| Izlaz:
|| Suma elemenata je 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]

[Rešenje 1.26]

[Rešenje 1.27]

## 1 Uvodni zadaci

---

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:  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 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 testira prethodno napisane funkcije tako što najpre iscrtava Paskalov trougao

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

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

**Zadatak 1.36** ++ *Modifikacija Hanojskih kula*: Data su četiri vertikalna štapa, na jednom se nalazi  $n$  diskova poluprečnika  $1, 2, 3, \dots$  do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala 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 */
   typedef struct {
6     float real;
       float imag;
8   } KompleksanBroj;

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

16  /* 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
18  ispis nam nije neophodno da imamo adresu
   */
   void ispisi_kompleksan_broj(KompleksanBroj z) {
20     printf("(");
       if( z.real != 0 ) {
22         printf("%.2f",z.real);
           if(z.imag > 0 )
24             printf(" +");
26     }

       if(z.imag !=0 )
28         printf(" %.2f i ",z.imag);

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

32     printf(")");
   }

36  /* Funkcija vraca vrednosti realnog dela kompleksnog broja */

```



## 1 Uvodni zadaci

---

```
38 float realan_deo(KompleksanBroj z) {
    return z.real;
40 }

42 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
float imaginaran_deo(KompleksanBroj z) {
44     return z.imag;
46 }

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

52 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
    odgovara kompleksnom broju poslatom kao argument */
KompleksanBroj konjugovan(KompleksanBroj z) {
54     KompleksanBroj z1 = z;

56     z1.imag *= -1;

58     return z1;
60 }

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

64     z.real += z2.real;
66     z.imag += z2.imag;

68     return z;
70 }

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

74     z.real -= z2.real;
76     z.imag -= z2.imag;

78     return z;
80 }

/* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
    argumenata funkcije */
82 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z;
84 }
```

```
86     z.real = z1.real * z2.real - z1.imag * z2.imag;
      z.imag = z1.real * z2.imag + z1.imag * z2.real;

88     return z;
89 }

90 /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
      kao argument */
92 float argument(KompleksanBroj z) {
      return atan2(z.imag, z.real);
94 }

96 /* U main() funkciji testiramo sve funkcije koje smo definisali */
98 int main() {
      /* deklariseмо promenljive tipa KompleksanBroj */
100     KompleksanBroj z1, z2;

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

104     /* Ucitavamo i drugi kompleksan broj */
106     ucitaj_kompleksan_broj(&z2);

108     /* Ispisujemo prvi kompleksan broj, a zatim i njegov realan i
      imaginaran deo, kao i moduo kompleksnog broja z1 */
      ispisi_kompleksan_broj(z1);
110     printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
      realan_deo(z1), imaginaran_deo(z1), moduo(z1));
      printf("\n");

112     /* Ispisuje drugi kompleksan broj, a zatim i racunamo i
      ispisujemo konjugovano kompleksan broj od z2 */
114     ispisi_kompleksan_broj(z2);
      printf("\nNjegov konjugovano kompleksan broj: ");
116     ispisi_kompleksan_broj( konjugovan(z2) );

118     /* Testiramo funkciju koja racuna argument kompleksnih brojeva */
      printf("\nArgument kompleksnog broja %.f\n",argument(z2) );
120     printf("\n");

122     /* Testiramo sabiranje kompleksnih brojeva */
      printf("\n");
124     ispisi_kompleksan_broj(z1);
      printf(" + ");
126     ispisi_kompleksan_broj(z2);
      printf(" = ");
128     ispisi_kompleksan_broj(saberi(z1, z2));
      printf("\n");

130     /* Testiramo oduzimanje kompleksnih brojeva */
132     printf("\n");
```

## 1 Uvodni zadaci

---

```
134     ispisi_kompleksan_broj(z1);
    printf(" - ");
    ispisi_kompleksan_broj(z2);
136     printf(" = ");
    ispisi_kompleksan_broj(oduzmi(z1, z2));
138     printf("\n");

140     /* Testiramo mnozenje kompleksnih brojeva */
    printf("\n");
142     ispisi_kompleksan_broj(z1);
    printf(" * ");
144     ispisi_kompleksan_broj(z2);
    printf(" = ");
146     ispisi_kompleksan_broj(mnozi(z1, z2));

148     /* Program se završava uspešno, tj, bez greske*/
    return 0;
150 }
```

### Rešenje 1.2

```
2  /* Uključujemo zaglavlje neophodno za rad sa kompleksnim brojevima
   * Ovde je to neophodno jer nam je neophodno da bude poznata
   * definicija tipa KompleksanBroj
   * i da budu uključena zaglavlja standardne biblioteke, neophodna za
   * definicije, a
4  * njih smo već naveli u complex.h
   */
6  #include "complex.h"

8  /* Funkcija učitava sa standardnog ulaza realan i imaginara deo
   * kompleksnog broja i smesta ih u strukturu čija adresa je argument
   * funkcije */
void učitaj_kompleksan_broj(KompleksanBroj* z) {
10     printf("Unesite realan i imaginaran deo kompleksnog broja: ");
    scanf("%f", &z->real);
12     scanf("%f", &z->imag);
}

14

16 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
   * se šalje kao argument u obliku (x + i y)
   * Ovoj funkciji se kompleksan broj prenosi po vrednosti, jer za
   * ispis nam nije neophodno da imamo adresu
   */
18 void ispisi_kompleksan_broj(KompleksanBroj z) {
    printf("(");
20     if( z.real != 0 ) {
        printf("%.2f", z.real);
22         if(z.imag > 0 )
            printf(" +");
24     }
```

```

    }
26     if(z.imag !=0 )
        printf(" %.2f i ",z.imag);
28
    if(z.imag ==0 && z.real ==0 )
30         printf("0 ");

32     printf("\n");
}

34 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
36 float realan_deo(KompleksanBroj z) {
    return z.real;
38 }

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

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

50 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
    odgovara kompleksnom broju poslatom kao argument */
KompleksanBroj konjugovan(KompleksanBroj z) {
52     KompleksanBroj z1 = z;

54     z1.imag *= -1;

56     return z1;
}

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

62     z.real += z2.real;
64     z.imag += z2.imag;

66     return z;
}

68 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
    argumenata funkcije */
70 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z = z1;
72

```

## 1 Uvodni zadaci

---

```
74     z.real -= z2.real;
75     z.imag -= z2.imag;
76
77     return z;
78 }
79
80 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
81    argumenata funkcije */
82 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) {
83     KompleksanBroj z;
84
85     z.real = z1.real * z2.real - z1.imag * z2.imag;
86     z.imag = z1.real * z2.imag + z1.imag * z2.real;
87
88     return z;
89 }
90
91 /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
92    kao argument */
93 float argument(KompleksanBroj z) {
94     return atan2(z.imag, z.real);
95 }
96 }
```

```
2 /*
3  * Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
4  * deklaracije funkcija za rad sa kompleksnim brojevima.
5  * Zaglavlje nikada ne treba da sadrzi definicije funkcija.
6  * Bilo koji program koji bi htio da koristi ove brojeve i funkcije
7  * iz ove biblioteke, neophodno je da ukljuci ovo zaglavlje
8  */
9
10 /* Ovim preprocesorskim direktivama zakljucavamo zaglavlje i time
11    onemogucujemo
12    * da se sadrzaj zaglavlja vise puta ukljuci, ukoliko se u kodu vise
13    puta ukljuci isto zaglavlje
14
15    * Niska posle kljucne reci ifndef je proizvoljna ali treba da se
16    ponovi u narednoj preprocesorskoj define direktivi
17 */
18 #ifndef _COMPLEX_H
19 #define _COMPLEX_H
20
21 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
22    koje se koriste u definicijama funkcija koje smo naveli u complex
23    .c */
24 #include <stdio.h>
25 #include <math.h>
26
27 /* struktura kojom predstavljamo kompleksan broj, cuvajuci njegov
28    realan i imaginaran deo */
29
30 typedef struct {
```

```

22     float real;
23     float imag;
24 } KompleksanBroj;

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

30 void ispisi_kompleksan_broj(KompleksanBroj z) ;

32 float realan_deo(KompleksanBroj z) ;

34 float imaginaran_deo(KompleksanBroj z);

36 float moduo(KompleksanBroj z);

38 KompleksanBroj konjugovan(KompleksanBroj z) ;

40 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) ;

42 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) ;

44 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) ;

46 float argument(KompleksanBroj z) ;

48 /* Kraj zakljucanog dela */
    #endif

```

```

1  /*
2  * Koristimo korektno definisanu biblioteku kompleksnih brojeva.
3  * U zaglavlju complex.h nalazi se definicija kompleksnog broja i
4  * popis deklaracija podrzanih funkcija
5  * a u complex.c se nalaze njihove definicije.
6  *
7  * Ovdje pisemo i main() funkciju drugaciju od prethodnog zadatka.
8  *
9  * I dalje kompilacija programa se najjednostavnije postize naredbom
10 * gcc -Wall -lm -o izvrsni complex.c main.c
11
12 Kompilaciju mozemo uraditi i na sledeci nacin:
13 gcc -Wall -c -o complex.o complex.c
14 gcc -Wall -c -o main.o main.c
15 gcc -lm -o complex complex.o main.o
16
17 Objasnjenje naredbi
18 gcc -Wall -c -o complex.o complex.c
19 Poziva prevodilac za kod complex.c sa opcijama:
20
21 -Wall (stampaj upozorenja prevodioca),
22 -lm (za linkovanje sa math.h bibliotekom),
23 -o (fajl koji prevodilac generise imenuj sa complex.o)

```

## 1 Uvodni zadaci

---

```
23 -c (ne vrši prevodjenje do izvršnog programa, već samo
do objektnog koda).
25 Rezultat ovoga je objektni fajl complex.o, koji sadrži
program na mašinskom jeziku. Još uvek nije izvršni
27 program, jer nije urađeno uvezivanje (linkovanje)
biblioteka koje su u njemu korišćene, i ostalih
29 objektnih fajlova koji se koriste sa njim.
Druga naredba radi analogno za main.c.
31
Ova dva objektna fajla treba ulinkovati međusobno, i sa
33 objektnim kodom standardne biblioteke. To se radi trećom
naredbom. Prevodilac gcc prepoznaje da su njegovi
35 argumenti objektni fajlovi i da ne treba da ih prevodi, već
samo da ih ulinkuje ispravno.
37 gcc -lm -o complex complex.o main.o
*/
39
41 #include <stdio.h>
/* Uključujemo zaglavlje neophodno za rad sa kompleksnim brojevima */
43 #include "complex.h"
45
/* U main funkciji za uneti kompleksan broj ispisujemo njegov polarni
oblik */
47 int main() {
KompleksanBroj z;
49
/* Učitavamo kompleksan broj */
ucitaj_kompleksan_broj(&z);
51
ispisi_kompleksan_broj(z);
53
printf("\n");
55
printf("Polarni oblik kompleksnog broj je %.2f * e^i * %.2f\n",
moduo(z), argument(z));
57
return 0;
59 }
```

### Rešenje 1.3

```
1 #include <stdio.h>
#include <stdlib.h>
3 #include "polinom.h"
5
/* Funkcija koja ispisuje polinom na stdout u citljivom obliku
7 Polinom prenosimo po adresi, da bi uštedeli kopiranje cele
strukture,
```

```

    vec samo prenosimo adresu na kojoj se nalazi polinom kog
    ispisujemo */
9 void ispisi(const Polinom * p)
{
11     int i;
    for (i = p->stepen; i >= 0; i--) {
13     if (p->koef[i]) {
        if (p->koef[i] >= 0)
15         putchar('+');
        if (i > 1)
17         printf("%.2fx^%d", p->koef[i], i);
        else if (i == 1)
19         printf("%.2fx", p->koef[i]);
        else
21         printf("%.2f", p->koef[i]);
    }
23     }
    putchar('\n');
25 }

27 /* Funkcija koja učitava polinom sa tastature */
Polinom ucitaj()
29 {
    int i;
    Polinom p;

33     /* Učitavamo stepen polinoma */
    scanf("%d", &p.stepen);

35     /* Ponavljamo učitavanje stepena sve dok ne unesemo stepen iz
    dozvoljenog opsega */
37     while (p.stepen > MAX_STEPEN || p.stepen < 0) {
        printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
39         scanf("%d", &p.stepen);
    }

41     /* Unosimo koeficijente polinoma */
43     for (i = p.stepen; i >= 0; i--)
        scanf("%lf", &p.koef[i]);
45     return p;
}

47 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
    algoritmom */
49 /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
double izracunaj(const Polinom * p, double x)
51 {
    double rezultat = 0;
53     int i = p->stepen;
    for (; i >= 0; i--)
55     rezultat = rezultat * x + p->koef[i];
    return rezultat;
}

```



```
57 }
59 /* Funkcija koja sabira dva polinoma */
Polinom saberi(const Polinom * p, const Polinom * q)
61 {
    Polinom rez;
63     int i;

    rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;

67     for (i = 0; i <= rez.stepen; i++)
rez.koef[i] =
69         (i > p->stepen ? 0 : p->koef[i]) + (i >
            q->stepen ? 0 : q->
71             koef[i]);
    return rez;
73 }
75
77 /* Funkcija mnozi dva polinoma p i q */
Polinom pomnozi(const Polinom * p, const Polinom * q)
79 {
    int i, j;
    Polinom r;

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

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

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

95     return r;
97 }

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

103     if (p->stepen > 0) {
105         r.stepen = p->stepen - 1;

107         for (i = 0; i <= r.stepen; i++)
            r.koef[i] = (i + 1) * p->koef[i + 1];
```

```

109     } else
110     {
111         r.koef[0] = r.stepen = 0;
112
113         return r;
114     }
115
116 /* Funkcija racuna n-ti izvod polinoma p */
117 Polinom nIzvod(const Polinom * p, int n)
118 {
119     int i;
120     Polinom r;
121
122     if (n < 0) {
123         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
124         exit(EXIT_FAILURE);
125     }
126
127     if (n == 0)
128         return *p;
129
130     r = izvod(p);
131     for (i = 1; i < n; i++)
132         r = izvod(&r);
133
134     return r;
135 }

```

```

2  /* Ovim preprocesorskim direktivama zakljucavamo zaglavlje i time
   *   onemogucujemo
   *   da se sadrzaj zaglavlja vise puta ukljuci
4  */
5  #ifndef _POLINOM_H
6  #define _POLINOM_H
7
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 /* Maksimalni stepen polinoma */
12 #define MAX_STEPEN 20
13
14 /* Polinome predstavljamo strukturom koja cuva
15    koeficijente (koef[i] je koeficijent uz clan x^i)
16    i stepen polinoma */
17 typedef struct {
18     double koef[MAX_STEPEN + 1];
19     int stepen;
20 } Polinom;
21
22 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku

```

## 1 Uvodni zadaci

---

```
24     Polinom prenosimo po adresi, da bi uštedeli kopiranje cele
        strukture,
        vec samo prenosimo adresu na kojoj se nalazi polinom kog
        ispisujemo */
26 void ispisi(const Polinom * p);

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

30
/* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
    algoritmom */
32 /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
double izracunaj(const Polinom * p, double x);

34
/* Funkcija koja sabira dva polinoma */
36 Polinom saberi(const Polinom * p, const Polinom * q);

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

40
/* Funkcija racuna izvod polinoma p */
42 Polinom izvod(const Polinom * p);

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

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

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

8 ili:
gcc -c polinom.c
10 gcc -c test-polinom.c
gcc -o test-polinom polinom.o test-polinom.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
```

```

26      /* Ispis polinoma */
      ispisi(&p);

28      /* Unos tacke u kojoj se racuna vrednost polinoma */
      printf("\nUnesite tacku u kojoj racunate vrednost polinoma\n");
30      scanf("%lf", &x);

32      /* Ispisujemo vrednost polinoma u toj tacki */
      printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&p, x));

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

40      /* Sabiramo polinome i ispisujemo zbir ta dva polinoma */
      r = saberi(&p, &q);
42      printf("Zbir polinoma ");
      ispisi(&p);
44      printf("i polinoma ");
      ispisi(&q);
46      printf("je : ");
      ispisi(&r);

48      /* Mnozimo polinome i ispisujemo proizvod ta dva polinoma */
50      r = pomnozi(&p, &q);
      printf("\nProzvod polinoma ");
52      ispisi(&p);
      printf("i polinoma ");
54      ispisi(&q);
      printf("je : ");
56      ispisi(&r);

58      /* Izvod polinoma */
      printf("\nUnosite izvod polinoma koji zelite: ");
60      scanf("%d", &n);
      r = nIzvod(&p, n);
62      printf("%d. izvod polinoma ", n);
      ispisi(&p);
64      printf("je : ");
      ispisi(&r);

66      /* Uspesno završavamo program */
68      return 0;
}

```

## Rešenje 1.5

```

#include <stdio.h>
2

```

## 1 Uvodni zadaci

```
/* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
   celog broja u memoriji.
4 Bitove u zapisu broja treba da ispisujemo sa leva na desno, tj. od
   bita najvece tezine ka bitu najmanje tezine. Iz tog razloga, za
   pocetnu vrednost maske uzimamo vrednost ciji binarni zapis je
   takav da je bit najvece tezine 1, a svi ostali nule.
   Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u desno,
   kako bismo ocitali naredni bit, gledano s leva na desno.
   Odgovarajuci karakter, ('0' ili '1'), ispisuje se na ekranu.
6 Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
   postavljen na 1, neophodno je da maska bude neoznacena ceo broj i
   da se siftovanjem u desno ova 1 ne bi smatrala znakom i
   prepisivala, vec da bi nam se svakim siftovanjem sa levog kraja
   binarnog zapisa pojavljivale 0.*/
void print_bits( int x) {
8     unsigned velicina = sizeof(int)*8;    /* broj bitova celog broja
   */
   unsigned maska;    /* maska koju cemo koristiti za "ocitavanje"
   bitova */

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

14     putchar('\n');
16 }

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

22     return 0;
24 }
```

### Rešenje 1.6

```
1 #include <stdio.h>

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

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

11     putchar('\n');
13 }
```

```

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

20      /* Kako je argument funkcije oznacen ceo broj x ne mozemo da
    siftujemo x u desno.
21      naredba x>>=1 vrsila bi aritmeticki sift u desno, tj. bitove
    sa desne strane bi bili popunjavani bitom znaka.
22      Npr. -3 bit znaka je 1. U tom slucaju nikad nece biti
    ispunjen uslov x!=0 i program ce biti zarobljen u
23      beskonacnoj petlji.
24      */

25      /* Formiramo masku 1000000...0000000,koja sluzi za
    očitavanje bita najveće težine. U svakoj iteraciji
26      x se pomera u levo za 1 mesto, i očitavamo sledeci
    bit. Petlja se zavrsava kada vise nema jedinica tj.
27      kada x postane nula. */
28      for( ; x!=0 ; x<<=1 )
29          x & maska ? br++ : 1;

30      return br;
31  }

32  int main() {
33      int x;
34      scanf("%x", &x);
35      printf("Broj bitova u zapisu je %d.\n", count_bits(x));
36
37      return 0;
38  }

```

```

1  #include <stdio.h>

2  /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
3  void print_bits( int x) {
4      unsigned velicina = sizeof(int)*8;    /* broj bitova celog broja
    */
5      unsigned maska;    /* maska koju cemo koristiti za "ocitavanje"
    bitova */

6      /* Bitove u zapisu broja treba da ispisujemo sa leva na desno, tj
    . od bita najveće težine ka
7      * bitu najmanje težine. Iz tog razloga, za pocetnu vrednost
    maske uzimamo vrednost
8
9

```

## 1 Uvodni zadaci

---

```

    * ciji binarni zapis je takav da je bit najveće težine 1, a svi
    * ostali nule.
11    * Nakon toga, u svakoj iteraciji ćemo tu jedinicu pomerati u
    * desno, kako bismo očitali
    * naredni bit, gledano s leva na desno. Odgovarajući karakter,
    * ('0' ili '1'), ispisuje se na ekranu.
13    *
    * Zbog siftovanja maske u desno koja na početku ima najviši bit
    * postavljen na 1,
15    * neophodno je da maska bude neoznačen ceo broj i da se
    * siftovanjem u desno ova 1
    * ne bi smatrala znakom i prepisivala, već da bi nam se svakim
    * siftovanjem sa levog kraja
17    * binarnog zapisa pojavljivale 0. */

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

21    putchar('\n');
23 }

25 /* Funkcija vraća broj jedinica u binarnoj reprezentaciji broja x
    * pomeranjem broja x*/
    int count_bits1(int x) {
27        int br=0;
        unsigned wl = sizeof(int)*8 -1;
29        unsigned maska= 1 << wl;

31        /* Kako je argument funkcije označen ceo broj x ne možemo da
        * siftujemo x u desno.
        * naredba x>>=1 vrsila bi aritmetički sift u desno, tj. bitove
33        * sa desne strane bi bili popunjavani bitom znaka.
        * Npr. -3 bit znaka je 1. U tom slučaju nikad neće biti
35        * ispunjen uslov x!=0 i program će biti zarobljen u
        * beskončnoj petlji.
37        */

39        /* Formiramo masku 100000...0000000, koja služi za
        * očitavanje bita najveće težine. U svakoj iteraciji
        * x se pomera u levo za 1 mesto, i očitavamo sledeći
        * bit. Petlja se završava kada više nema jedinica tj.
43        * kada x postane nula. */
        for( ; x!=0 ; x<<=1 )
45            x & maska ? br++ : 1;

47        return br;
49    }

51 int main() {
    int x;
53    scanf("%x", &x);
```

```

55     printf("Broj bitova u zapisu je %d.\n", count_bits1(x));
56     return 0;
57 }

```

### Rešenje 1.7

```

#include <stdio.h>

2
/* Funkcija vraca najveći neoznaceni broj sastavljen iz istih
3 bitova kao i x */
4 unsigned najveći(unsigned x) {
5     unsigned velicina = sizeof(unsigned)*8;
6
7     /* Formiramo masku 100000...0000000 */
8     unsigned maska = 1 << (velicina-1);
9
10    /* Inicijalizujemo rezultat na 0 */
11    unsigned rezultat = 0;
12
13    /* Dokle god postoje jedinice u binarnoj reprezentaciji broja x (
14     tj. dokle god je x razlicit od nule) pomeramo ga ulevo. */
15    for( ; x!=0; x<<=1 ) {
16        /* Za svaku jedinicu, potiskujemo jednu
17         * novu jedinicu sa leva u rezultat */
18        if( x & maska) {
19            rezultat >>= 1;
20            rezultat |= maska;
21        }
22    }
23
24    return rezultat;
25 }
26
27 /* Funkcija vraca najmanji neoznaceni broj
28 sa istim binarnim ciframa kao i x */
29 unsigned najmanji(unsigned x) {
30     /* Inicijalizujemo rezultat na 0 */
31     unsigned rezultat =0;
32
33     /* Dokle god imamo jedinice u broju x, pomeramo ga udesno. */
34     for( ;x!=0 ; x>>=1){
35         /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
36         desna u rezultat */
37         if( x& 1) {
38             rezultat <<=1;
39             rezultat |= 1;
40         }
41     }
42
43     return rezultat;

```



## 1 Uvodni zadaci

```

}
44
/* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
   celog broja u memoriji */
46 void print_bits( int x)  {
   unsigned velicina = sizeof(int)*8;
48   unsigned maska;

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

52   putchar('\n');
54 }

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

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

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

66   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,
5  * pri cemu se broji od nule .
   * Npr, za n=5,  p=10
7  *      1010 1011 1100 1101 1110 1010 1110 0111
   *      1010 1011 1100 1101 1110 1000 0010 0111
9  */
unsigned reset(unsigned x, unsigned n, unsigned p ) {
11   /* Cilj nam je da samo zeljene bitove anuliramo, a da ostali
      ostanu nepromenjeni.
      * Formiramo masku koja ima n bitova postavljenih na 0 pocev od
      pozicije p,
13   * dok su svi ostali postavljeni na 1.
      *
15   * Na primer, za n=5 i p=10
      * formiramo masku oblika
17   * 1111 1111 1111 1111 1111 1000 0011 1111
      * To postizemo na sledeci nacin:
19   * ~0                               1111 1111 1111 1111 1111 1111
      1111
```

```

21      * (~0 << n)          1111 1111 1111 1111 1111 1111 1110 0000
    * ~(~0 << n)          0000 0000 0000 0000 0000 0000 0001 1111
    * ~(~0 << n) << ( p-n+1))          0000 0000 0000 0000 0000
0111 1100 0000
23      * ~(~0 << n) << ( p-n+1))          1111 1111 1111 1111 1111
1000 0011 1111
    */
25      unsigned maska = ~(~0 << n) << ( p-n+1));

27      return x & maska;
29  }

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

45      return x|maska;
47  }

/* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
predstavlja n bitova
49  * pocev od pozicije p u binarnoj reprezentaciji broja x, pri cemu se
pozicija broji
  * sa desna ulevo, gde je pocetna pozicija 0.
51  * Na primer za n = 5 i p = 10 i broj
  *      1010 1011 1100 1101 1110 1010 1110 0111
53  *      0000 0000 0000 0000 0000 0000 0000 1011
  */
55  unsigned get_bits(unsigned x, unsigned n, unsigned p ) {
    /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a ostali
    su 0.
57      * Na primer za n=5
    * 0000 0000 0000 0000 0000 0000 0001 1111
59      */
    unsigned maska = ~(~0 << n);

61      /* Pomeramo sadrzaj u desno tako da trazeno polje bude uz desni
    kraj.
63      * Zatim maskiramo ostale bitove, sem zeljenih n i vracamo

```

```

    vrednost */
    return maska & ( x >> (p-n+1));
65 }

67
69 /* Funkcija vraća broj x kome su n bitova počev od pozicije p
    postavljeni na
    vrednosti n bitova najniže težine binarne reprezentacije broja y
    */
    unsigned set_n_bits(unsigned x, unsigned n, unsigned p , unsigned y)
    {
71         /* Kreiramo masku kod koje su poslednjih n bitova 1, a
            ostali su 0.
            * Na primer za n=5
            * 0000 0000 0000 0000 0000 0000 0001 1111
            */
73         unsigned last_n_1 = ~(~0 << n);

75
77         /* Kao ranije u funkciji reset, kreiramo masku koja ima n bitova
            postavljenih
            * na 0 počevši od pozicije p, dok su ostali bitovi 1.
            * Na primer za n=5 i p =10
            * 1111 1111 1111 1111 1111 1000 0011 1111
            */
79         unsigned middle_n_0 = ~(~(~0 << n ) << (p-n+1));

81
83         /* x sa resetovanih n bita na pozicijama počev od p */
85         unsigned x_reset = x & middle_n_0;

87         /* y cijih je n bitova najniže težine pomereni tako da stoje
            * počev od pozicije p. Ostali bitovi su nule.
            * (y & last_n_1) resetuje sve bitove osim najnižih n
            */
89         unsigned y_shift_middle= (y & last_n_1) << (p-n+1);

91         return x_reset ^ y_shift_middle;
93     }

95
97 /* Funkcija invertuje bitove u zapisu broja x počevši od pozicije p
    njih n*/
    unsigned invert(unsigned x, unsigned n, unsigned p )
99 {
101     /* Formiramo masku sa n jedinica počev od pozicije p
        * Na primer za n=5 i p=10
        * 0000 0000 0000 0000 0000 0111 1100 0000
        */
103     unsigned maska = ~(~0 << n) << (p-n+1);

105
107     /* Operator ekskluzivno ili invertuje sve bitove gde je
        odgovarajući
        bit maske 1. Ostali bitovi ostaju nepromenjeni. */

```

```

    return maska ^ x;
109 }

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

117     for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
        putchar( x & maska ? '1' : '0' );
119
    putchar('\n');
121 }

123
125 int main() {
    unsigned broj, p, n, y;
127     scanf("%u%u%u%u", &broj, &n, &p, &y);
    printf("Broj %5u %25s= ", broj, "");
129     print_bits(broj);

131
    printf("reset(%5u,%5u,%5u)%11s = ", broj, n, p, "");
133     print_bits( reset(broj, n, p));

135     printf("set(%5u,%5u,%5u)%13s = ", broj, n, p, "");
    print_bits( set(broj, n, p));
137
    printf("get_bits(%5u,%5u,%5u)%8s = ", broj, n, p, "");
139     print_bits( get_bits(broj, n, p));

141     printf("y = %31u = ", y);
    print_bits(y);
143     printf("set_n_bits(%5u,%5u,%5u,%5u) = ", broj, n, p, y);
    print_bits( set_n_bits(broj, n, p, y));
145
    printf("invert(%5u,%5u,%5u)%10s = ", broj, n, p, "");
147     print_bits( invert(broj, n, p));

149     return 0;
}

```

### Rešenje 1.9

```

#include <stdio.h>
2
/* Funkcija broj x rotira u levo za n mesta
4 * Na primer za n =5 i x cija je interna reprezentacija

```

```

6  * 1010 1011 1100 1101 1110 0001 0010 0011
   * 0111 1001 1011 1100 0010 0100 0111 0101
   */
8  unsigned rotate_left(int x, unsigned n) {
   unsigned first_bit;
10  /* Maska koja ima samo najvisi bit postavljen na 1 neophodna da
   bismo pre siftovanja u levo za 1 sacuvali najvisi bit. */
   unsigned first_bit_mask = 1 << (sizeof(unsigned)*8 -1);
12  int i;

14  /* n puta vrsimo rotaciju za jedan bit u levo. U svakoj iteraciji
   odredimo prvi bit, a potom pomeramo sadrzaj broja x u levo za 1
   i potom najnizi bit postavljamo na vrednost koju je imao prvi bit
   koji smo istisnuli siftovanjem */
   for( i= 0; i<n; i++) {
16       first_bit = x & first_bit_mask;
       x = x<< 1 | first_bit >> (sizeof(unsigned)*8-1);
18   }
   return x;
20 }

22 /* Funkcija neoznaceni broj x rotira u desno za n
   * Na primer za n =5 i x cija je interna reprezentacija
24  * 1010 1011 1100 1101 1110 0001 0010 0011
   * 0001 1101 0101 1110 0110 1111 0000 1001
26  */
   unsigned rotate_right(unsigned x, unsigned n) {
28       unsigned last_bit;
       int i;

30       /* n puta ponavljamo rotaciju u desno za jedan bit. U svakoj
       iteraciji odredjujemo bit najmanje tezine broja x, zatm tako
       odredjeni bit siftujemo u levo tako da najnizi bit dode do
       pozicije najviseg bita i nakon siftovanja x za 1 u desno
       postavljamo x-ov najvisi bit na vrednost najnizeg bita. */
32       for(i=0; i<n; i++){
           last_bit = x & 1 ;
34           x = x >> 1 | last_bit << (sizeof(unsigned)*8-1);
       }

36       return x;
38   }

40  /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je x
   oznaceni broj */
   int rotate_right_signed(int x, unsigned n) {
42       unsigned last_bit;
       int i;

44       /* U svakoj iteraciji odredjujemo bit najmanje tezine tj.
       last_bit.
46

```

```

Kako je x oznacen ceo broj, tada se prilikom siftovanja u desno
vrsi aritmeticki sift i cuva se znak broja. Iza tog razloga imamo
dva slucaja u zavisnosti od znaka od x.
48 Nije dovoljno da se ova provera izvrši pre petlje, jer rotiranjem
u desno na mesto najviseg bita moze doci i 0 i 1, nezavisno od
pocetnog znaka x.
    */
50 for(i=0; i<n; i++) {
    last_bit = x & 1;

52     if( x<0 )
54         /* Siftovanjem u desno broja koji je negativan dobijamo 1
na najvisoj
        * poziciji. Na primer ako je x
        * 1010 1011 1100 1101 1110 0001 0010 001b
        * (sa b oznacavam u primeru 1 ili 0 na najnizoj
poziciji)
56         *
58         * last_bit je
60         * 0000 0000 0000 0000 0000 0000 0000 000b
        * nakon Siftovanja za 1 u desno
62         * 1101 0101 1110 0110 1111 0000 1001 0001
        *
64         * da bismo najvisu 1 u x postavili na b nije dovoljno da
ga siftujemo
        * na najvisu poziciju jer bi se time dobile 0, a nama su
potrebne 1
66         * zbog bitovskog &
        * zato prvo komplementiramo, pa tek onda siftujemo
68         * ~last_bit << (sizeof(int)*8 -1)
        * B000 0000 0000 0000 0000 0000 0000 0000
70         * (B oznacava ~b )
        * i ponovo komplementiramo da bismo imali b na najvisoj
poziciji
72         * i sve 1 na ostalim pozicijama
        * ~(~last_bit << (sizeof(int)*8 -1))
74         * b111 1111 1111 1111 1111 1111 1111 1111
        */
76         x = (x >>1) & ~(~last_bit << (sizeof(int)*8 -1));
    else
78         x = (x >>1) | last_bit<< (sizeof(int)*8 -1);
    }

80     return x;
82 }

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

```

## 1 Uvodni zadaci

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

92     putchar('\n');
93 }

94 int main() {
95     unsigned x, k;
96     scanf("%x%x", &x, &k);
97     printf("x %36s = ", "");
98     print_bits(x);
99     printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
100    print_bits( rotate_left(x, k));

102    printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
103    print_bits( rotate_right(x, k));

104    printf("rotate_right_signed(%7u,%6u) = ", x, k);
105    print_bits( rotate_right_signed(x, k));

106    return 0;
107 }
108
109
110 }
```

### Rešenje 1.10

```
2 #include <stdio.h>

3 /* Funkcija vraca vrednost cija je binarna reprezentacija slika u
4    ogledalu binarne reprezentacije broja x. Na primer za x cija
5    binarna reprezentacija izgleda ovako
6    * 101010111100110111100100100100011
7    funkcija treba da vrati broj cija binarna reprezentacija izgleda:
8    * 11000100100001111011001111010101
9    */
10 unsigned mirror(unsigned x) {
11     unsigned najnizi_bit;
12     unsigned rezultat = 0;

13     int i;
14     /* Krecemo od najnizeg bita u zapisu broja x i dodajemo ga u
15        rezultat */
16     for(i =0; i < sizeof(x)*8; i++) {
17         najnizi_bit = x & 1;
18         x >>=1;
19         /* Potiskujemo trenutni rezultat ka levom kraju tako svi
20            prethodno postavljeni bitovi dobijaju vecu poziciju, a novi bit
21            postavljamo na najnizu poziciju */
22         rezultat <<= 1;
23         rezultat |= najnizi_bit;
24     }
25     return rezultat;
26 }
```

```

22 }
24
26 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
   celog broja u memoriji */
26 void print_bits( int x) {
   unsigned velicina = sizeof(int)*8;
28   unsigned maska;
   for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
30     putchar( x & maska ? '1' : '0' );
32   putchar('\n');
34 }
36 int main() {
   int broj;
   scanf("%x", &broj);
38
   /*Ispisujemo binarnu reprezentaciju unetog broja*/
40   print_bits(broj);
   putchar('\n');
42   /*Ispisujemo binarnu reprezentaciju broja dobijenog
   pozivom funkcije mirror
44   */
   print_bits( mirror(broj));
46
   return 0;
48 }

```

### Rešenje 1.11

```

#include <stdio.h>
2
/* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
   jedinica veci od broja nula.
4   U suprotnom funkcij vraca 0 */
int Broj01(unsigned int n){
6
   int broj_nula, broj_jedinica;
8   unsigned int maska;

10   /* Postavljamo broj jedinica i broj nula na 0 */
   broj_nula=0;
12   broj_jedinica=0;

14   /* Postavljamo masku tako da pocinjemo sa analiziranjem bita
   najvece tezine */
   maska=1<<(sizeof(unsigned int)*4-1);
16
18

```



## 1 Uvodni zadaci

---

```
/* Dok ne obidjemo sve bitove u zapisu broj n */
20 while(maska!=0){

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

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

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

int main(){
    40 unsigned int n;

    42 /* Ucitavamo broj */
    scanf("%u", &n);
    44

    46 /* Ispujemo vrednost funkcije */
    printf("%d\n", Broj01(n));
    48

    50 return 0;
}
```

### Rešenje 1.12

```
#include <stdio.h>

2 int broj_parova(unsigned int x){

    4     int broj_parova;
    6     unsigned int maska;

    8     /* Postavljamo broj parova na 0 */
    broj_parova=0;

    10     /* Postavljamo masku tako da mozemo da procitamo da li su dva
    najmanja bita u zapisu broja x 11 */
    12     /* broj 3 je binarno 000...00011 */
    maska=3;
}
```

```
14
16
18 /* Dok ne obidjemo sve parove bitova u zapisu broja x */
while(x!=0){

20     /* Proveravamo da li se na najmanjim pozicijama broj x nalazi 11
    par */
    if((x & maska) == maska){
22         broj_parova++;
    }

24     /* Pomeramo broj u desnu stranu tako da mozemo da očitamo
    vrednost sledeceg para bitova */
    x=x>>1;
26 }

30 return broj_parova;

32 }

34 int main(){
    unsigned int x;

36     /* Ucitavamo broj */
    scanf("%u", &x);

38     /* Ispsujemo vrednost funkcije */
    printf("%d\n", broj_parova(x));

42     return 0;

44 }
```

### Rešenje 1.13

### Rešenje 1.14

```
#include <stdio.h>

2
/*
4     Niska koju formiramo je duzine
    (sizeof(unsigned int)*8)/4 +1
6     jer za svaku heksadekadnu cifru nam trebaju 4 binarne cifre i 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
    */
```

## 1 Uvodni zadaci

---

```
12  #define MAX_DUZINA sizeof(unsigned int)*2 +1
14
16  void prevod(unsigned int x, char s[]){
18      int i;
19      unsigned int maska;
20      int vrednost;
21
22      /* Heksadekadni zapis broja 15 je 000...0001111 - ovo nam odgovara
23         ako hocemo da citamo 4 uzastopne cifre */
24      maska=15;
25
26      /*
27         Broj cemo citati od pozicije najmanje tezine ka poziciji najvece
28         tezine;
29         npr. za broj 00000000001101000100001111010101
30         u prvom koraku cemo procitati bitove:
31         0000000000110100010000111101<0101> (bitove izdvojene sa <...>)
32         u drugom koraku cemo procitati:
33         000000000011010001000011<1101>0101
34         u trecem koraku cemo procitati:
35         00000000001101000100<0011>11010101
36         i tako redom
37
38         indeks i oznacava poziciju na koju smestamo vrednost
39
40      */
41      for(i=MAX_DUZINA-2; i>=0; i--){
42          /* Vrednost izdvojene cifre */
43          vrednost=x&maska;
44
45          /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter
46             dobijamo dodavanjem ASCII koda '0'
47             Ako je vrednost iz opsega od 10 do 15 odgovarajuci karakter
48             dobijamo tako sto prvo oduzmemo 10 (dobijamo vrednosti od 0 do 5)
49             pa dodamo ASCII kod 'A' (time dobijamo slova 'A', 'B', ... 'F'
50             ')
51          */
52          if(vrednost<10){
53              s[i]=vrednost+'0';
54          }
55          else{
56              s[i]=vrednost-10+'A';
57          }
58
59          /* Broj pomeramo za 4 bita u desnu stranu tako da mozemo da
60             procitamo sledecu cifru */
61          x=x>>4;
62      }
```

```

54     s[MAX_DUZINA-1]='\0';
55 }
56
57 int main(){
58     unsigned int x;
59     char s[MAX_DUZINA];
60
61     /* Ucitavamo broj */
62     scanf("%u", &x);
63
64     /* Pozivamo funkciju */
65     prevod(x, s);
66
67     /* Ispisujemo dobijenu nisku */
68     printf("%s\n", s);
69
70     return 0;
71 }

```

### Rešenje 1.17

```

1  #include <stdio.h>
2
3
4  /* Iskomentarisana je deo koji se ispisuje svaki put kad se
5  * udje u funkciju. Odkomentarisati pozive printf funkcije u
6  * obe funkcije da uocite razliku u broju rekursivnih poziva
7  * obe verzije.
8  */
9
10 /* Linearno resenje se zasniva na cinjenici:
11 *  $x^0 = 1$ 
12 *  $x^k = x * x^{(k-1)}$ 
13 */
14 int stepen(int x, int k)
15 {
16     printf("Racunam stepen (%d, %d)\n", x, k);
17     if(k==0)
18         return 1;
19
20     return x * stepen(x, k-1);
21
22     /*Celo telo funkcije se moze ovako kratko zapisati
23     return k == 0 ? 1 : x * stepen(x,k-1); */
24 }
25
26 /*Druga verzija prethodne funkcije.
27 Obratiti paznju na efikasnost u odnosu na prvu verziju! */
28
29 /* Logaritamsko resenje je zasnovano na cinjenicama:

```

## 1 Uvodni zadaci

---

```
30 * - x^0 =1;
31 * - x^k = x * (x^2 )^(k/2) , za neparno k
32 * - x^k = (x^2)^(k/2) , za parno k
33 *
34 * Ovom resenju ce biti potrebno manje
35 * rekursivnih poziva da bi doslo do rezultata,
36 * i stoga je efikasnije.
37 */
38 int stepen2(int x, int k)
39 {
40     printf("Racunam stepen2 (%d, %d)\n",x,k);
41     if( k == 0)
42         return 1;
43
44     /*Ako je stepen paran*/
45     if((k % 2) == 0)
46         return stepen2(x*x, k/2);
47     /*Inace (ukoliko je stepen neparan) */
48     return x*stepen2(x*x, k/2);
49 }
50
51 main() {
52     int x, k;
53     scanf("%d%d", &x, &k);
54
55     printf("%d",stepen(2,10));
56     printf("\n-----\n");
57     printf("%d\n",stepen2(2,10));
58 }
```

### Rešenje 1.18

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje
7  * uzajamnu (posrednu) rekurziju.
8  */
9
10 /* Deklaracija funkcije neparan mora da bude navedena
11  * jer se ta funkcija koristi u telu funkcije paran,
12  * tj. koristi se pre svoje definicije.
13  * Funkcija je mogla biti deklarirana i u telu funkcije paran.
14  */
15
16 unsigned neparan(unsigned n);
17
18 /* Funkcija vraca 1 ako broj n ima paran broj cifara inace vraca 0.
19  */
20 unsigned paran(unsigned n) {
```

```

20     if(n>=0 && n<=9)
21         return 0;
22     else
23         return neparan(n/10);
24 }

26 /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace vraca
   0. */
unsigned neparan(unsigned n) {
28     if(n>=0 && n<=9)
29         return 1;
30     else
31         return paran(n/10);
32 }

34 /* Glavna funkcija za testiranje */
int main( ) {
36     int n;
37     printf("Unesite ceo broj: ");
38     scanf("%d", &n);

40     printf("Uneti broj ima %sparan broj cifara\n", (paran(n) == 1 ? "
       ": " ne"));

42     return 0;
}

```

### Rešenje 1.19

```

#include <stdio.h>

2
/* Repno-rekurzivna (eng. tail recursive) je ona funkcija
4 * Cije se telo završava rekurzivnim pozivom, pri čemu
   * taj rekurzivni poziv ne učestvuje u nekom izrazu.
6 *
   * Kod ovih funkcija se po završetku za tekuci rekurzivni poziv
8 * umesto skoka na adresu povratka skace na adresu
   * povratka za prethodni poziv, odnosno za poziv na manjoj
10 * dubini. Time se stedi i prostor i vreme.
   *
12 * Ovakve funkcije se mogu lako zameniti odgovarajućom
   * iterativnom funkcijom,
14 * čime se smanjuje prostorna složenost algoritma.
   */

16
/* Pomoćna funkcija koja izračunava n! * result.
18 * Koristi repnu rekurziju. */

20 /* Result je argument u kom ćemo akumulirati do tada izračunatu
   * vrednost faktoriijela.
22 * Kada završimo, tj. kada dodjemo do izlaza iz rekurzije potrebno je

```

```
24  * da vratimo result. */
    int faktorijelRepna(int n, int result) {
        if (n == 0)
26             return result;

28         return faktorijelRepna(n - 1, n * result);
    }

30
31  /* Sada Zelimo da se oslobodimo repne rekurzije koja postoji u
32  * funkciji faktorijelRepna, koristeći algoritam sa predavanja.
33  *
34  * Najpre ćemo vrednost argumenta funkcije postaviti na vrednost
35  * koja bi se prosledjivala rekurzivnom pozivu i pomoću goto naredbe
36  * vratiti se na početak tela funkcije.
37  */
38
39  int faktorijelRepna_v1(int n, int result) {
40      pocetak:
41      if (n == 0)
42          return result;

43
44      result = n*result;
45      n=n-1;
46      goto pocetak;
47  }

48
49  /* Pisanje bezuslovnih skokova (goto naredbi)
50  * nije dobra programerska praksa.
51  * Iskoristićemo prethodni medjukorak da bismo
52  * dobili iterativno rešenje bez bezuslovnih skokova.
53  */
54  int faktorijelRepna_v2(int n, int result) {
55      while( n!=0){
56          result = n*result;
57          n=n-1;
58      }

59
60      return result;
61  }

62
63
64
65  /* Nasim gore navedenim funkcijama pored n, mora da se salje
66  * i 1 za vrednost drugog argumenta u kome će se akumulirati
67  * rezultat. Funkcija faktorijel(n) je ovde radi udobnosti
68  * korisnika, jer je sasvim prirodno da za faktorijel zahteva
69  * samo 1 parametar.
70  * Funkcija faktorijel izracunava n!, tako Sto odgovarajućoj gore
71  * navedenoj funkciji koja zaista racuna faktorijel,
72  * salje ispravne argumente i vraća rezultat koju joj ta funkcija
73  * vrati.
74  * Za testiranje, zameniti u telu funkcije faktorijel poziv
```

```

76  * faktorijelRepna sa pozivom faktorijelRepna_v1, a zatim sa pozivom
77  * funkcije faktorijelRepna_v2.
78  */
79  int faktorijel(int n) {
80      return faktorijelRepna(n, 1);
81  }
82
83  /* Test program */
84  int main(){
85      int n;
86
87      printf("Unesite n (<= 12): ");
88      scanf("%d", &n);
89
90      printf("%d! = %d\n", n , faktorijel(n));
91
92      return 0;
93  }

```

## Rešenje 1.20

## Rešenje 1.21

## Rešenje 1.22

```

1  #include <stdio.h>
2  #define MAX_DIM 1000
3
4  /*
5   * n==0, suma(a,0) = 0
6   * n >0, suma(a,n) = a[n-1]+suma(a,n-1)
7   * Suma celog niza je jednaka sumi prvih n-1 elementa
8   * uveCenoj za poslednji element celog niza.
9   */
10
11  int sumaNiza(int *a, int n)
12  {
13      /* Ne stavljamo strogu jednakost n==0,
14       * za slucaj da korisnik prilikom prvog poziva,
15       * poSalje negativan broj za velicinu niza.
16       */
17      if(n<=0 )
18          return 0;
19
20      return a[n-1] + sumaNiza(a,n-1);
21  }
22
23  /*
24   * n==0, suma(a,0) = 0

```



## 1 Uvodni zadaci

---

```
26  * n > 0, suma(a,n) = a[0]+suma(a+1,n-1)
    *      Suma celog niza je jednaka zbiru prvog elementa
    *      niza i sume preostalih n-1 elementa.
28  */
    int sumaNiza2(int *a, int n)
30  {
        if(n<=0)
32             return 0;

        return a[0] + sumaNiza2(a+1,n-1);
34  }
    int main()
36  {
        int x, a[MAX_DIM];
        int n, i=0;

        /* Ucitavamo broj elemenata niza */
42        scanf("%d", &n);

        /* Ucitavamo n elemenata niza. */
44        for(i=0; i<n; i++)
46            scanf("%d", &a[i]);

        printf("Suma elemenata je %d\n",sumaNiza(a, n));
50
        /*
52        printf("Suma elemenata je %d\n",sumaNiza2(a, n));
        */
54        return 0;
    }
```

### Rešenje 1.23

```
1  #include <stdio.h>
    #define MAX_DIM 256
3
    /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
       dimenzije n */
5  int maksimum_niza(int niz[], int n) {
        /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je
           ujedno i jedini element niza */
7      if (n==1) return niz[0];

        /* Rešavamo problem manje dimenzije */
9      int max=maksimum_niza(niz, n-1);

11     /* Ako nam je poznato rešenje problema dimenzije n-1, rešavamo
        problem dimenzije n */
13     return niz[n-1] > max ? niz[n-1] : max ;
    }
```

```

15 int main ()
16 {
17     int brojevi[MAX_DIM];
18     int n;
19
20     /* Sve dok ne dodjemo do kraja ulaza, ucitavamo brojeve u niz; i
21     predstavlja indeks tekućeg broja. */
22     int i=0;
23     while (scanf("%d", &brojevi[i]) != EOF){
24         i++;
25     }
26     n=i;
27
28     /* Stampamo maksimum unetog niza brojeva */
29     printf("%d\n", maksimum_niza(brojevi, n));
30     return 0;
31 }

```

### Rešenje 1.24

```

1 #include <stdio.h>
2 #define MAX_DIM 256
3
4 int skalarno(int a[], int b[], int n)
5 {
6     /* Izlazak iz rekurzije */
7     if(n==0) return 0;
8
9     /* Na osnovu rešenja problema dimenzije n-1, resavamo problem
10    dimenzije n */
11    else return a[n-1] * b[n-1] + skalarno(a,b,n-1);
12 }
13
14 int main()
15 {
16     int i, a[MAX_DIM], b[MAX_DIM], n;
17
18     /* Unosimo dimenziju nizova, */
19     scanf("%d",&n);
20
21     /* a zatim i same nizove. */
22     for(i=0; i<n; i++)
23         scanf("%d",&a[i]);
24
25     for(i=0; i<n; i++)
26         scanf("%d",&b[i]);
27
28     /* Ispisujemo rezultat skalarnog proizvoda dva ucitana niza. */
29     printf("%d\n", skalarno(a,b,n));

```

## 1 Uvodni zadaci

---

```
30     return 0;
    }
```

### 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 */
8     if(n==1) 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     /* UCitavamo broj koji se trazi */
20     scanf("%d", &x);

22     /* Sve dok ne dodjemo do kraja ulaza, učitavamo brojeve u niz; i
    predstavlja indeks tekućeg broja */
    i=0;
24     while(scanf("%d", &a[i])!=EOF){
        i++;
26     }
    n=i;
28     /* Ispisujemo broj pojave broja x u niz a */
30     printf("%d\n", br_pojave(x,a,i));
    return 0;
32 }
```

### 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) return 0;
8 }
```

```

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

12 int main ()
13 {
14     int x,y,z, a[MAX_DIM];
15     int n;
16
17     /* UCitavaju se tri cela broja za koje se ispituje da li su
18        uzastopni Clanovi niza */
19     scanf("%d%d%d",&x,&y,&z);
20
21     /* Sve dok ne dodjemo do kraja ulaza, uCitavamo brojeve u niz */
22     int i=0;
23     while(scanf("%d", &a[i])!=EOF){
24         i++;
25     }
26     n=i;
27
28     if(tri_uzastopna_clana(x,y,z,a,i))
29         printf("da\n");
30     else
31         printf("ne\n");
32
33     return 0;
34 }

```

### Rešenje 1.27

```

#include <stdio.h>

2
/* funkcija koja broji bitove svog argumenta*/
4 /*
5  * ako je x ==0, onda je  count(x) = 0
6  * inace count(x) = najvisi_bit +count(x<<1)
7  *
8  * Za svaki naredni rekurzivan poziv prosledjuje se x<<1.
9  * Kako se siftovanjem sa desne strane uvek dopisuju 0,
10 * argument x ce u nekom rekurzivnom pozivu biti baS 0 i
11 * izacicemo iz rekurzije.
12 */

13
14 int count(int x) {
15     /* izlaz iz rekurzije*/
16     if(x==0)
17         return 0;
18
19     /*Dakle, neki bit je postavljen na 1.*/
20     /* Proveravamo vrednost najviseg bita
21      * Kako za rekurzivni poziv moramo slati siftovano x i

```

## 1 Uvodni zadaci

```
22      * x je oznacen ceo broj, onda ne smemo koristiti siftovanje
23      * desno, jer funkciji moze biti prosleden i negativan broj.
24      * Iz tog razloga, odlucujemo se da proveramo najvisi,
25      * umesto najnizeg bita*/
26      if( x& (1<<(sizeof(x)*8-1)))
27          return 1 +count(x<<1);
28      /* Najvisi bit je 1.
29      * Sacekacemo da zavrshi poziv koji racuna koliko ima
30      * jedinica u ostatku binarnog zapisa x i potom uvecati
31      * taj rezultat za 1. */
32      else
33          /* Najvisi bit je 0. Stoga je broj jedinica u zapisu x isti
34          * kao broj jedinica u zapisu broja x<<1, jer se
35          * siftovanjem u levo sa desne stane dopisuju 0.*/
36          return count(x<<1);
37
38      /* jednolinijska return naredba sa proverom i rekurzivnim pozivom
39      return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1); */
40  }
41
42
43  int main() {
44      int x;
45      scanf("%x", &x);
46      printf("%d\n",count(x));
47
48      return 0;
49  }
```

### Rešenje 1.29

```
1  #include<stdio.h>
2
3
4  /* Rekurzivna funkcija za odredjivanje najvece heksadekadne cifre u
5  broju */
6  int max_heksadekadna_cifra(unsigned x) {
7      /* izlazak iz rekurzije */
8      if(x==0) return 0;
9      /* Odredjivanje poslednje heksadekadne cifre u broju*/
10     int poslednja_cifra = x&15;
11     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
12     izbrise poslednja heksadekadna cifra*/
13     int max_bez_poslednje_cifre = max_heksadekadna_cifra(x>>4);
14     return poslednja_cifra > max_bez_poslednje_cifre ?
15     poslednja_cifra : max_bez_poslednje_cifre;
16 }
17
18 int main()
```

```

18 {
    unsigned x;
20     scanf("%u", &x);
    printf("%d\n", max_heksadekadna_cifra(x));
22     return 0;
}

```

### Rešenje 1.30

```

#include<stdio.h>

2

4 /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
   */
int max_oktalna_cifra(unsigned x) {
6     /* Izlazak iz rekurziije */
    if(x==0) return 0;
    /* Odredjivanje poslednje oktalne cifre u broju*/
    int poslednja_cifra = x&7;
10    /* Odredjivanje maksimalne oktalne cifre broja kada se iz njega
       izbriše poslednja oktalna cifra*/
    int max_bez_poslednje_cifre = max_oktalna_cifra(x>>3);
12    return poslednja_cifra > max_bez_poslednje_cifre ?
        poslednja_cifra : max_bez_poslednje_cifre;
}

14

16
int main()
18 {
    unsigned x;
20     scanf("%u", &x);
    printf("%d\n", max_oktalna_cifra(x));
22     return 0;
}

```

### Rešenje 1.31

```

#include<stdio.h>
#include<string.h>
2 /* niska može imati najviše 32 karaktera + 1 za terminalnu nulu */
#define MAX_DIM 33

4

6 int palindrom(char s[], int n)
{
8     if((n==1) || (n==0)) return 1;
    return (s[n-1]==s[0]) && palindrom(s+1, n-2);
10 }

```

## 1 Uvodni zadaci

---

```
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 nije */
25      if(palindrom(s,n))
26          printf("da\n");
27      else printf("ne\n");
28
29      return 0;
30  }
```

### 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      int i;
7
8      for(i=1;i<=n;i++)
9          printf("%d ", a[i]);
10     printf("\n");
11 }
12
13 /* Funkcija proverava da li se x vec
14    nalazi u permutaciji na prethodnih 1...n mesta*/
15 int koriscen(int a[], int n, int x){
16     int i;
17     for(i=1; i<=n; i++)
18         if(a[i] == x) return 1;
19
20     return 0;
21 }
22
23 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n}
24    * a[] je niz u koji smesta permutacije
25    * m - oznacava da se na m-tu poziciju u permutaciji
26    *      smesta jedan od preostalih celih brojeva
27    * n- je velicina skupa koji se permutuje
28    * Funkciju pozivamo sa argumentom m=1 jer krecemo da
```

```

29  * formiramo permutaciju od 1. pozicije i nikada
    * ne koristimo a[0].
31  */
void permutacija(int a[], int m, int n){
33      int i;

35      /* Izlaz iz rekurzije:
    * Ako je pozicija na koju treba smestiti broj premasila
37      * velicinu skupa, onda se svi brojevi vec nalaze u
    * permutaciji i ispisujemo permutaciju. */
39      if(m>n) {
          ispisiNiz(a,n);
41      return;
    }

43      /*Ideja: pronalazimo prvi broj koji mozemo da
45      postavimo na m-to mesto u nizu (broj koji se do
    sada nije pojavio u permutaciji). Zatim, rekurzivno
47      pronalazimo one permutacije koje odgovaraju
    ovako postavljenom pocetku permutacije.
49      Kada to zavrismo, proveravamo da li postoji jos neki
    broj koji moze da se stavi na m-to mesto u nizu
51      (to se radi u petlji). Ako
    ne postoji, funkcija je zavrSila sa radom.
53      Ukoliko takav broj postoji, onda ponovo pozivamo
    rekurzivno pronalazenje odgovarajucih permutacija,
55      ali sada sa drugacije postavljenim prefiksom. */

57      for(i=1;i<=n;i++){
59          /* Ako se broj i nije do sada pojavio u permutaciji
    * od 1 do m-1 pozicije, onda ga stavljamo na poziciju m
61      * i pozivamo funkciju da napravi permutaciju za jedan
    * vece duzine, tj. m+1. Inace, nastavljamo dalje, trazeci
63      * broj koji se nije pojavio do sada u permutaciji.
    */
65      if(! koriscen(a,m-1,i)) {
          a[m]=i;
67          /* Pozivamo ponovo funkciju da dopuni ostatak
    * permutacije posle upisivanja i na poziciju m.
69      */
          permutacija(a,m+1,n);
71      }
    }
73 }

75 int main(void) {
    int n;
77     int a[MAX_DUZINA_NIZA];

79     printf("Unesite duzinu permutacije: ");
    scanf("%d", &n);

```



## 1 Uvodni zadaci

---

```
81     if ( n < 0 || n >= MAX_DUZINA_NIZA) {
            fprintf(stderr, "Duzina permutacije mora biti broj veci od 0
i manji od %d!\n", MAX_DUZINA_NIZA);
83         exit(EXIT_FAILURE);
    }

85     permutacija(a,1,n);

87     exit(EXIT_SUCCESS);

89 }
```

### Rešenje 1.33

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

3
   /* Rekurzivna funkcija za racunanje binomnog koeficijenta.      */
5  /* ako je k=0 ili k=n, onda je binomni koeficijent 0
   ako je k 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) {
        return (0<k && k<n) ? binomniKoeficijent (n-1, k-1) +
            binomniKoeficijent (n-1, k) : 1;
9  }
   /* Iterativno izracunavanje datog binomnog koeficijenta.

11
   int binomniKoeficijent (int n, int k) {
13     int i, j, b;
        for (b=i=1, j=n; i<=k; b=b*j--/i++);
15     return b;
   }

17
   /*
19
   /* Prostim opaZanjem se uocava da se svaki element n-te hipotenuze (
       osim ivicnih 1)
21  * dobija kao zbir 2 elementa iz n-1 hipotenuze. Uz pomenute dve nove
       ivicne jedinice lako se zakljucuje
       * da ce suma elementa n-te hipotenuze biti tacno 2 puta veka.
23  */
   int sumaElemenataHipotenuze(int n)
25  {
        return n > 0 ? 2 * sumaElemenataHipotenuze(n-1) : 1;
27  }

29

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

33     printf("Unesite veliCinu Paskalovog trougla: \n");
```

```
35     scanf("%d", &d);

37     /* Ispisivanje Paskalovog trougla */
    putchar ('\n');
39     for (n=0; n<=d; n++) {
        for (i=0; i<d-n; i++) printf (" ");
41         for (k=0; k<=n; k++) printf ("%4d", binomniKoeficient(n, k));
        putchar ('\n');
43     }

45     printf("Racunamo sumu koje hipotenuze: \n");
    scanf("%d", &n);
47

49     if(n<0){
        fprintf(stderr, "Redni broj hipotenuze mora biti veci ili
        jednak od 0!\n");
        exit(EXIT_FAILURE);
51     }
    printf("Suma %d. hipotenuze je: %d\n",n, sumaElemenataHipotenuze(n)
    );
53
    exit(EXIT_SUCCESS);
55 }
```



## 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>

**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.

## 2 Pokazivači

---

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
```

**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?**

*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.
```

**Zadatak 2.4** Napisati program koji ispisuje broj prihvaćenih argumenata komandne linije, a zatim i same argumente 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 iskomentarise? Milena: Meni se cini da je bolje bez komentarisanja, vec da su oba prisutna.

*Test 1*

```
|| Poziv: ./a.out prvi 2. treci -4
|| Izlaz: 5
||      0 ./a.out
||      1 prvi
||      2 2.
||      3 treci
||      4 -4
||      . p 2 -
```

*Test 2*

```
|| Poziv: ./a.out
|| Izlaz: 1
||      0 ./a.out
||      .
```

**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.

*Test 1*

```
|| Poziv: ./a.out programiranje anavolimilovana topot ana anagram t
|| Izlaz: 4
```

*Test 2*

```
|| Poziv: ./a.out a b 11 212
|| Izlaz: 4
```

*Test 3*

```
|| Poziv: ./a.out
|| Izlaz: 0
```

**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 sadrzaj 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.
```

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

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

## 2.2 Višedimenzioni nizovi

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

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati 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. Slično mi se čini i za zadatak 2.17. Zadatak 2.17 - čini mi se da je resenje bez koriscenja bibliotekskih funkcija visak? Zadatak 2.19 — izvuci komentare za učitaj i ispisi ispred funkcija, umesto što 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

## 2.3 Dinamička alokacija memorije

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.

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

Test 2
|| Ulaz:  -1
|| Izlaz: malloc(): neuspela alokacija memorije.
```

**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.

```
Test 1
|| Ulaz:  Jedan Dva
|| Izlaz: JedanDva
```

**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.

```
Test 1
|| Ulaz:  2 3
||       1.2 2.3 3.4
||       4.5 5.6 6.7
|| Izlaz: 6.80
```

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

- (a) Napisati funkciju koja vrši učitavanje matrice sa standardnog ulaza.
- (b) 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`, `sqrt`, `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 }
```

```
12     for(i = 0, j = n-1; i < j; i++, j--) {
13         int t = a[i];
14         a[i] = a[j];
15         a[j] = t;
16     }

18 /* Funkcija obrće elemente niza koriscenjem pokazivacke
19    sintakse. Umesto "void obrni_niz(int *a, int n)" potpis
20    metode bi mogao da bude i "void obrni_niz(int a[], int n)".
21    U oba slucaja se argument funkcije "a" tumaci kao pokazivac,
22    ili tacnije, kao adresu prvog elementa niza. U odnosu na
23    njega se odredjuju adrese ostalih elemenata u nizu */
24 void obrni_niz_v2(int *a, int n)
25 {
26     /* Pokazivaci na elemente niza a */
27     int *prvi, *poslednji;
28
29     for(prvi = a, poslednji = a + n - 1;
30         prvi < poslednji; prvi++, poslednji--) {
31         int t = *prvi;
32         *prvi = *poslednji;
33         *poslednji = t;
34     }
35 }

38 /* Funkcija obrće elemente niza koriscenjem pokazivacke
39    sintakse - modifikovano koriscenje pokazivaca */
40 void obrni_niz_v3(int *a, int n)
41 {
42     /* Pokazivaci na elemente niza a */
43     int *prvi, *poslednji;
44
45     /* Obrćemo niz */
46     for(prvi = a, poslednji = a + n - 1; prvi < poslednji; ) {
47         int t = *prvi;
48
49         /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
50            vrednost koja se nalazi na adresi na koju pokazuje
51            pokazivac "poslednji". Nakon toga se pokazivac "prvi"
52            uvecava za jedan sto za posledicu ima da "prvi" pokazuje
53            na sledeci element u nizu */
54         *prvi++ = *poslednji;
55
56         /* Vrednost promenljive "t" se postavlja na adresu na koju
57            pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
58            umanjuje za jedan, sto za posledicu ima da pokazivac
59            "poslednji" sada pokazuje na element koji mu prethodi u
60            nizu */
61         *poslednji-- = t;
62     }
```

```

}
64
int main()
66 {
    /* Deklaracija niza a od najviše 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++]) ;
13
14     return s;
15 }
16
17 /* Funkcija racuna proizvod elemenata niza */
18 double proizvod(double a[], int n)
19 {
20     double p = 1;
21
22     for(; n; n--)
23         p *= *a++;
24
25     return p;
26 }
27
28 /* Funkcija racuna najmanji element niza */
29 double min(double *a, int n)
30 {
31     /* Za najmanji element se najpre postavlja prvi element */
32     double min = a[0];
33     int i;
34
35     /* Ispitujemo da li se medju ostalim elementima niza
36        nalazi najmanji */
37     for(i = 1; i < n; i++)
38         if ( a[i] < min )
39             min = a[i];
40
41     return min;
42 }
43
44 /* Funkcija racuna najveći element niza */
45 double max(double *a, int n)
46 {
47     /* Za najveći element se najpre postavlja prvi element */
48     double max = *a;
49
50     /* Ispitujemo da li se medju ostalim elementima niza
51        nalazi najveći */
52     for(a++, n--; n > 0; a++, n--)
53         if (*a > max)
54             max = *a;
55
56     return max;
57 }
58
59 int main()
60 {
61     double a[MAX];
62     int n, i;
```

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

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

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

74      /* Testiramo definisane funkcije */
75      printf("zbir = %5.3f\n", zbir(a, n));
76      printf("proizvod = %5.3f\n", proizvod(a, n));
77      printf("min = %5.3f\n", min(a, n));
78      printf("max = %5.3f\n", max(a, n));

79      return 0;
80  }
```

### Rešenje 2.3

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

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

    while( prvi < poslednji ){

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

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

        /* Smanjuje se vrednost elementa na koji pokazuje
        pokazivac poslednji */
        (*poslednji)--;
    }
}
```

```

28     /* Pokazivac poslednji se pomera na prethodni element */
    poslednji--;
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
44 int main()
{
46     int a[MAX];
    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 */
    for(p = a; p - a < n; p++)
66         printf("%d ", *p);
    printf("\n");
68

70     return 0;
    }

```

## Rešenje 2.4

```

#include <stdio.h>

2
/* Argumenti funkcije main mogu da budu broj argumenta komandne

```

## 2 Pokazivači

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

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

24
26    /* Nakon ove petlje "argc" će biti jednako nuli a "argv" će
      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
30    "argv" da 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 */
    for(i=0; i<argc; i++)
38        printf("%c ", argv[i][0]);
    printf("\n");

40
42    /* koristeći pokazivacku sintaksu */

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

46    return 0;
}
```

### 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
16 int main(int argc, char **argv)
{
    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 }

15 int main(int argc, char **argv)
{
17     char rec[MAX_KARAKTERA];
    int br = 0, i = 0, n;
19     FILE *in;

21     /* Ako korisnik nije uneo trazene argumente,
        prijavljujemo gresku */
23     if(argc < 3) {
        printf("Greska: ");
25     printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_dat br_karaktera.\n",

```



```
27                                     argv[0]);
28     exit(EXIT_FAILURE);
29 }
30
31 /* Otvaramo datoteku sa imenom koje se zadaje kao prvi
32    argument komandne linije. */
33 in = fopen(*(argv+1), "r");
34 if(in == NULL){
35     fprintf(stderr, "Greska: ");
36     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
37                                     argv[1]);
38     exit(EXIT_FAILURE);
39 }
40
41 n = atoi(*(argv+2));
42
43 /* Broje se reci cija je duzina jednaka broju zadatom drugim
44    argumentom komandne linije */
45 while(fscanf(in, "%s", rec) != EOF)
46     if(duzina(rec) == n)
47         br++;
48
49 printf("%d\n", br);
50
51 /* Zatvaramo datoteku */
52 fclose(in);
53 return 0;
54 }
```

### Rešenje 2.7

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Funkcija strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
8  {
9      int i;
10     for (i = 0; *(src+i); i++)
11         *(dest+i) = *(src+i);
12 }
13
14 /* Funkcija strcmp() iz standardne biblioteke */
15 int poredjenje_niski(char *s, char *t)
16 {
17     int i;
18     for (i = 0; *(s+i) == *(t+i); i++)
19         if(*(s+i) == '\0')
20             return 0;
```

```

21     return *(s+i) - *(t+i);
22 }
23
24 /* Funkcija strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
26 {
27     int i;
28     for(i = 0; *(s+i); i++)
29         ;
30     return i;
31 }
32
33 /* Funkcija ispituje da li je niska zadata drugim argumentom
34    funkcije sufiks niske zadate prvi argumentom funkcije */
35 int sufiks_niske(char *niska, char *sufiks) {
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     int i;
47     if(duzina_niske(prefiks) <= duzina_niske(niska)) {
48         for(i=0; i<duzina_niske(prefiks); i++)
49             if(*(prefiks+i) != *(niska+i))
50                 return 0;
51         return 1;
52     }
53     else return 0;
54 }
55
56 int main(int argc, char **argv)
57 {
58     /* Ako korisnik nije uneo trazene argumente,
59        prijavljujemo gresku */
60     if(argc < 4) {
61         printf("Greska: ");
62         printf("Nedovoljan broj argumenata komandne linije.\n");
63         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
64               argv[0]);
65         exit(EXIT_FAILURE);
66     }
67
68     FILE *in;
69     int br = 0, i = 0, n;
70     char rec[MAX_KARAKTERA];
71
72     in = fopen(*(argv+1), "r");

```

```
73     if(in == NULL) {
74         fprintf(stderr, "Greska: ");
75         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
76                     argv[1]);
77         exit(EXIT_FAILURE);
78     }
79
80     /* Proveravamo kojom opcijom je pozvan program a zatim
81        učitavamo reci iz datoteke brojimo koliko reci
82        zadovoljava trazeni uslov */
83     if(!(poredjenje_niski(*(argv + 3), "-s")))
84         while(fscanf(in, "%s", rec) != EOF)
85             br += sufiks_niske(rec, *(argv+2));
86     else if (!(poredjenje_niski(*(argv+3), "-p")))
87         while(fscanf(in, "%s", rec) != EOF)
88             br += prefiks_niske(rec, *(argv+2));
89
90     printf("%d\n", br);
91     fclose(in);
92     return 0;
93 }
```

### Rešenje 2.8

```
#include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 #define MAX 100
6
7 /* Deklarisemo funkcije koje cemo kasnije da definisemo */
8 double euklidska_norma( int M[][MAX], int n);
9 int trag(int M[][MAX], int n);
10 int gornja_vandijagonalna_norma(int M[][MAX], int n);
11
12 int main()
13 {
14     int A[MAX][MAX];
15     int i,j,n;
16
17     /* Unosimo dimenziju kvadratne matrice */
18     scanf("%d",&n);
19
20     /* Proveravamo da li je prekoraceno ogranicenje */
21     if( n > MAX || n <= 0) {
22         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
23         fprintf(stderr, "matrice.\n");
24         exit(EXIT_FAILURE);
25     }
26
27     /* 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
42         smesteni u memoriju, kao na traci. To znaci da su svi
44         elementi prve vrste redom smesteni jedan iza drugog. Odmah
        iza poslednjeg elementa prve vrste smesten je prvi element
46         druge vrste za kojim slede svi elementi te vrste
        i tako dalje redom */
48     /*
        for( i = 0; i<n; i++) {
50         for ( j=0 ; j<n; j++)
            printf("%d ", *(A+i+j));
52         printf("\n");
        }
54     */

56     int tr = trag(A,n);
    printf("trag = %d\n",tr);

58     printf("euklidska norma = %.2f\n",euklidska_norma(A,n));
60     printf ("vandijagonalna norma = %d\n",
        gornja_vandijagonalna_norma(A,n));

62     return 0;
64 }

66 /* Definisemo funkcije koju smo ranije deklarirali */

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

76 /* Funkcija izracunava euklidsku normu matrice */
double euklidska_norma(int M[][MAX], int n)
78 {

```

## 2 Pokazivači

```
80     double norma = 0.0;
81     int i,j;
82
83     for(i= 0; i<n; i++)
84         for(j = 0; j<n; j++)
85             norma += M[i][j] * M[i][j];
86
87     return sqrt(norma);
88 }
89
90 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
91 int gornja_vandijagonalna_norma(int M[][MAX], int n)
92 {
93     int norma =0;
94     int i,j;
95
96     for(i=0 ;i<n; i++) {
97         for(j = i+1; j<n; j++)
98             norma += abs(M[i][j]);
99     }
100
101     return norma;
102 }
```

### 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     int i, j;
19
20     for(i=0; i<n; i++) {
21         for(j=0; j<n; j++)
22             printf("%d ", m[i][j]);
23         printf("\n");
24     }
```

```
26     }
27 }
28
29 /* Funkcija proverava da li su zadate kvadratne matrice a i b
30    dimenzije n jednake */
31 int jednake_matrice(int a[][MAX], int b[][MAX], int n) {
32     int i, j;
33
34     for(i=0; i<n; i++)
35         for(j=0; j<n; j++)
36             /* Nasli smo elemente na istim pozicijama u matricama
37                koji se razlikuju */
38             if(a[i][j]!=b[i][j])
39                 return 0;
40
41     /* Prosla je provera jednakosti za sve parove elemenata koji
42        su na istim pozicijama sto znaci da su matrice jednake */
43     return 1;
44 }
45
46 /* Funkcija izracunava zbir dve kvadratne matrice */
47 void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
48 {
49     int i, j;
50
51     for(i=0; i<n; i++)
52         for(j=0; j<n; j++)
53             c[i][j] = a[i][j] + b[i][j];
54 }
55
56 /* Funkcija izracunava proizvod dve kvadratne matrice */
57 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
58 {
59     int i, j, k;
60
61     for(i=0; i<n; i++)
62         for(j=0; j<n; j++) {
63             /* Mnozimo i-tu vrstu prve sa j-tom kolonom druge matrice */
64             c[i][j] = 0;
65             for(k=0; k<n; k++)
66                 c[i][j] += a[i][k] * b[k][j];
67         }
68 }
69
70 int main()
71 {
72     /* Matrice ciji se elementi zadaju sa ulaza */
73     int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
74
75     /* Matrice zbira i proizvoda */
76     int zbir[MAX][MAX], proizvod[MAX][MAX];
```

```
78  /* Dimenzija matrica */
    int n;
80  int i, j;

82  /* Ucitavamo dimenziju kvadratnih matrica i proveravamo njenu
       korektnost */
84  scanf("%d", &n);

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

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

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

100

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

106

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

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

112  return 0;
114 }
```

### 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++) {
17         if(m[i][i] != 1)
18             return 0;
19     }

20     return 1;
21 }

24 /* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono
    je odredjeno matricom koja sadrzi sve elemente polazne matrice
    dopunjene jedinicama na glavnoj dijagonali */
26 void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
27 {
28     int i, j;

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

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

40 /* Funkcija proverava da li je relacija simetricna. Relacija je
    simetricna ako za svaki par elemenata vazi: ako je element
    "i" u relaciji sa elementom "j", onda je i element "j" u
    relaciji sa elementom "i". Ovakve matrice su simetricne u
    odnosu na glavnu dijagonalu */
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++)
53         for(j=0; j<i; j++)
54             if(m[i][j] != m[j][i])
55                 return 0;

56     return 1;
57 }

60 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono
    je odredjeno matricom koja sadrzi sve elemente polazne 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;
67
68     /* Prepisujemo vrednosti elemenata matrice m */
69     for(i=0; i<n; i++)
70         for(j=0; j<n; j++)
71             zatvorenje[i][j] = m[i][j];
72
73     /* Odredjujemo simetricno zatvorenje matrice */
74     for(i=0; i<n; i++)
75         for(j=0; j<n; j++)
76             if(zatvorenje[i][j] == 1)
77                 zatvorenje[j][i] = 1;
78 }
79
80 /* Funkcija proverava da li je relacija tranzitivna. Relacija je
81 tranzitivna ako ispunjava sledece svojstvo: ako je element "i"
82 u relaciji sa elementom "j" i element "j" u relaciji sa
83 elementom "k", onda je i element "i" u relaciji sa elementom
84 "k" */
85 int tranzitivnost (int m[][MAX], int n)
86 {
87     int i, j, k;
88
89     for(i=0; i<n; i++)
90         for(j=0; j<n; j++)
91             /* Pokušavamo da pronadjemo element koji narušava
92              * tranzitivnost */
93             for(k=0; k<n; k++)
94                 if(m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
95                     return 0;
96
97     return 1;
98 }
99
100 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje
101 zadate relacije koriscenjem Varsalovog algoritma */
102 void tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
103 {
104     int i, j, k;
105
106     /* Kopiramo pocetnu matricu u matricu rezultata */
107     for(i=0; i<n; i++)
108         for(j=0; j<n; j++)
109             zatvorenje[i][j] = m[i][j];
110
111     /* Primenom Varsalovog algoritma odredjujemo
112     refleksivno-tranzitivno zatvorenje matrice */
113     for(k=0; k<n; k++)
```

```

116     for(i=0; i<n; i++)
117         for(j=0; j<n; j++)
118             if((zatvorenje[i][k] == 1) && (zatvorenje[k][j] ==1)
119                 && (zatvorenje[i][j] == 0))
120                 zatvorenje[i][j] = 1;
121 }
122
123 /* 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,
143        prijavljujemo 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     }
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
26        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     int i, j;
44     /* Za maksimalni element uzimamo gornji levi ugao */
45     int max=m[0][0], indeks_vrste=0;
46
47     for(i=0; i<n; i++)
48         for(j=0; j<n; j++)
49             /* Ako je tekuci element manji od minimalnog */
50             if(m[i][j]>max) {
51                 /* cuvamo njegovu vrednost */
52                 max=m[i][j];
53                 /* i cuvamo indeks vrste u kojoj se nalazi */
54                 indeks_vrste=i;
55             }
56     return indeks_vrste;
57 }
58
59 /* Funkcija izracunava broj negativnih elemenata matrice */
60 int broj_negativnih(int m[][MAX], int n) {
61     int i, j;
62
63     int broj_negativnih=0;
64
65     for(i=0; i<n; i++)
66         for(j=0; j<n; j++)

```

```
67         if(m[i][j]<0)
68             broj_negativnih++;
69     return broj_negativnih;
70 }
71
72 int main(int argc, char* argv[])
73 {
74     int m[MAX][MAX];
75     int n;
76     int i, j;
77
78     /* Proveravamo broj argumenata komandne linije */
79     if(argc < 2) {
80         printf("Greska: ");
81         printf("Nedovoljan broj argumenata komandne linije.\n");
82         printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
83         exit(EXIT_FAILURE);
84     }
85
86     /* Ucitavamo vrednost dimenzije i proveravamo njenu
87        korektnost */
88     n = atoi(argv[1]);
89
90     if( n > MAX || n <= 0) {
91         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
92         fprintf(stderr, "matrice.\n");
93         exit(EXIT_FAILURE);
94     }
95
96     /* Ucitavamo element po element matrice */
97     for(i=0; i<n; i++)
98         for(j=0; j<n; j++)
99             scanf("%d", &m[i][j]);
100
101     int max_sd = max_sporedna_dijagonala(m, n);
102     int i_min = indeks_min(m, n);
103     int i_max = indeks_max(m, n);
104     int bn = broj_negativnih(m, n);
105
106     /* Ispisujemo rezultat */
107     printf("%d %d %d %d\n", max_sd, i_min, i_max, bn);
108
109     /* Prekidamo izvršavanje programa */
110     return 0;
111 }
```

### Rešenje 2.12

```
#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)
{
10     int i, j;

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

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

22     for(i=0; i<n; i++) {
24         for(j=0; j<n; j++)
            printf("%d ", m[i][j]);
26         printf("\n");
    }
28 }

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

34     /* Proveravam uslov normiranosti, odnosno da li je proizvod
       svake vrste matrice sa samom sobom jednak jedinici */
36     for(i=0; i<n; i++) {
38         /* Izracunavamo skalarni proizvod vrste sa samom sobom */
        proizvod = 0;

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

44         /* Ako proizvod bar jedne vrste nije jednak jedinici, odmah
           zakljucujemo da matrica nije normirana */
46         if(proizvod!=1)
48             return 0;
50     }

52     /* Proveravam uslov ortogonalnosti, odnosno da li je proizvod
       dve bilo koje razlicite vrste matrice jednak nuli */
54     for(i=0; i<n-1; i++) {
        for(j=i+1; j<n; j++) {
```

```
56      /* Izracunavamo skalarni proizvod */
57      proizvod = 0;
58
59      for(k=0; k<n; k++)
60          proizvod += m[i][k] * m[j][k];
61
62      /* Ako proizvod dve bilo koje razlicite vrste nije jednak
63         nuli, odmah zakljucujemo da matrica nije ortogonalna */
64      if(proizvod!=0)
65          return 0;
66  }
67  }
68
69  /* Ako su oba uslova ispunjena, vracamo jedinicu kao
70     rezultat */
71  return 1;
72  }
73
74  int main()
75  {
76      int A[MAX][MAX];
77      int n;
78
79      /* Ucitavamo vrednost dimenzije i proveravamo njenu
80         korektnost */
81      scanf("%d", &n);
82
83      if( n > MAX || n <= 0) {
84          fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
85          fprintf(stderr, "matrice.\n");
86          exit(EXIT_FAILURE);
87      }
88
89      /* Ucitavamo matricu */
90      ucitaj_matricu(A, n);
91
92      /* Ispisujemo rezultat rada funkcije */
93      if(ortonormirana(A,n))
94          printf("da\n");
95      else
96          printf("ne\n");
97
98      return 0;
99  }
100 }
```

### Rešenje 2.13

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

```
4  #define MAX_V 10
   #define MAX_K 10
6
   /* Funkcija proverava da li su ispisani svi elementi iz matrice,
   odnosno da li se nariusio prirodan poredak medju granicama */
8  int krajIspisa(int top, int bottom, int left, int right)
10 {
   return !(top <= bottom && left <= right);
12 }
14 /* Funkcija spiralno ispisuje elemente matrice */
void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16 {
   int i,j,top, bottom,left, right;
18
   top=left = 0;
   bottom=n-1;
   right = m-1;
22
   while( !krajIspisa(top, bottom, left, right) ) {
24     /* Ispisuje se prvi red*/
     for(j=left; j<=right; j++)
26         printf("%d ",a[top][j]);

     /* Spustamo prvi red */
     top++;
30
     if(krajIspisa(top,bottom,left,right))
32         break;

     for(i=top; i<=bottom; i++ )
34         printf("%d ",a[i][right]);
36
     /* Pomeramo desnu kolonu za naredni krug ispisa
     blize levom kraju */
     right--;
40
     if(krajIspisa(top,bottom,left,right))
42         break;

     /* Ispisujemo donju vrstu */
     for(j=right; j>=left; j-- )
44         printf("%d ",a[bottom][j]);
46
     /* Podizemo donju vrstu za naredni krug ispisa */
     bottom--;
50
     if(krajIspisa(top,bottom,left,right))
52         break;

     /* Ispisujemo prvu kolonu*/
     for(i=bottom; i>=top; i-- )
54
```



```
56     printf("%d ",a[i][left]);

58     /* Pripremamo levu kolonu za naredni krug ispisa */
    left++;

60 }
    putchar('\n');
62 }

64 void ucitaj_matricu(int a[][MAX_K], int n, int m)
{
66     int i, j;

68     for(i=0 ;i<n; i++)
        for(j=0; j<m; j++)
70         scanf("%d", &a[i][j]);
72 }

74 int main( )
{
76     int a[MAX_V][MAX_K];
    int m,n;

78     /* Ucitaj broj vrsta i broj kolona matrice */
    scanf("%d",&n);
80     scanf("%d", &m);

82     if( n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
        fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
84         fprintf(stderr, "matrice.\n");
        exit(EXIT_FAILURE);
86     }

88     ucitaj_matricu(a, n, m);
    ispisi_matricu_spiralno(a, n, m);

90     return 0;
92 }
```

### Rešenje 2.14

### Rešenje 2.15

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

4 /* NAPOMENA: Primer demonstrira dinamičku alokaciju niza od n
    elemenata. Dovoljno je alocirati n * sizeof(T) bajtova, gde
6     je T tip elemenata niza. Povratnu adresu malloc()-a treba
    pretvoriti iz void * u T *, kako bismo dobili pokazivac
8     koji pokazuje na prvi element niza tipa T. Na dalje se
```

```

10     elementima moze pristupati na isti nacin kao da nam
11     je dato ime niza (koje se tako i ponasa - kao pokazivac
12     na element tipa T koji je prvi u nizu) */
13 int main()
14 {
15     int *p = NULL;
16     int i, n;
17
18     /* Unosimo dimenziju niza. Ova vrednost nije ogranicena
19     bilo kakvom konstantom, kao sto je to ranije bio slucaj
20     kod staticke alokacije gde je dimenzija niza bila unapred
21     ogranicena definisanim prostorom. */
22     scanf("%d", &n);
23
24     /* Alociramo prostor za n celih brojeva */
25     if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
26         fprintf(stderr, "malloc(): ");
27         fprintf(stderr, "greska pri alokaciji memorije.\n");
28         exit(EXIT_FAILURE);
29     }
30
31     /* Od ovog trenutka pokazivac "p" mozemo da koristimo kao da
32     je ime niza, odnosno i-tom elementu se moze pristupiti
33     sa p[i] */
34
35     /* Unosimo elemente niza */
36     for (i = 0; i < n; i++)
37         scanf("%d", &p[i]);
38
39     /* Ispisujemo elemente niza unazad */
40     for (i = n - 1; i >= 0; i--)
41         printf("%d ", p[i]);
42     printf("\n");
43
44     /* Oslobadjamo prostor */
45     free(p);
46
47     return 0;
48 }

```

### 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;

```

```
10  /* Velicina alocirane memorije */
    int alocirano;

12

14  /* Broj elemenata niza */
    int n;

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

20

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

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

26

28  /* Sve dok je procitani broj razlicit od nule... */
    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 vrsi rezervacija za jos KORAK dodatnih
           mesta za buduće elemente */
36        if(n == alocirano) {
            /* Povecava se broj alociranih mesta */
38            alocirano = alocirano + KORAK;

40            /* Vrsi se realokacija memorije sa novom velicinom */
            /******
42            /* Resenje sa funkcijom malloc() */
            /******
44            /* Vrsi se alokacija memorije sa novom velicinom, a adresa
           pocetka novog memorijskog bloka se cuva u
46            promenljivoj b */
            b = (int*) malloc (alocirano * sizeof(int));

48

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

54

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

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

```
62     }
63
64     /* Svih n elemenata koji pocinju na adresi a prepisujemo
        na novu adresu b */
65     for(i = 0; i < n; i++)
66         b[i] = a[i];
67
68     /* Posle prepisivanja oslobadjamo blok memorije sa pocetnom
        adresom u a */
69     free(a);
70
71     /* Promenljivoj a dodeljujemo adresu pocetka novog, veceg
        bloka koji je prilikom alokacije zapamcen u
        promenljivoj b */
72     a = b;
73
74     /******
        /* Resenje sa funkcijom realloc() */
        /******
75     /* Zbog funkcije realloc je neophodno da i u prvoj
        iteraciji "a" bude inicijalizovano na NULL */
76     /*
        a = (int*) realloc(a,alocirano*sizeof(int));
77
78     if(a == NULL) {
79         fprintf(stderr, "realloc(): ");
80         fprintf(stderr, "greska pri alokaciji memorije.\n");
81         exit(EXIT_FAILURE);
82     }
83     */
84 }
85
86 /* Smestamo element u niz */
87 a[n++] = x;
88
89 /* i učitavamo sledeći element */
90 scanf("%d", &x);
91 }
92
93 /* Ispisujemo brojeve u obrnutom poretaku */
94 for(n--; n>=0; n--)
95     printf("%d ", a[n]);
96 printf("\n");
97
98 /* Oslobadjamo dinamički alociranu memoriju */
99 free(a);
100
101 /* Program se završava */
102 exit(EXIT_SUCCESS);
103 }
```

### Rešenje 2.17

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX 1000
6
7  /* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
8   Ovakvo nesto se moze improvizovati tako sto se u funkciji
9   dinamički kreira niz potrebne velicine, popuni se potrebnim
10  informacijama, a zatim se vrati njegova adresa. Imajuci u
11  vidu cinjenicu da dinamički kreiran objekat ne nestaje
12  kada se izađe iz funkcije koja ga je kreirala, vraceni
13  pokazivac se kasnije u pozivajucoj funkciji moze koristiti
14  za pristup "vracenom" nizu. Medjutim, pozivajuca funkcija
15  ima odgovornost i da se brine o dealokaciji istog prostora */
16
17  /* Funkcija dinamički kreira niz karaktera u koji smesta
18   rezultat nadovezivanja niski. Adresa niza se vraca kao
19   povratna vrednost. */
20  char *nadovezi(char *s, char *t) {
21      /* Dinamički kreiramo prostor dovoljne velicine */
22      char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
23                               * sizeof(char));
24
25      /* Proveravamo uspeh alokacije */
26      if (p == NULL) {
27          fprintf(stderr, "malloc(): ");
28          fprintf(stderr, "greska pri alokaciji memorije.\n");
29          exit(EXIT_FAILURE);
30      }
31
32      /* Kopiramo i nadovezujemo stringove */
33
34      /* Resenje bez koriscenja biblioteckih funkcija */
35      /*
36      int i,j;
37      for(i=j=0; s[j]!='\0'; i++, j++)
38          p[i]=s[j];
39
40      for(j=0; t[j]!='\0'; i++, j++)
41          p[i]=t[j];
42
43      p[i]='\0';
44      */
45
46      /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
47       string.h */
48      strcpy(p, s);
49      strcat(p, t);
50  }
```

```

52  /* Vracamo pokazivac p */
    return p;
54  }

54  int main() {
56      char *s = NULL;
56      char s1[MAX], s2[MAX];

58      /* Ucitavamo dve niske koje cemo da nadovezemo */
60      scanf("%s", s1);
60      scanf("%s", s2);

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

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

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

72      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 dinamicki alociran niz pokazivaca na vrste
        matrice */
10     double** A = NULL;

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

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

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

22     /* Dinamicki 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  /* Dinamicki 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
38  moramo 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)
{
17     int i, j;

19     for(i=0; i<n; i++) {
21         /* Ispisujemo elemente ispod glavne dijagonale matrice */
            for(j=0; j<=i; j++)
23                 printf("%d ", M[i][j]);
                printf("\n");
25     }
}

27 int main() {
29     int m, n, i, j;
    int **matrica = NULL;

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

33     /* Alociramo prostor za niz pokazivaca na vrste matrice */
    matrica = (int **) malloc(n * sizeof(int*));
35     if(matrica == NULL) {
        fprintf(stderr,"malloc(): Neuspela alokacija\n");
37         exit(EXIT_FAILURE);
39     }

41     /* Alociramo prostor za svaku vrstu matrice */
    for(i=0; i<n; i++) {
43         matrica[i] = (int*) malloc(m * sizeof(int));

45         if(matrica[i] == NULL) {
            fprintf(stderr,"malloc(): Neuspela alokacija\n");
47             for(j=0; j<i; j++)
                free(matrica[j]);
            free(matrica);
49             exit(EXIT_FAILURE);
51         }
53     }
```



## 2 Pokazivači

---

```
55   ucitaj_matricu(matrica, n, m);

57   ispisi_elemente_ispod_dijagonale(matrica, n, m);

59   /* Oslobadjamo dinamicki alociranu memoriju za matricu.
   Prvo oslobadjamo prostor rezervisan za svaku vrstu */
61   for( j=0; j<n; j++)
       free(matrica[j]);

63

65   /* Zatim oslobadjamo memoriju za niz pokazivaca na vrste
   matrice */
   free(matrica);

67

   return 0;
69 }
```

### Rešenje 2.20

```
#include<stdio.h>

2
int main(){
4   printf("Hello pokazivaci!\n");
   return 0;
6 }
```

### 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 */
           a[i][j]/=2;
16   else
18       /* Ako je indeks vrste veci od indeksa kolone */
       if(i>j)
20       /* element se nalazi ispod glavne dijagonale
           pa ga dupliramo*/
           a[i][j] *= 2;
22 }
```

```
24 }
25
26 /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
27    sporedne dijagonale */
28 float zbir_ispod_sporedne_dijagonale(float** m, int n)
29 {
30     int i, j;
31     float zbir=0;
32
33     for(i=0; i<n; i++)
34         for(j=0; j<n; j++)
35             /* Ukoliko je zbir indeksa vrste i indeksa kolone
36                elementa veci od n-1, to znaci da se element nalazi
37                ispod sporedne dijagonale */
38             if(i+j>n-1)
39                 zbir+=fabs(m[i][j]);
40
41     return zbir;
42 }
43
44 /* Funkcija ucitava elemente kvadratne matrice dimenzije n
45    iz zadate datoteke */
46 void ucitaj_matricu(FILE* ulaz, float** m, int n) {
47     int i, j;
48
49     for(i=0; i<n; i++)
50         for(j=0; j<n; j++)
51             fscanf(ulaz, "%f", &m[i][j]);
52 }
53
54 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n
55    na standardni izlaz */
56 void ispisi_matricu(float** m, int n) {
57     int i, j;
58
59     for(i=0; i<n; i++){
60         for(j=0; j<n; j++)
61             printf("%.2f ", m[i][j]);
62         printf("\n");
63     }
64 }
65
66 /* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
67 float** alociraj_memoriju(int n) {
68     int i, j;
69     float** m;
70
71     m = (float**) malloc(n * sizeof(float*));
72     if(m == NULL) {
73         fprintf(stderr, "malloc(): Neuspela alokacija\n");
74         exit(EXIT_FAILURE);
75     }
76 }
```

```
76  /* Za svaku vrstu matrice */
77  for(i=0; i<n; i++) {
78      /* Alociramo memoriju */
79      m[i] = (float*) malloc(n * sizeof(float));
80
81      /* Proveravamo da li je doslo do greske pri alokaciji */
82      if(m[i] == NULL) {
83          /* Ako jeste, ispisujemo poruku */
84          printf("malloc(): neuspela alokacija memorije!\n");
85
86          /* Oslobadjamo memoriju zauzetu do ovog koraka */
87          for(j=0; j<i; j++)
88              free(m[j]);
89          free(m);
90          exit(EXIT_FAILURE);
91      }
92  }
93  return m;
94  }
95
96  /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom
97   dimenzije n */
98  void oslobodi_memoriju(float** m, int n)
99  {
100     int i;
101
102     for(i=0; i<n; i++)
103         free(m[i]);
104     free(m);
105 }
106
107 int main(int argc, char* argv[])
108 {
109     FILE* ulaz;
110     float** a;
111     int n;
112
113     /* Ako korisnik nije uneo trazene argumente,
114        prijavljujemo gresku */
115     if(argc < 2) {
116         printf("Greska: ");
117         printf("Nedovoljan broj argumenata komandne linije.\n");
118         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
119         exit(EXIT_FAILURE);
120     }
121
122     /* Otvaramo datoteku za citanje */
123     ulaz = fopen(argv[1], "r");
124     if(ulaz == NULL) {
125         fprintf(stderr, "Greska: ");
126         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
```

```
128         exit(EXIT_FAILURE);
129     }
130
131     /* citamo dimenziju matrice */
132     fscanf(ulaz, "%d", &n);
133
134     /* Alociramo memoriju */
135     a = alociraj_memoriju(n);
136
137     /* Ucitavamo elemente matrice */
138     ucitaj_matricu(ulaz, a, n);
139
140     float zbir = zbir_ispod_sporedne_dijagonale(a, n);
141
142     /* Pozivamo funkciju za modifikovanje elemenata */
143     izmeni(a, n);
144
145     /* Ispisujemo rezultat */
146     printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
147     printf("je %.2f.\n", zbir);
148
149     printf("Transformisana matrica je:\n");
150     ispisi_matricu(a,n);
151
152     /* Oslobadjamo memoriju */
153     oslobodi_memoriju(a, n);
154
155     /* Zatvaramo datoteku */
156     fclose(ulaz);
157
158     /* i prekidamo sa izvršavanjem programa */
159     return 0;
160 }
```

Rešenje [2.22](#)

Rešenje [2.23](#)

Rešenje [2.24](#)

Rešenje [2.25](#)

Rešenje [2.26](#)

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

```
1  #include <stdlib.h>
2  #include <math.h>
3  #include <string.h>
4
5
6  /* NAPOMENA:
7   Zaglavlje math.h sadrzi deklaracije raznih matematickih
8   funkcija. dIzmeu ostalog, to su cSledee funkcije:
9
10  double sin(double x);
11  double cos(double x);
12  double tan(double x);
13  double asin(double x);
14  double acos(double x);
15  double atan(double x);
16  double atan2(double y, double x);
17  double sinh(double x);
18  double cosh(double x);
19  double tanh(double x);
20  double exp(double x);
21  double log(double x);
22  double log10(double x);
23  double pow(double x, double y);
24  double sqrt(double x);
25  double ceil(double x);
26  double floor(double x);
27  double fabs(double x);
28 */
29
30 /* Funkcija tabela() prihvata granice intervala a i b, broj
31 ekvidistantnih ctaaka n, kao i cpokaziva f koji pokazuje
32 na funkciju koja prihvata double argument, i cvraa double
33 vrednost. Za tako datu funkciju ispisuje njene vrednosti
34 u intervalu [a,b] u n ekvidistantnih ctaaka intervala */
35 void tabela(double a, double b, int n, double (*fp)(double))
36 {
37     int i;
38     double x;
39
40     printf("-----\n");
41     for(i=0; i<n; i++) {
42         x= a + i*(b-a)/(n-1);
43         printf("| %8.5f | %8.5f |\n", x, (*fp)(x));
44     }
45     printf("-----\n");
46 }
47
48 /* Umesto da koristimo stepenu funkciju iz zaglavlja
49 math.h -> pow(a,2) cpozivaemo ckorisniku sqr(a) */
50 double sqr (double a)
51 {
52     return a*a;
53 }
54
```

```
int main(int argc, char *argv[])
{
    double a, b;
    int n;
    /* Imena funkcija koja ćemo navoditi su ckraa ili ctano duga
       5 karaktera */
    char ime_fje[6];
    /* Pokazivac na funkciju koja ima jedan argument tipa double i
       povratnu vrednost istog tipa */
    double (*fp)(double);

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

    /* Niska ime_fje žsadi ime žtraene funkcije koja je navedena
       u komandnoj liniji */
    strcpy(ime_fje, argv[1]);

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

```
    fp=&sqr;
108 else {
    printf("Program jos uvek ne podrzava trazenu funkciju!\n");
110     exit(EXIT_SUCCESS);
}

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

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

120 /* Mreza mora da čukljuuje bar krajeve intervala,
    tako da se mora uneti broj veci od 2 */
122 if (n < 2) {
    fprintf(stderr, "Broj čtaaka žmree mora biti bar 2!\n");
124     exit(EXIT_FAILURE);
}

126 /* Ispisujemo ime funkcije */
128 printf("      x %10s(x)\n", ime_fje);

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

134 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), 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
-----
```

[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, svođenjem 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.

#### Test 1

```
Ulaz: 11 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.
```

#### Test 2

```
Ulaz: 14 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. 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.

*Test 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
Ulaz:
  20140076
  Popovic
Izlaz:
  Indeks: 20140076, Ime i prezime: Sonja Stevanovic
  Indeks: 20140032, Ime i prezime: Dejan Popovic
```

[Rešenje 3.3]

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

[Rešenje 3.4]

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

*Test 1*

```
Poziv: ./a.out dat.txt -x
Datoteka:
  12 53
  2.342 34.1
  -0.3 23
  -1 23.1
  123.5 756.12
Izlaz: -0.3 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 prvi element veći od nule i kao rezultat vraća njegovu poziciju u nizu. 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.

*Test 1*

|| Ulaz: -151 -44 5  
12 13 15  
|| Izlaz: 2

*Test 2*

|| Ulaz: -100 -15 -11  
-8 -7 -5  
|| Izlaz: -1

*Test 3*

|| Ulaz: -100 -15 0 13  
55 124 258  
315 516 7000  
|| Izlaz: 3

[Rešenje 3.7]

**Zadatak 3.8** Napisati funkciju koja u opadajuće sortiranom nizu celih brojeva binarnom pretragom pronalazi prvi element manji od nule i kao rezultat vraća njegovu poziciju u nizu. 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.

*Test 1*

|| Ulaz: 151 44 5 -12 -13 -15  
|| Izlaz: 3

*Test 2*

|| Ulaz: 100 55 15 0 -15 -124 -155  
-258 -315 -516 -7000  
|| Izlaz: 4

*Test 3*

|| Ulaz: 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.

- (a) Napisati funkciju, linearne složenosti, koja određuje logaritam pomeranjem broja udesno.

- (b) Napisati funkciju, logaritmske složenosti, koja određuje logaritam koristeći binarnu pretragu.

Tražene funkcije testirati programom koji broj učitava sa standardnog ulaza, a logaritam ispisuje na standardni izlaz.

<p><i>Test 1</i></p> <pre>   Ulaz: 10    Izlaz: 3 3</pre>	<p><i>Test 2</i></p> <pre>   Ulaz: 4    Izlaz: 2 2</pre>	<p><i>Test 3</i></p> <pre>   Ulaz: 17    Izlaz: 4 4</pre>
<p><i>Test 4</i></p> <pre>   Ulaz: 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  $[0, 90^\circ]$ .*

*Test 1*

```
|| Ulaz:
|| 2
|| 2 0 2 1
|| 1 2 2 2
|| Izlaz: 26.57
```

**Zadatak 3.11** Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime (uređen u rastućem poretku 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 standardni izlaz.

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

Test 1

```
|| Ulaz:  R
|| Izlaz: Dusko Radovic
```

## 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.*

Test 1

```
|| Ulaz:  23 64 123 76 22 7
|| Izlaz:  1
```

Test 2

```
|| Ulaz:  21 654 65 123 65 12 61
|| Izlaz:  0
```

[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.*

Test 1

```
|| Ulaz:  anagram ramgana
|| Izlaz:  jesu
```

Test 2

```
|| Ulaz:  anagram anagrm
|| Izlaz:  nisu
```

[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.*

*Test 1*

```

|| Ulaz:  4 23 5 2 4 6 7 34 6 4 5
|| Izlaz: 4

```

*Test 2*

```

|| Ulaz:  2 4 6 2 6 7 99 1
|| Izlaz: 2

```

[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.*

*Test 1*

```

|| Ulaz:  34 134 4 1 6 30 23
|| Izlaz: da

```

*Test 2*

```

|| Ulaz:  12 53 1 43 3 56 13
|| Izlaz: ne

```

[Rešenje 3.15]

**Zadatak 3.16** Napisati funkciju potpisa `int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3, int dim3)` koja prima dva sortirana niza, i na osnovu njih pravi novi sortirani niz koji 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 i program koji testira funkciju, u kome se nizovi unose sa standardnog ulaza, sve dok se ne unese 0. Dimenzija nizova neće biti preko 256.

*Test 1*

```

|| Ulaz:  3 6 7 11 14 35 0 3 5 8 0
|| Izlaz: 3 3 5 6 7 8 11 14 35

```

*Test 2*

```

|| Ulaz:  1 4 7 0 9 11 23 54 75 0
|| Izlaz: 1 4 7 9 11 23 54 75

```

[Rešenje 3.16]

**Zadatak 3.17** Napisati program koji čita sadržaj dve datoteke od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima i kreira jedinstven spisak studenata sortiranih takođe po imenu rastuće. Program dobija nazive datoteka iz komandne linije, i jedinstven 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 3.17]

**Zadatak 3.18** 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 već sortiranim nizovima i na naopako 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

```
Poziv: time ./a.out 100000 -i -o
Izlaz:
real    0m17.631s
user    0m17.604s
sys     0m0.000s
```

#### Upotreba programa 2

```
Poziv: time ./a.out 100000 -b -r
Izlaz:
real    0m0.005s
user    0m0.004s
sys     0m0.000s
```

*Upotreba programa 3*

```

Poziv: time ./a.out 100000 -s
Izlaz:
      real    0m0.071s
      user    0m0.068s
      sys     0m0.000s

```

[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):
Andrija Petrovic
Anja Ilic
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Ivana Markovic
Lazar Micic
Marija Stankovic
Izlaz: 3
```

[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
```

**Zadatak 3.22** Napisati funkciju koja sortira niz niski po broju suglasnika u niski, ukoliko reči imaju isti broj suglasnika tada 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 proizvođjaca      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 trazenim 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 aktivnosti studenata na praktikumima i u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt` upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po prezimenu opadajuće. U datoteci se nalazi ime, prezime studenta, broj časova na

kojima je prisustvovao, kao i ukupan broj urađenih zadataka. Pretpostaviti da studenata neće biti više od 500 i da je za ime studenta dovoljno 20, a za prezime 25 karaktera.

### 3 Algoritmi pretrage i sortiranja

---

#### 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 opadajuce,
pa po duzini imena rastuce:
Aleksandra Cvetic 4 6
Uros Milic 2 5
Dragan Markovic 3 5
Andrija Petrovic 2 5
Nenad Brankovic 2 4
Lazar Micic 1 3
Bojan Golubovic 4 3
Marija Stankovic 1 3
Ognjen Peric 1 2
Anja Ilic 3 1
Ivana Stankovic 3 1
Izlazna datoteka (dat3.txt):
Studenti sortirani po prisustvu opadajuce,
pa po broju zadataka,
pa po prezimenima leksikografski opadajuce:
Aleksandra Cvetic 4 6
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Anja Ilic 3 1
Andrija Petrovic 2 5
Uros Milic 2 5
Nenad Brankovic 2 4
Marija Stankovic 1 3
Lazar Micic 1 3
Ognjen Peric 1 2
```

[Rešenje 3.24]

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

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

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

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

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

#### Test 1

```
Poziv: ./a.out
Datoteka: 5
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
        Mika - Kisa, 5341
Izlaz:  Jelena - Sunce, 92321
        Mika - Kisa, 5341
        Ana - Nebo, 2342
        Pera - Ptice, 327
        Laza - Oblaci, 29
```

#### Test 2

```
Poziv: ./a.out -i
Datoteka: 5
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
        Mika - Kisa, 5341
Izlaz:  Ana - Nebo, 2342
        Jelena - Sunce, 92321
        Laza - Oblaci, 29
        Mika - Kisa, 5341
        Pera - Ptice, 327
```

#### Test 3

```
Poziv: ./a.out -n
Datoteka: 5
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
        Mika - Kisa, 5341
Izlaz:  Mika - Kisa, 5341
        Ana - Nebo, 2342
        Laza - Oblaci, 29
        Pera - Ptice, 327
        Jelena - Sunce, 92321
```

[Rešenje 3.25]

### 3 Algoritmi pretrage i sortiranja

---

**\*\* 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 i na standardni izlaz 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 standardni izlaz.

#### 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 bibliotečke 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 standardni izlaz.



#### Test 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 bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova, ili poruka ukoliko nema takvog. Potom, sa standardnom 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.

#### Test 1

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

[Rešenje 3.32]

**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 karaktera). Sortirati niz  $S$  (bibliotečkom funkcijom `qsort`) i proveriti da li u njemu ima identičnih stringova.

#### Test 1

```
|| Ulaz: 4 prog search sort search
|| Izlaz: ima
```

#### Test 2

```
|| Ulaz: 3 test kol ispit
|| Izlaz: nema
```

[Rešenje 3.34]

### 3 Algoritmi pretrage i sortiranja

---

**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. Napisati program koji sortira (korišćenjem funkcije `qsort`) studente po broju poena (ukoliko je prisutna opcija `-p`) ili po nalogu (ukoliko je prisutna opcija `-n`). Studenti se po nalogu sortiraju tako što se sortiraju na osnovu godine, zatim na osnovu smeru, i na kraju na osnovu broja indeksa. Sortirane studente upisati u datoteku `izlaz.txt`. Ukoliko je u komandnoj liniji uz opciju `-n` naveden i nalog nekog studenta, funkcijom `bsearch` potražiti i prijaviti broj poena studenta sa tim nalogom.

#### Test 1

```
Poziv: ./a.out -n mm13321
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
```

[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), 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

```
Poziv: ./a.out datoteka.txt
Datoteka:      Ulaz:      Izlaz:
1.1.2013        13.12.2016    postoji
13.12.2016     10.5.2015    ne postoji
11.11.2011     5.2.2009    postoji
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 program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz.

*Test 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
```

[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 standardni izlaz.

## 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
7    opcijom -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz celih brojeva duzine n, trazeci u
10    njemu element x. Pretraga se vrši prostom iteracijom kroz
11    niz. Ako se element pronadje funkcija vraća indeks pozicije
12    na kojoj je pronadjen. Ovaj indeks je uvek nenegativan. Ako
13    element nije pronadjen u nizu, funkcija vraća -1, kao
14    indikator neuspesne pretrage. */
15 int linearna_pretraga(int a[], int n, int x)
16 {
17     int i;
```

### 3 Algoritmi pretrage i sortiranja

---

```

    for (i = 0; i < n; i++)
19         if (a[i] == x)
                return i;
21     return -1;
}

23
/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
25     indeks pozicije nadjenog elementa ili -1, ako element nije
    pronadjen */
27 int binarna_pretraga(int a[], int n, int x)
{
    int levi = 0;
    int desni = n - 1;
    int srednji;
    /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
        /* Racunamo srednji indeks */
        srednji = (levi + desni) / 2;
        /* Ako je srednji element veci od x, tada se x mora nalaziti
        u levoj polovini niza */
        if (x < a[srednji])
            desni = srednji - 1;
        /* Ako je srednji element manji od x, tada se x mora
        nalaziti u desnoj polovini niza */
        else if (x > a[srednji])
            levi = srednji + 1;
        else
            /* Ako je srednji element jednak x, tada smo pronasli x na
            poziciji srednji */
            return srednji;
    }
    /* Ako nije pronadjen vracamo -1 */
    return -1;
}

51
/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
53     indeks pozicije nadjenog elementa ili -1, ako element nije
    pronadjen */
55 int interpolaciona_pretraga(int a[], int n, int x)
{
    int levi = 0;
    int desni = n - 1;
    int srednji;
    /* Dokle god je indeks levi levo od indeksa desni... */
    while (levi <= desni) {
        /* Ako je element manji od pocetnog ili veci od poslednjeg
        clana u delu niza a[levi],...,a[desni] tada nije u tom
        delu niza. Ova provera je neophodna, da se ne bi dogodilo
        da se prilikom izracunavanja indeksa srednji izadje izvan
        opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
            return -1;
    }
}
```

```
71  /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
72     a[levi] i a[desni] jednaki, tada je jasno da je x jednako
73     ovim vrednostima, pa vracamo indeks levi (ili indeks
74     desni. Ova provera je neophodna, zato sto bismo inace
75     prilikom izracunavanja srednji imali deljenje nulom. */
76     else if (a[levi] == a[desni])
77         return levi;
78     /* Racunamo srednji indeks */
79     srednji =
80         levi +
81         ((double) (x - a[levi]) / (a[desni] - a[levi])) *
82         (desni - levi);
83     /* Napomena: Indeks srednji je uvek izmedju levi i desni,
84     ali ce verovatno biti blize trazenoj vrednosti nego da
85     smo prosto uvek uzimali srednji element. Ovo se moze
86     porediti sa pretragom recnika: ako neko trazi rec na
87     slovo 'B', sigurno nece da otvori recnik na polovini, vec
88     verovatno negde blize pocetku. */
89     /* Ako je srednji element veci od x, tada se x mora nalaziti
90     u levoj polovini niza */
91     if (x < a[srednji])
92         desni = srednji - 1;
93     /* Ako je srednji element manji od x, tada se x mora
94     nalaziti u desnoj polovini niza */
95     else if (x > a[srednji])
96         levi = srednji + 1;
97     else
98         /* Ako je srednji element jednak x, tada smo pronasli x na
99         poziciji srednji */
100        return srednji;
101    }
102    /* Ako nije pronadjen vracamo -1 */
103    return -1;
104 }
105
106 /* Funkcija main */
107 int main(int argc, char **argv)
108 {
109     int a[MAX];
110     int n, i, x;
111     struct timespec time1, time2, time3, time4, time5, time6;
112     FILE *f;
113
114     /* Provera argumenata komandne linije */
115     if (argc != 3) {
116         fprintf(stderr,
117             "koriscenje programa: %s dim_niza trazeni_br\n",
118             argv[0]);
119         exit(EXIT_FAILURE);
120     }
121
122     /* Dimenzija niza */
```

### 3 Algoritmi pretrage i sortiranja

---

```
123  n = atoi(argv[1]);
    if (n > MAX || n <= 0) {
125      fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
      exit(EXIT_FAILURE);
    }

127  /* Broj koji se trazi */
129  x = atoi(argv[2]);

131  /* Elemente niza odredjujemo slucajno, tako da je svaki
    sledeci veci od prethodnog. srandom() funkcija obezbedjuje
133     novi seed za pozivanje random() funkcije. Kako nas niz ne
    bi uvek isto izgledao seed smo postavili na tekuce vreme u
135     sekundama od Nove godine 1970. random()%100 daje brojeve
    izmedju 0 i 99 */
137  srandom(time(NULL));
    for (i = 0; i < n; i++)
139      a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;

141  /* Linearna pretraga */
    printf("Linearna pretraga\n");
143  /* Racunamo vreme proteklo od Nove godine 1970 */
    clock_gettime(CLOCK_REALTIME, &time1);
145  /* Pretrazujemo niz */
    i = linearna_pretraga(a, n, x);
147  /* Racunamo novo vreme i razlika predstavlja vreme utroseno za
    lin pretragu */
149  clock_gettime(CLOCK_REALTIME, &time2);
    if (i == -1)
151      printf("Element nije u nizu\n");
    else
153      printf("Element je u nizu na poziciji %d\n", i);
    printf("-----\n");

155  /* Binarna pretraga */
    printf("Binarna pretraga\n");
157  clock_gettime(CLOCK_REALTIME, &time3);
    i = binarna_pretraga(a, n, x);
159  clock_gettime(CLOCK_REALTIME, &time4);
    if (i == -1)
161      printf("Element nije u nizu\n");
    else
163      printf("Element je u nizu na poziciji %d\n", i);
    printf("-----\n");

167  /* Interpolaciona pretraga */
    printf("Interpolaciona pretraga\n");
169  clock_gettime(CLOCK_REALTIME, &time5);
    i = interpolaciona_pretraga(a, n, x);
171  clock_gettime(CLOCK_REALTIME, &time6);
    if (i == -1)
173      printf("Element nije u nizu\n");
```

```

else
175     printf("Element je u nizu na poziciji %d\n", i);
    printf("-----\n");
177
    /* Upisujemo podatke o izvršavanju programa u log fajl */
179     if ((f = fopen("vremena.txt", "a")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
181         exit(EXIT_FAILURE);
    }

183
    fprintf(f, "Dimenzija niza od %d elemenata.\n", n);
185     fprintf(f, "\tLinearna pretraga:%10ld ns\n",
        (time2.tv_sec - time1.tv_sec) * 1000000000 +
187         time2.tv_nsec - time1.tv_nsec);
    fprintf(f, "\tBinarna: %19ld ns\n",
189         (time4.tv_sec - time3.tv_sec) * 1000000000 +
        time4.tv_nsec - time3.tv_nsec);
191     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
        (time6.tv_sec - time5.tv_sec) * 1000000000 +
193         time6.tv_nsec - time5.tv_nsec);

195     /* Zatvaramo datoteku */
    fclose(f);
197
    return 0;
199 }

```

### Rešenje 3.2

```

#include <stdio.h>

2
int lin_pretgraga_rek_sufiks(int a[], int n, int x)
4 {
    int tmp;
6     /* Izlaz iz rekurzije */
    if (n <= 0)
8         return -1;
    /* Ako je prvi element trazeni */
10     if (a[0] == x)
        return 0;
    /* Pretraga ostatka niza */
12     tmp = lin_pretgraga_rek_sufiks(a + 1, n - 1, x);
14     return tmp < 0 ? tmp : tmp + 1;
}

16
int lin_pretgraga_rek_prefiks(int a[], int n, int x)
18 {
    /* Izlaz iz rekurzije */
20     if (n <= 0)
        return -1;
22     /* Ako je poslednji element trazeni */

```



```
    if (a[n - 1] == x)
24         return n - 1;
    /* Pretraga ostatka niza */
26     return lin_pretgraga_rek_prefiks(a, n - 1, x);
}

28
int bin_pretgraga_rek(int a[], int l, int d, int x)
30 {
    int srednji;
32     if (l > d)
        return -1;
34     /* Srednja pozicija na kojoj se trazi vrednost x */
    srednji = (l + d) / 2;
36     /* Ako je sredisnji element trazeni */
    if (a[srednji] == x)
38         return srednji;
    /* Ako je trazeni broj veci od srednjeg, pretrazujemo desnu
40     polovinu niza */
    if (a[srednji] < x)
42         return bin_pretgraga_rek(a, srednji + 1, d, x);
    /* Ako je trazeni broj manji od srednjeg, pretrazujemo levu
44     polovinu niza */
    else
46         return bin_pretgraga_rek(a, l, srednji - 1, x);
}

48
int interp_pretgraga_rek(int a[], int l, int d, int x)
50 {
    int p;
52     if (x < a[l] || x > a[d])
        return -1;
54     if (a[d] == a[l])
        return l;
56     /* Pozicija na kojoj se trazi vrednost x */
    p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
58     if (a[p] == x)
        return p;
60     if (a[p] < x)
        return interp_pretgraga_rek(a, p + 1, d, x);
62     else
        return interp_pretgraga_rek(a, l, p - 1, x);
64 }

66
#define MAX 1024
68
int main()
70 {
    int a[MAX];
72     int x;
    int i, indeks;
74 }
```

```

76  /* Ucitavamo trazeni broj */
    printf("Unesite trazeni broj: ");
    scanf("%d", &x);

78
    /* Ucitavamo elemente niza sve do kraja ulaza - ocekujemo da
80     korisnik pritisne CTRL+D za naznaku kraja */
    i = 0;
82    printf("Unesite sortiran niz elemenata: ");
    while (scanf("%d", &a[i]) == 1) {
84        i++;
    }

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

94
    /* Binarna pretraga */
96    printf("Binarna pretraga\n");
    indeks = bin_pretgraga_rek(a, 0, i - 1, x);
98    if (indeks == -1)
        printf("Element se ne nalazi u nizu.\n");
100    else
        printf("Pozicija elementa je %d.\n", indeks);

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

110    return 0;
112 }

```

### Rešenje 3.3

```

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

5  #define MAX_STUDENATA 128
   #define MAX_DUZINA 16

7
   /* O svakom studentu imamo 3 informacije i njih objedinjujemo u
9   strukturu kojom cemo predstavljati svakog studenta. */
   typedef struct {

```

### 3 Algoritmi pretrage i sortiranja

---

```
11  /* Indeks mora biti tipa long jer su podaci u datoteci
    preveliki za int, npr. 20140123 */
13  long indeks;
    char ime[MAX_DUZINA];
15  char prezime[MAX_DUZINA];
    } Student;
17
18  /* Ucitani niz studenata ce biti sortiran prema indeksu, jer cemo
19  ih, redom, kako citamo smestati u niz, a u datoteci su vec
    smesteni sortirani rastuce prema broju indeksa. Iz tog
21  razloga pretragu po indeksu cemo vrsiti binarnom pretragom,
    dok pretragu po prezimenu moramo vrsiti linearno, jer nemamo
23  garancije da postoji uredjenje po prezimenu. */

25  /* Funkcija trazi u sortiranom nizu studenata a[] duzine n
    studenta sa indeksom x. Vraca indeks pozicije nadjenog clana
27  niza ili -1, ako element nije pronadjen */
    int binarna_pretraga(Student a[], int n, long x)
29  {
        int levi = 0;
        int desni = n - 1;
        int srednji;
31        /* Dokle god je indeks levi levo od indeksa desni */
        while (levi <= desni) {
33            /* Racunamo srednji indeks */
            srednji = (levi + desni) / 2;
35            /* Ako je srednji element veci od x, tada se x mora nalaziti
                u levoj polovini niza */
            if (x < a[srednji].indeks)
37                desni = srednji - 1;
            /* Ako je srednji element manji od x, tada se x mora
                nalaziti u desnoj polovini niza */
            else if (x > a[srednji].indeks)
41                levi = srednji + 1;
            else
43                /* Ako je srednji element jednak x, tada smo pronasli x na
                    poziciji srednji */
                    return srednji;
45        }
        /* Ako nije pronadjen vracamo -1 */
47        return -1;
    }

51
52  /* Linearnom pretragom niza studenata trazimo prezime x */
    int linearna_pretraga(Student a[], int n, char x[])
53  {
        int i;
        for (i = 0; i < n; i++)
55            /* Poredimo prezime i-tog studenta i poslato x */
            if (strcmp(a[i].prezime, x) == 0)
57                return i;
        return -1;
    }
```

```
63 }
64
65 /* Main funkcija mora imate argumente jer se ime datoteke dobija
   kao argument komandne linije */
67 int main(int argc, char *argv[])
68 {
69     /* Ucitacemo redom sve studente iz datoteke u niz. */
   Student dosije[MAX_STUDENATA];
71     FILE *fin = NULL;
   int i;
73     int br_studenata = 0;
   long trazen_indeks = 0;
75     char trazeno_prezime[MAX_DUZINA];
76
77     /* Proveravamo da li nam je korisnik prilikom poziva prosledio
   ime datoteke sa informacijama o studentima */
79     if (argc != 2) {
   fprintf(stderr,
81         "Greska: Program se poziva sa %s ime_datoteke\n",
   argv[0]);
83     exit(EXIT_FAILURE);
   }
84
85     /* Otvaramo datoteku */
87     fin = fopen(argv[1], "r");
   if (fin == NULL) {
89         fprintf(stderr,
   "Neuspesno otvaranje datoteke %s za citanje\n",
91         argv[1]);
   exit(EXIT_FAILURE);
93     }
94
95     /* Citamo sve dok imamo red sa informacijama o studentu */
   i = 0;
97     while (1) {
   if (i == MAX_STUDENATA)
99         break;
   if (fscanf
101     (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
   dosije[i].prezime) != 3)
103         break;
   i++;
105     }
   br_studenata = i;
106
107     /* Nakon citanja datoteka nam vise nije neophodna i odmah je
   zatvaramo */
109     fclose(fin);
110
111     /* Unos indeksa koji se binarno trazi u nizu */
113     printf("Unesite indeks studenta cije informacije zelite: ");
   scanf("%ld", &trazen_indeks);
```

### 3 Algoritmi pretrage i sortiranja

```
115 i = binarna_pretraga(dosije, br_studenata, trazeni_indeks);
/* Rezultat binarne pretrage */
117 if (i == -1)
    printf("Ne postoji student sa indeksom %ld\n",
119         trazeni_indeks);
else
121     printf("Indeks: %ld, Ime i prezime: %s %s\n",
        dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
123
/* Unos prezimena koje se linearno trazi u nizu */
125 printf("Unesite prezime studenta cije informacije zelite: ");
scanf("%s", trazeno_prezime);
127 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
/* Rezultat linearne pretrage */
129 if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n",
131         trazeno_prezime);
else
133     printf("Indeks: %ld, Ime i prezime: %s %s\n",
        dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135
return 0;
137 }
```

#### Rešenje 3.4

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

5 #define MAX_STUDENATA 128
#define MAX_DUZINA 16

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

13 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
15                                 long x)
{
17     /* Ako je indeks elementa na levom kraju veci od indeksa
        elementa na desnom kraju dela niza koji se pretrazuje, onda
19     zapravo pretrazujemo prazan deo niza. U praznom nizu nema
        elementa koji trazimo i zato vracamo -1 */
21     if (levi > desni)
        return -1;
23     /* Racunamo indeks srednjeg elementa */
    int srednji = (levi + desni) / 2;
25     /* Da li je srednji, bas onaj kog trazimo? */
```

```

27     if (a[srednji].indeks == x) {
28         return srednji;
29     }
30     /* Ako je trazeni indeks manji od indeksa srednjeg, onda
31        potragu nastavljamo u levoj polovini niza jer znamo da je
32        niz sortirani po indeksu u rastucem poretku. */
33     if (x < a[srednji].indeks)
34         return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
35     /* Inace ga treba traziti u desnoj polovini */
36     else
37         return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
38 }
39
40 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
41 {
42     /* Ako je niz prazan, vracamo -1, jer ga ne mozemo naci */
43     if (n == 0)
44         return -1;
45     /* Kako trazimo prvog studenta sa trazanim prezimenom,
46        pocinjemo sa prvim studentom u nizu. */
47     if (strcmp(a[0].prezime, x) == 0)
48         return 0;
49     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
50     return i >= 0 ? 1 + i : -1;
51 }
52
53 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
54 {
55     /* Ako je niz prazan, vracamo -1, jer ga ne mozemo naci */
56     if (n == 0)
57         return -1;
58     /* Kako trazimo poslednjeg studenta sa trazanim prezimenom,
59        pocinjemo sa poslednjim studentom u nizu. */
60     if (strcmp(a[n - 1].prezime, x) == 0)
61         return n - 1;
62     return linearna_pretraga_rekurzivna(a, n - 1, x);
63 }
64
65 /* Main funkcija mora imate argumente jer se Ime datoteke dobija
66    kao argument komandne linije */
67 int main(int argc, char *argv[])
68 {
69     /* Ucitacemo redom sve studente iz datoteke u niz. */
70     Student dosije[MAX_STUDENATA];
71     FILE *fin = NULL;
72     int i;
73     int br_studenata = 0;
74     long trazeni_indeks = 0;
75     char trazeno_prezime[MAX_DUZINA];
76
77     /* Proveravamo da li nam je korisnik prilikom poziva prosledio
78        ime datoteke sa informacijama o studentima */

```

```
79     if (argc != 2) {
80         fprintf(stderr,
81             "Greska: Program se poziva sa %s ime_datoteke\n",
82             argv[0]);
83         exit(EXIT_FAILURE);
84     }
85
86     /* Otvaramo datoteku */
87     fin = fopen(argv[1], "r");
88     if (fin == NULL) {
89         fprintf(stderr,
90             "Neuspesno otvaranje datoteke %s za citanje\n",
91             argv[1]);
92         exit(EXIT_FAILURE);
93     }
94
95     /* Citamo sve dok imamo red sa informacijama o studentu */
96     i = 0;
97     while (1) {
98         if (i == MAX_STUDENATA)
99             break;
100         if (fscanf
101             (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
102              dosije[i].prezime) != 3)
103             break;
104         i++;
105     }
106     br_studenata = i;
107
108     /* Nakon citanja datoteka nam vise nije neophodna i odmah je
109        zatvaramo */
110     fclose(fin);
111
112     /* Unos indeksa koji se binarno trazi u nizu */
113     printf("Unesite indeks studenta cije informacije zelite: ");
114     scanf("%ld", &trazen_indeks);
115     i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
116                                     trazen_indeks);
117     if (i == -1)
118         printf("Ne postoji student sa indeksom %ld\n",
119               trazen_indeks);
120     else
121         printf("Indeks: %ld, Ime i prezime: %s %s\n",
122               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
123
124     printf("Unesite prezime studenta cije informacije zelite: ");
125     scanf("%s", trazeno_prezime);
126     i = linearna_pretraga_rekurzivna(dosije, br_studenata,
127                                     trazeno_prezime);
128     if (i == -1)
129         printf("Ne postoji student sa prezimenom %s\n",
130               trazeno_prezime);
```

```

131     else
        printf
            ("Poslednji takav student:\nIndeks: %ld, Ime i prezime: %s %s
133             \n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135     return 0;
}

```

### Rešenje 3.5

```

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

6  /* Struktura koja opisuje tacku u ravni */
typedef struct Tacka {
8      float x;
      float y;
10 } Tacka;

12 /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
      pocetka (0,0) */
14 float rastojanje(Tacka A)
{
16     return sqrt(A.x * A.x + A.y * A.y);
}

18
20 /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u
      nizu zadatih tacaka t dimenzije n */
Tacka najbliza_koordinatnom(Tacka t[], int n)
22 {
    Tacka najbliza;
24     int i;
    najbliza = t[0];
26     for (i = 1; i < n; i++) {
        if (rastojanje(t[i]) < rastojanje(najbliza)) {
28             najbliza = t[i];
        }
30     }
    return najbliza;
32 }

34 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
      tacaka t dimenzije n */
Tacka najbliza_x_osi(Tacka t[], int n)
36 {
38     Tacka najbliza;
40     int i;

```



```
    najbliza = t[0];
42  for (i = 1; i < n; i++) {
    if (fabs(t[i].x) < fabs(najbliza.x)) {
44      najbliza = t[i];
    }
46  }
    return najbliza;
48 }

50 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih
    tacaka t dimenzije n */
52 Tacka najbliza_y_osi(Tacka t[], int n)
{
54     Tacka najbliza;
    int i;
    najbliza = t[0];
56     for (i = 1; i < n; i++) {
58         if (fabs(t[i].y) < fabs(najbliza.y)) {
            najbliza = t[i];
60         }
    }
62     return najbliza;
}

64 #define MAX 1024
66
68 int main(int argc, char *argv[])
{
    FILE *ulaz;
70     Tacka tacke[MAX];
    Tacka najbliza;
72     int i, n;

74     /* Ocekujemo da korisnik unese barem ime izvrsne verzije
        programa i ime datoteke sa tackama */
76     if (argc < 2) {
        fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
        return EXIT_FAILURE;
80     }

82     /* Otvaramo datoteku za citanje */
    ulaz = fopen(argv[1], "r");
84     if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
        return EXIT_FAILURE;
88     }

90     /* Sve dok ima tacaka u datoteci, smestamo ih u niz sa
        tackama; i predstavlja indeks tekuce tacke */
92     i = 0;
```

```

94     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
95         i++;
96     }
97     n = i;
98
99     /* Proveravamo koji su dodatni argumenti komandne linije. Ako
100        nema dodatnih argumenata */
101     if (argc == 2)
102         /* Trazimo najblizu tacku u odnosu na koordinatni pocetak */
103         najbliza = najbliza_koordinatnom(tacke, n);
104     /* Inace proveravamo koji je dodatni argument. Ako je u
105        pitanju opcija -x */
106     else if (strcmp(argv[2], "-x") == 0)
107         /* Racunamo rastojanje u odnosu na x osu */
108         najbliza = najbliza_x_osi(tacke, n);
109     /* Ako je u pitanju opcija -y */
110     else if (strcmp(argv[2], "-y") == 0)
111         /* Racunamo rastojanje u odnosu na y osu */
112         najbliza = najbliza_y_osi(tacke, n);
113     else {
114         /* Ako nije zadata opcija -x ili -y, ispisujemo obavestenje
115            za korisnika i prekidamo izvršavanje programa */
116         fprintf(stderr, "Pogresna opcija\n");
117         return EXIT_FAILURE;
118     }
119
120     /* Stampamo koordinate trazene tacke */
121     printf("%g %g\n", najbliza.x, najbliza.y);
122
123     /* Zatvaramo datoteku */
124     fclose(ulaz);
125
126     return 0;
127 }

```

### 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     /* Posto je u pitanju interval [0, 2] leva granica je 0, a
12        desna 2 */
13     l = 0;
14     d = 2;

```

### 3 Algoritmi pretrage i sortiranja

---

```
15  /* Sve dok ne pronadjemo trazenu vrednost argumenta */
17  while (1) {
18      /* Pronalazimo sredinu intervala */
19      s = (l + d) / 2;
20      /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
21         tacnosti, prekidamo pretragu */
22      if (fabs(cos(s)) < EPS) {
23          break;
24      }
25      /* Ako je nula u levom delu intervala, nastavljamo pretragu
26         na intervalu [l, s] */
27      if (cos(l) * cos(s) < 0)
28          d = s;
29      else
30          /* Inace, nastavljamo pretragu na intervalu [s, d] */
31          l = s;
32  }
33
34  /* Stampamo vrednost trazene tacke */
35  printf("%g\n", s);
36
37  return 0;
38 }
```

#### Rešenje 3.7

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int prvi_veci_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         /* Racunamo sredisnju poziciju */
12         s = (l + d) / 2;
13         /* Ako je broj na toj poziciji veci od nule a eventualni
14            njegov prethodnik manji ili jednak nuli */
15         if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
16             return s;
17         /* Pretrazujemo desnu polovinu niza */
18         if (niz[s] <= 0)
19             l = s + 1;
20         /* Pretrazujemo levu polovinu binarnog zapisa */
21         else
22             d = s - 1;
23     }
24     return -1;
25 }
```

```

}
26
#define MAX 256
28
int main()
30
{
    int niz[MAX];
32    int n = 0;

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

38    /* Stampaње rezultata */
40    printf("%d\n", prvi_veci_od_nule(niz, n));

42    return 0;
}

```

### Rešenje 3.8

```

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

4 int prvi_manji_od_nule(int niz[], int n)
{
    /* Granice pretrage */
    int l = 0, d = n - 1;
    int s;
    /* Sve dok je leva manja od desne granice */
10    while (l <= d) {
        /* Racunamo sredisnju poziciju */
12        s = (l + d) / 2;
        /* Ako je broj na toj poziciji manji od nule a eventualni
           njegov prethodnik veci ili jednak nuli */
14        if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
            return s;
        /* Pretrazujemo desnu polovinu niza */
18        if (niz[s] >= 0)
            l = s + 1;
        /* Pretrazujemo levu polovinu binarnog zapisa */
20        else
            d = s - 1;
22    }
    return -1;
24 }

26 #define MAX 256
28
int main()

```

### 3 Algoritmi pretrage i sortiranja

---

```
30 {
    int niz[MAX];
32     int n = 0;

34     /* Unos niza */
    printf("Unesi opadajuće sortiran niz celih brojeva: ");
36     while (scanf("%d", &niz[n]) == 1)
        n++;
38
    /* Stampanje rezultata */
40     printf("%d\n", prvi_manji_od_nule(niz, n));
42
    return 0;
}
```

#### Rešenje 3.9

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

4 unsigned int logaritam_a(unsigned int x)
{
6     /* Izlaz iz rekurzije */
    if (x == 1)
8         return 0;
    /* Rekurzivni korak */
10    return 1 + logaritam_a(x >> 1);
}

12 unsigned int logaritam_b(unsigned int x)
14 {
    /* Binarnom pretragom trazimo jedinicu u binarnom zapisu broja
16     x najveće važnosti, tj. najlevlju. Pretragu radimo od
        pozicije 0 do 31 */
18    int d = 0, l = sizeof(unsigned int) * 8 - 1;
    int s;
20    /* Sve dok je desna granica pretrage desnije od leve */
    while (d <= l) {
22        /* Racunamo sredisnju poziciju */
        s = (l + d) / 2;
24        /* Proveravamo da li je na toj poziciji trazena jedinica */
        if ((1 << s) <= x && (1 << (s + 1)) > x)
26            return s;
        /* Pretrazujemo desnu polovinu binarnog zapisa */
        if ((1 << s) > x)
28            l = s - 1;
        /* Pretrazujemo levu polovinu binarnog zapisa */
        else
30            d = s + 1;
32    }
34    return s;
}
```

```

}
36
int main()
38 {
    unsigned int x;
40
    /* Unos podatka */
42    printf("Unesi pozitivan ceo broj: ");
    scanf("%u", &x);
44
    /* Provera da li je uneti broj pozitivan */
46    if (x == 0) {
        fprintf(stderr, "Logaritam od nule nije definisan\n");
48        exit(EXIT_FAILURE);
    }
50
    /* Ispis povratnih vrednosti funkcija */
52    printf("%u %u\n", logaritam_a(x), logaritam_b(x));
54
    return 0;
}

```

### Rešenje 3.12

```

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

4  /* Iterativna verzija funkcije koja sortira niz celih brojeva,
    primenom algoritma Selection Sort */
6  void selectionSort(int a[], int n)
    {
8      int i, j;
        int min;
10     int pom;

12     /* U svakoj iteraciji ove petlje se pronalazi najmanji element
        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
        poziciju min, na kojoj se nalazio najmanji od gore
16     navedenih elemenata. */
        for (i = 0; i < n - 1; i++) {
18         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
            nalazi najmanji od elemenata a[i],...,a[n-1]. */
20         min = i;
            for (j = i + 1; j < n; j++)
22             if (a[j] < a[min])
                min = j;
24         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
            samo ako su (i) i min razliciti, inace je nepotrebno. */
26         if (min != i) {
            pom = a[i];

```

### 3 Algoritmi pretrage i sortiranja

---

```
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
46 int main()
47 {
48     int i, a[MAX];
49
50     /* Ucitavaju se elementi niza sve do kraja ulaza */
51     i = 0;
52     printf("Unesite elemente niza: ");
53     while (scanf("%d", &a[i]) != EOF)
54         i++;
55
56     /* Sortiranje */
57     selectionSort(a, i);
58
59     /* Ispis rezultata */
60     printf("%d\n", najmanje_rastojanje(a, i));
61
62     return 0;
63 }
64 }
```

#### Rešenje 3.13

```
#include<stdio.h>
2 #include<string.h>
3
4 #define MAX_DIM 128
5
6 /* Funkcija za sortiranje niza */
7 void selectionSort(char s[], int n)
8 {
9     int i, j, min;
10    char pom;
11    for (i = 0; i < n; i++) {
```

```

12     min = i;
13     for (j = i + 1; j < n; j++)
14         if (s[j] < s[min])
15             min = j;
16     if (min != i) {
17         pom = s[i];
18         s[i] = s[min];
19         s[min] = pom;
20     }
21 }
22 }

24 /* Funkcija vraca: 1 - ako jesu anagrami; 0 - inace.
   pretpostavlja se da su niske s i t sortirane */
26 int anagrami(char s[], char t[], int n_s, int n_t)
27 {
28     int i, n;

30     /* Ako dve niske imaju razlicit broj elemenata onda nisu
       anagrami */
32     if (n_s != n_t)
33         return 0;

34     n = n_s;

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

44 int main()
45 {
46     char s[MAX_DIM], t[MAX_DIM];
47     int n_s, n_t;

48     /* Ucitavamo dve niske sa ulaza */
50     printf("Unesite prvu nisku: ");
51     scanf("%s", s);
52     printf("Unesite drugu nisku: ");
53     scanf("%s", t);

54     /* Odredjujemo duzinu niski */
56     n_s = strlen(s);
57     n_t = strlen(t);

58     /* Sortiramo niske */
60     selectionSort(s, n_s);
61     selectionSort(t, n_t);

62     /* Proveravamo da li su niske anagrami */

```



### 3 Algoritmi pretrage i sortiranja

---

```
64     if (anagrami(s, t, n_s, n_t))
        printf("jesu\n");
66     else
        printf("nisu\n");
68     return 0;
}
```

#### Rešenje 3.14

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

4  /* Funkcija za sortiranje niza */
void selectionSort(int s[], int n)
6  {
    int i, j, min;
    char pom;
    for (i = 0; i < n; i++) {
10         min = i;
        for (j = i + 1; j < n; j++)
12             if (s[j] < s[min])
                min = j;
14         if (min != i) {
            pom = s[i];
16             s[i] = s[min];
            s[min] = pom;
18         }
    }
20 }

22 /* Funkcija za odredjivanje onog elementa sortiranog niza koji
    se najvise puta pojavio u tom nizu */
24 int najvise_puta(int a[], int n)
    {
26         int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
        /* Za i-ti element izracunavamo koliko se puta pojavio u nizu */
28         for (i = 0; i < n; i = j) {
            br_pojava = 1;
30             for (j = i + 1; j < n && a[i] == a[j]; j++)
                br_pojava++;
32             /* Ispitujemo da li se do tog trenutka i-ti element pojavio
                najvise puta u nizu */
34             if (br_pojava > max_br_pojava) {
                max_br_pojava = br_pojava;
36                 i_max_pojava = i;
            }
38         }
        /* Vracamo element koji se najvise puta pojavio u nizu */
40         return a[i_max_pojava];
    }
42 }
```

```

44 int main()
45 {
46     int a[MAX_DIM], i;
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)
52         i++;
53
54     /* Niz se sortira */
55     selectionSort(a, i);
56
57     /* Odredjuje se broj koji se najvise puta pojavio u nizu */
58     printf("%d\n", najvise_puta(a, i));
59
60     return 0;
61 }

```

### Rešenje 3.15

```

#include<stdio.h>
#define MAX_DIM 256

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

/* Funkcija za binarnu pretragu niza. funkcija vraca: 1 - ako se
   element x nalazi u nizu; 0 - inace. pretpostavlja se da je
   niz sortiran u rastucem poretku */
int binarna_pretraga(int a[], int n, int x)
{
    int levi = 0, desni = n - 1, srednji;

    while (levi <= desni) {
        srednji = (levi + desni) / 2;
        if (a[srednji] == x)

```

```
        return 1;
32     else if (a[srednji] > x)
        desni = srednji - 1;
34     else if (a[srednji] < x)
        levi = srednji + 1;
36     }
    return 0;
38 }

40 int main()
{
42     int a[MAX_DIM], n = 0, zbir, i;

44     /* Ucitava se trazeni zbir */
    printf("Unesite trazeni zbir: ");
46     scanf("%d", &zbir);

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

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

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

68     return 0;
}
```

#### Rešenje 3.16

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

4 int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
           int dim3)
6 {
    int i = 0, j = 0, k = 0;
8     /* U slucaju da je dimenzija treceg niza manja od neophodne,
        funkcija vraca -1 */
}
```

```
10     if (dim3 < dim1 + dim2)
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);
56
57     /* Ispis niza */
58     for (k = 0; k < dim3; k++)
59         printf("%d ", niz3[k]);
60     printf("\n");
```

```
62  return 0;
    }
```

#### Rešenje 3.17

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

4
int main(int argc, char *argv[])
6  {
    FILE *fin1 = NULL, *fin2 = NULL;
    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 */
    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
71  /* Ako smo iz prethodne petlje izašli zato što se doslo do
72     kraja druge datoteke, onda ima još imena u prvoj datoteci,
73     i prepisujemo ih, redom, jer su već sortirani po imenu. */
74  while (!kraj1) {
75      fprintf(fout, "%s %s\n", ime1, prezime1);
76      if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
77          kraj1 = 1;
78  }

79
80  /* Ako smo iz prve petlje izašli zato što se doslo do kraja
81     prve datoteke, onda ima još imena u drugoj datoteci, i
82     prepisujemo ih, redom, jer su već sortirani po imenu. */
83  while (!kraj2) {
84      fprintf(fout, "%s %s\n", ime2, prezime2);
85      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
86          kraj2 = 1;
87  }

88
89  /* Zatvaramo datoteke */
90  fclose(fin1);
91  fclose(fin2);
92  fclose(fout);

93
94  return 0;
95 }
```

## Rešenje 3.18

### 3 Algoritmi pretrage i sortiranja

---

```
1  /* Datoteka sort.h */
2  #ifndef __SORT_H__
3  #define __SORT_H__ 1
4
5  /* Selection sort */
6  void selectionsort(int a[], int n);
7  /* Insertion sort */
8  void insertionsort(int a[], int n);
9  /* Bubble sort */
10 void bubblesort(int a[], int n);
11 /* Shell sort */
12 void shellsort(int a[], int n);
13 /* Merge sort */
14 void mergesort(int a[], int l, int r);
15 /* Quick sort */
16 void quicksort(int a[], int l, int r);
17
18 #endif
```

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

```

    samo ako su (i) i min razliciti, inace je nepotrebno. */
34  if (min != i) {
    pom = a[i];
36  a[i] = a[min];
    a[min] = pom;
38  }
}
40
42
44 /* Funkcija sortira niz celih brojeva metodom sortiranja
    umetanjem. Ideja algoritma je sledeca: neka je na pocetku
46 i-te iteracije niz prvih i elemenata (a[0],a[1],...,a[i-1])
    sortirano. U i-toj iteraciji zelimo da element a[i] umetnemo
48 na pravu poziciju medju prvih i elemenata tako da dobijemo
    niz duzine i+1 koji je sortiran. Ovo radimo tako sto i-ti
50 element najpre uporedimo sa njegovim prvim levim susedom
    (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 precu na
    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
    potiskivanje elementa u levo, poredeci ga sa njegovim novim
56 levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
    pravo mesto u nizu. */
58 void insertionsort(int a[], int n)
{
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
            se završava ili kada element dodje do levog kraja (j==0)
70 ili dok ne naidjemo na element a[j-1] koji je manji od
            a[j]. */
72         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

```



### 3 Algoritmi pretrage i sortiranja

---

```
86     poretku. Ovim se najveći element poput mehurica istiskuje na
88     "površinu", tj. na krajnju desnu poziciju. Nakon toga je
    potrebno ovaj postupak ponoviti nad nizom a[0],...,a[n-2],
    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, čime se
    jedan po jedan istiskuju elementi na svoje prave pozicije. */
92 void bubblesort(int a[], int n)
93 {
94     int i, j;
95     int ind;
96
97     for (i = n, ind = 1; i > 1 && ind; i--)
98
99         /* Poput "mehurica" potiskujemo najveći element među
100            elementima od a[0] do a[i-1] na poziciju i-1 upoređujući
101            susedne elemente niza i potiskujući veći u desno */
102         for (j = 0, ind = 0; j < i - 1; j++)
103             if (a[j] > a[j + 1]) {
104                 int temp = a[j];
105                 a[j] = a[j + 1];
106                 a[j + 1] = temp;
107
108                 /* Promenljiva ind registruje da je bilo premestanja.
109                    Samo u tom slučaju ima smisla ici na sledeću
110                    iteraciju, jer ako nije bilo premestanja, znaci da su
111                    svi elementi već u dobrom poretku, pa nema potrebe
112                    prelaziti na kraći prefiks niza. Moglo je naravno i
113                    bez ovoga, algoritam bi radio ispravno, ali bi bio
114                    manje efikasan, jer bi često nepotrebno vrsio mnoga
115                    upoređivanja, kada je već jasno da je sortiranje
116                    završeno. */
117                 ind = 1;
118             }
119     }
120
121     /* Selsort je jednostavno proširenje sortiranja umetanjem koje
122        dopušta direktnu razmenu udaljenih elemenata. Proširenje se
123        sastoji u tome da se kroz algoritam umetanja prolazi više
124        puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
125        koji je manji od n (sto omogućuje 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 početne vrednosti za h, i načina
132        njegovog smanjivanja menja u nekim slučajevima brzinu
133        algoritma, ali bilo koja vrednost će rezultovati ispravnim
134        sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
135        vrednost 1. */
136 void shellsort(int a[], int n)
```

```

138 {
139     int h = n / 2, i, j;
140     while (h > 0) {
141         /* Insertion sort sa korakom h */
142         for (i = h; i < n; i++) {
143             int temp = a[i];
144             j = i;
145             while (j >= h && a[j - h] > temp) {
146                 a[j] = a[j - h];
147                 j -= h;
148             }
149             a[j] = temp;
150         }
151         h = h / 2;
152     }
153 }
154
155 #define MAX 1000000
156
157 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem.
158    Sortiranje se vrši od elementa na poziciji l do onog na
159    poziciji d. Na pocetku, da bismo dobili niz kompletno
160    sortiran, l mora biti 0, a d je jednako poslednjem validnom
161    indeksu u nizu. Funkcija niz podeli na dve polovine, levu i
162    desnu, koje zatim rekurzivno sortira. Od ova dva sortirana
163    podniza, dobijamo sortiran niz ucesljavanjem, tj.
164    istovremenim prolaskom kroz oba niza i izborom trenutnog
165    manjeg elementa koji se smesta u pomocni niz. Na kraju
166    algoritma, sortirani elementi su u pomocnom nizu, koji se
167    kopira u originalni niz. */
168 void mergesort(int a[], int l, int d)
169 {
170     int s;
171     static int b[MAX];          /* Pomocni niz */
172     int i, j, k;
173
174     /* Izlaz iz rekurzije */
175     if (l >= d)
176         return;
177
178     /* Odredjujemo sredisnji indeks */
179     s = (l + d) / 2;
180
181     /* Rekurzivni pozivi */
182     mergesort(a, l, s);
183     mergesort(a, s + 1, d);
184
185     /* Inicijalizacija indeksa. Indeks i prolazi kroz levu
186        polovinu niza, dok indeks j prolazi kroz desnu polovinu
187        niza. Indeks k prolazi kroz pomocni niz b[] */
188     i = l;
189     j = s + 1;

```

```
190 k = 0;
191
192 /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
193 while (i <= s && j <= d) {
194     if (a[i] < a[j])
195         b[k++] = a[i++];
196     else
197         b[k++] = a[j++];
198 }
199
200 /* U slucaju da se prethodna petlja zavrsla izlaskom
201    promenljive j iz dopustenog opsega u pomocni niz
202    prepisujemo ostatak leve polovine niza */
203 while (i <= s)
204     b[k++] = a[i++];
205
206 /* U slucaju da se prethodna petlja zavrsla izlaskom
207    promenljive i iz dopustenog opsega u pomocni niz
208    prepisujemo ostatak desne polovine niza */
209 while (j <= d)
210     b[k++] = a[j++];
211
212 /* Prepisujemo "ucesljani" niz u originalni niz */
213 for (k = 0, i = 1; i <= d; i++, k++)
214     a[i] = b[k];
215
216 /* Funkcija menja mesto i-tom i j-tom elementu niza a */
217 void swap(int a[], int i, int j)
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)
```

```

242     return;

244     /* Particionisanje niza. Svi elementi na pozicijama <=
246     pivot_position (izuzev same pozicije l) su strogo manji od
248     pivota. Kada se pronadje neki element manji od pivota,
250     uvecava se pivot_position i na tu poziciju se premesta
252     nadjeni element. Na kraju ce pivot_position zaista biti
254     pozicija na koju treba smestiti pivot, jer ce svi elementi
256     levo od te pozicije biti manji a desno biti veci ili
258     jednaki od pivota. */
260     pivot_position = l;
262     for (i = l + 1; i <= r; i++)
264         if (a[i] < a[l])
            swap(a, ++pivot_position, i);

    /* Postavljamo pivot na svoje mesto */
    swap(a, l, pivot_position);

    /* Rekurzivno sortiramo elemente manje od pivota */
    quicksort(a, l, pivot_position - 1);
    /* Rekurzivno sortiramo elemente vece pivota */
    quicksort(a, pivot_position + 1, r);
}

```

```

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

/* Maksimalna duzina niza */
#define MAX 1000000

int main(int argc, char *argv[])
{
    /******
12     tip_sortiranja == 0 => selectionsort
        (podrazumevano)
14     tip_sortiranja == 1 => insertionsort
        -i opcija komandne linije
16     tip_sortiranja == 2 => bubblesort
        -b opcija komandne linije
18     tip_sortiranja == 3 => shellsort
        -s opcija komandne linije
20     tip_sortiranja == 4 => mergesort
        -m opcija komandne linije
22     tip_sortiranja == 5 => quicksort
        -q opcija komandne linije
24     *****/
    int tip_sortiranja = 0;
26     /******
        tip_niza == 0 => slucajno generisani nizovi
    */
}

```

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

```

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
    /* 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
    nas niz ne bi uvek isto izgledao seed smo postavili na
    tekuce vreme u sekundama od Nove godine 1970. random()%100
    daje brojeve izmedju 0 i 99 */
    srandom(time(NULL));
98    if (tip_niza == 0)
        for (i = 0; i < dimenzija; i++)
            niz[i] = random();
    else if (tip_niza == 1)
102        for (i = 0; i < dimenzija; i++)
            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 */
    /******
112    Ovaj deo je iskomentaran jer ne zelimo da se sledeci ispis
    nadje na izlazu. Njegova svrha je samo bila provera da li je
114    niz generisan u skladu sa opcijama komandne linije.

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

120

122    /* Sortiramo niz na odgovarajuci nacin */
    if (tip_sortiranja == 0)
        selectionsort(niz, dimenzija);
124    else if (tip_sortiranja == 1)
        insertion sort(niz, dimenzija);
126    else if (tip_sortiranja == 2)
        bubblesort(niz, dimenzija);
128    else if (tip_sortiranja == 3)
        shellsort(niz, dimenzija);
130    else if (tip_sortiranja == 4)

```

### 3 Algoritmi pretrage i sortiranja

```
132     mergesort(niz, 0, dimenzija - 1);
133 else
134     quicksort(niz, 0, dimenzija - 1);

136 /* Ispisujemo elemente niza */
137 /******
138     Ovaj deo je iskomentarisan jer nismo zeleli da vreme potrebno
139     za njegovo izvršavanje 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 }
152 }
```

#### 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 }
```

```
29     for (i = 0; i < n - 1; i++) {
30         min = i;
31         for (j = i + 1; j < n; j++) {
32             if (rastojanje(t[j]) < rastojanje(t[min])) {
33                 min = j;
34             }
35         }
36         if (min != i) {
37             tmp = t[i];
38             t[i] = t[min];
39             t[min] = tmp;
40         }
41     }
42 }
43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
44 void sortiraj_po_x(Tacka t[], int n)
45 {
46     int min, i, j;
47     Tacka tmp;
48
49     for (i = 0; i < n - 1; i++) {
50         min = i;
51         for (j = i + 1; j < n; j++) {
52             if (abs(t[j].x) < abs(t[min].x)) {
53                 min = j;
54             }
55         }
56         if (min != i) {
57             tmp = t[i];
58             t[i] = t[min];
59             t[min] = tmp;
60         }
61     }
62 }
63
64 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
65 void sortiraj_po_y(Tacka t[], int n)
66 {
67     int min, i, j;
68     Tacka tmp;
69
70     for (i = 0; i < n - 1; i++) {
71         min = i;
72         for (j = i + 1; j < n; j++) {
73             if (abs(t[j].y) < abs(t[min].y)) {
74                 min = j;
75             }
76         }
77         if (min != i) {
78             tmp = t[i];
79             t[i] = t[min];
```



```

    t[min] = tmp;
81 }
    }
83 }

85

87 int main(int argc, char *argv[])
{
89     FILE *ulaz;
    FILE *izlaz;
91     Tacka tacke[MAX_BR_TACAKA];
    int i, n;
93
    /* Proveravamo broj argumenata komandne linije: ocekujemo ime
95     izvrsnog programa, opciju, ime ulazne datoteke i ime
        izlazne datoteke tj. ocekujemo 4 argumenta */
97     if (argc != 4) {
        fprintf(stderr,
99             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
        return 0;
101    }

103    /* Otvaramo datoteku u kojoj su zadate tacke */
    ulaz = fopen(argv[2], "r");
105    if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
107                argv[2]);
        return 0;
109    }

111    /* Otvaramo datoteku u koju treba upisati rezultat */
    izlaz = fopen(argv[3], "w");
113    if (izlaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
115                argv[3]);
        return 0;
117    }

119    /* Sve dok ne stignemo do kraja ulazne datoteke učitavamo
        koordinate tacaka i smestamo ih na odgovarajucu poziciju
121        odredjenu brojacem i; prilikom učitavanja oslanjamo se na
        svojstvo funkcije fscanf povratka EOF vrednosti kada stigne
123        do kraja ulaza */
    i = 0;
125    while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
        i++;
127    }

129    /* Cuvamo broj procitanih tacaka */
    n = i;
131
```

```

133  /* Analiziramo zadatu opciju: kako ocekujemo da je argv[1]
      "-x" ili "-y" ili "-o" sigurni smo da je argv[1][0] crtica
135      (karakter -) i dalje proveravamo sta je na sledecoj
      poziciji tj. sta je argv[1][1] */

137  switch (argv[1][1]) {
      case 'x':
139      /* Ako je u pitanju karakter x, pozivamo funkciju za
          sortiranje po vrednosti x koordinate */
141      sortiraj_po_x(tacke, n);
          break;
143      case 'y':
          /* Ako je u pitanju karakter y, pozivamo funkciju za
145          sortiranje po vrednosti y koordinate */
          sortiraj_po_y(tacke, n);
          break;
147      case 'o':
          /* Ako je u pitanju karakter o, pozivamo funkciju za
149          sortiranje po udaljenosti od koorinatnog pocetka */
          sortiraj_po_rastojanju(tacke, n);
          break;
153  }

155  /* Upisujemo dobijeni niz u izlaznu datoteku */
      for (i = 0; i < n; i++) {
157      fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
      }

159

161  /* Zatvaramo otvorene datoteke */
      fclose(ulaz);
      fclose(izlaz);

163

165  /* Završavamo sa programom */
      return 0;
  }

```

### 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;

```

### 3 Algoritmi pretrage i sortiranja

---

```
14 /* Funkcija sortira niz gradjana rastuce po imenima */
void sort_ime(Gradjanin a[], int n)
16 {
    int i, j;
    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];
            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;
    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;
            for (j = i + 1; j < n; j++)
50                 if (strcmp(a[j].prezime, a[min].prezime) < 0)
52                     min = j;
54         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
            samo ako su (i) i min razliciti, inace je nepotrebno. */
            if (min != i) {
56                 pom = a[i];
                    a[i] = a[min];
                    a[min] = pom;
58             }
60     }
    }

62 /* Pretraga niza Gradjana */
64 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
{
```

```

66     int i;
67     for (i = 0; i < n; i++)
68         if (strcmp(a[i].ime, x->ime) == 0
69             && strcmp(a[i].prezime, x->prezime) == 0)
70             return i;
71     return -1;
72 }
73
74
75 int main()
76 {
77     Gradjanin spisak1[MAX], spisak2[MAX];
78     int isti_rbr = 0;
79     int i, n;
80     FILE *fp = NULL;
81
82     /* Otvaranje datoteke */
83     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
84         fprintf(stderr,
85             "Neuspesno otvaranje datoteke biracki-spisak.txt.\n");
86         exit(EXIT_FAILURE);
87     }
88
89     /* Citanje sadrzaja */
90     for (i = 0;
91          fscanf(fp, "%s %s", spisak1[i].ime,
92                spisak1[i].prezime) != EOF; i++)
93         spisak2[i] = spisak1[i];
94     n = i;
95
96     /* Zatvaranje datoteke */
97     fclose(fp);
98
99     sort_ime(spisak1, n);
100
101     /******
102     Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
103     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
104
105     printf("Biracki spisak [uredjen prema imenima]:\n");
106     for(i=0; i<n; i++)
107         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
108     *****/
109
110     sort_prezime(spisak2, n);
111
112     /******
113     Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
114     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
115
116     printf("Biracki spisak [uredjen prema prezimenima]:\n");
117     for(i=0; i<n; i++)

```

### 3 Algoritmi pretrage i sortiranja

```
118     printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
119     *****/
120
121     /* Linearno pretrazivanje nizova */
122     for (i = 0; i < n; i++)
123         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
124             isti_rbr++;
125
126     /* Alternativno (efikasnije) resenje */
127     *****/
128     for(i=0; i<n ;i++)
129         if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
130             strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
131             isti_rbr++;
132     *****/
133
134     /* Ispis rezultata */
135     printf("%d\n", isti_rbr);
136
137     exit(EXIT_SUCCESS);
138 }
```

#### Rešenje 3.22

```
#include <stdio.h>
2 #include <string.h>
#include <ctype.h>
4
#define MAX_BR_RECI 128
6 #define MAX_DUZINA_RECI 32
8
/* Funkcija koja izracunava broj suglasnika u reci */
10 int broj_suglasnika(char s[])
{
12     char c;
13     int i;
14     int suglasnici = 0;
15     /* Obilazimo karakter po karakter zadate niske */
16     for (i = 0; s[i]; i++) {
17         /* Ako je u pitanju slovo */
18         if (isalpha(s[i])) {
19             /* Pretvaramo ga u veliko da bismo mogli da pokrijemo
20                slucaj i malih i velikih suglasnika */
21             c = toupper(s[i]);
22             /* Ukoliko slovo nije samoglasnik */
23             if (c != 'A' && c != 'E' && c != 'I' && c != 'O'
24                 && c != 'U') {
25                 /* Uvecavamo broj suglasnika */
26                 suglasnici++;
27             }
28         }
29     }
30 }
```

```
28     }
29 }
30 /* Vracamo izracunatu vrednost */
31 return suglasnici;
32 }
33
34 /* Funkcija koja sortira reci po zadatom kriterijumu.
35    Informacija o duzini reci se mora proslediti zbog pravilnog
36    upravljanja memorijom */
37 void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)
38 {
39     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
40         duzina_j, duzina_min;
41     char tmp[MAX_DUZINA_RECII];
42     for (i = 0; i < n - 1; i++) {
43         min = i;
44         for (j = i; j < n; j++) {
45             /* Prvo uporedjujemo broj suglasnika */
46             broj_suglasnika_j = broj_suglasnika(reci[j]);
47             broj_suglasnika_min = broj_suglasnika(reci[min]);
48             if (broj_suglasnika_j < broj_suglasnika_min)
49                 min = j;
50             else if (broj_suglasnika_j == broj_suglasnika_min) {
51                 /* Zatim, reci imaju isti broj suglasnika uporedjujemo
52                    duzine */
53                 duzina_j = strlen(reci[j]);
54                 duzina_min = strlen(reci[min]);
55
56                 if (duzina_j < duzina_min)
57                     min = j;
58                 else
59                     /* A ako reci imaju i isti broj suglasnika i iste
60                        duzine, uporedjujemo ih leksikografski */
61                     if (duzina_j == duzina_min
62                         && strcmp(reci[j], reci[min]) < 0)
63                         min = j;
64             }
65         }
66         if (min != i) {
67             strcpy(tmp, reci[min]);
68             strcpy(reci[min], reci[i]);
69             strcpy(reci[i], tmp);
70         }
71     }
72 }
73
74 int main()
75 {
76     FILE *ulaz;
77     int i = 0, n;
```

### 3 Algoritmi pretrage i sortiranja

```
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 korisćenog 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    /* Završavamo sa programom */
    return 0;
120 }
```

#### 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 proizvodjac[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 vracamo -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++) {
        min = i;
        for (j = i + 1; j < n; j++)
51             if (a[j].kod < a[min].kod)
53                 min = j;
        if (min != i) {
55             pom = a[i];
            a[i] = a[min];
57             a[min] = pom;
        }
59     }
```



```

    }
61 }

63 int main()
{
65     Artikal asortiman[MAX_ARTIKALA];
    long kod;
67     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 sortirani 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
       trazanja 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++)
112         printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113             asortiman[i].naziv, asortiman[i].proizvodjac,
114             asortiman[i].cena);
115
116     kod = 0;
117     while (1) {
118         printf("-----\n");
119         printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
120         printf("- Za nov racun unesite kod artikla!\n\n");
121         /* Unos bar koda provog artikla sledeceg kupca */
122         if (scanf("%ld", &kod) == EOF)
123             break;
124         /* Trenutno racun novog kupca */
125         racun = 0;
126         /* Za sve artikle trenutnog kupca */
127         while (1) {
128             /* Nalazimo ih u nizu */
129             if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
130                 printf
131                 ("\tGRESKA: Ne postoji proizvod sa trazanim kodom!\n");
132             } else {
133                 printf("\tTrazili ste:\t%s %s %12.2f\n",
134                     asortiman[i].naziv, asortiman[i].proizvodjac,
135                     asortiman[i].cena);
136                 /* I 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("\tUnesite 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: %.21f dinara.\n\n", racun);
148     }
149
150     printf("Kraj rada kase!\n");
151
152     exit(EXIT_SUCCESS);
153 }

```

### Rešenje 3.24

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 500

```

```
6
8  /* Struktura koja nam je neophodna za sve informacije o
   pojedinačnom studentu */
typedef struct {
10     char ime[20];
    char prezime[25];
12     int prisustvo;
    int zadaci;
14 } Student;

16 /* Funkcija kojom sortiramo niz struktura po prezimenu
   leksikografski rastuće */
18 void sort_ime_leksikografski(Student niz[], int n)
{
20     int i, j;
    int min;
22     Student pom;

24     for (i = 0; i < n - 1; i++) {
        min = i;
26         for (j = i + 1; j < n; j++)
            if (strcmp(niz[j].ime, niz[min].ime) < 0)
28             min = j;

30         if (min != i) {
            pom = niz[min];
32             niz[min] = niz[i];
            niz[i] = pom;
34         }
    }
36 }

38 /* Funkcija kojom sortiramo niz struktura po ukupnom broju
   uradjenih zadataka opadajuće, ukoliko neki studenti imaju
   isti broj uradjenih zadataka sortiraju se po dužini imena
   rastuće. */
40 void sort_zadatke_pa_imena(Student niz[], int n)
{
42     int i, j;
    int max;
44     Student pom;

46     for (i = 0; i < n - 1; i++) {
        max = i;
48         for (j = i + 1; j < n; j++)
            if (niz[j].zadaci > niz[max].zadaci)
50                 max = j;
            else if (niz[j].zadaci == niz[max].zadaci
52                     && strlen(niz[j].ime) < strlen(niz[max].ime))
                max = j;
54         if (max != i) {
            pom = niz[max];
56             niz[max] = niz[i];
            niz[i] = pom;
        }
    }
}
```

```

58     niz[i] = pom;
59 }
60 }
61 }
62
63 /* Funkcija kojom sortiramo niz struktura po broju casova na
64    kojima su bili opadajuće, a ukoliko * neki studenti imaju
65    isti broj casova, sortiraju se opadajuće po broju uradjenih
66    zadataka, * a ukoliko se i po broju zadataka poklapaju
67    sortirati ih po prezimenu opadajuće. */
68 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
69 {
70     int i, j;
71     int max;
72     Student pom;
73     for (i = 0; i < n - 1; i++) {
74         max = i;
75         for (j = i + 1; j < n; j++)
76             if (niz[j].prisustvo > niz[max].prisustvo)
77                 max = j;
78             else if (niz[j].prisustvo == niz[max].prisustvo
79                     && niz[j].zadaci > niz[max].zadaci)
80                 max = j;
81             else if (niz[j].prisustvo == niz[max].prisustvo
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 int main(int argc, char *argv[])
95 {
96     Student praktikum[MAX];
97     int i, br_studenata = 0;
98
99     FILE *fp = NULL;
100
101     /* Otvaranje datoteke za citanje */
102     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
103         fprintf(stderr,
104             "Neuspesno otvaranje datoteke aktivnost.txt.\n");
105         exit(EXIT_FAILURE);
106     }
107
108     /* Ucitavanje sadrzaja */

```

### 3 Algoritmi pretrage i sortiranja

---

```
110     for (i = 0;
111           fscanf(fp, "%s%s%d%d", praktikum[i].ime,
112                 praktikum[i].prezime, &praktikum[i].prisustvo,
113                 &praktikum[i].zadaci) != EOF; i++);
114 /* Zatvaranje datoteke */
115 fclose(fp);
116 br_studenata = i;
117
118 /* Kreiramo prvi spisak studenata na kom su sortirani
119    leksikografski po imenu rastuce */
120 sort_ime_leksikografski(praktikum, br_studenata);
121 /* Otvaranje datoteke za pisanje */
122 if ((fp = fopen("dat1.txt", "w")) == NULL) {
123     fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
124     exit(EXIT_FAILURE);
125 }
126 /* Upis niza u datoteku */
127 fprintf
128     (fp,
129      "Studenti sortirani po imenu leksikografski rastuce:\n");
130 for (i = 0; i < br_studenata; i++)
131     fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
132           praktikum[i].prezime, praktikum[i].prisustvo,
133           praktikum[i].zadaci);
134 /* Zatvaranje datoteke */
135 fclose(fp);
136
137 /* Na drugom su sortirani po ukupnom broju uradjenih zadataka
138    opadajuce, ukoliko neki studenti imaju isti broj uradjenih
139    zadataka sortiraju se po duzini imena rastuce. */
140 sort_zadatke_pa_imena(praktikum, br_studenata);
141 /* Otvaranje datoteke za pisanje */
142 if ((fp = fopen("dat2.txt", "w")) == NULL) {
143     fprintf(stderr, "Neuspesno otvaranje datoteke dat2.txt.\n");
144     exit(EXIT_FAILURE);
145 }
146 /* Upis niza u datoteku */
147 fprintf(fp,
148         "Studenti sortirani po broju zadataka opadajuce,\n");
149 fprintf(fp, "pa po duzini imena rastuce:\n");
150 for (i = 0; i < br_studenata; i++)
151     fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
152           praktikum[i].prezime, praktikum[i].prisustvo,
153           praktikum[i].zadaci);
154 /* Zatvaranje datoteke */
155 fclose(fp);
156
157 /* Na trecem spisku su sortirani po broju casova na kojima su
158    bili opadajuce, a ukoliko neki studenti imaju isti broj
159    casova, sortiraju se opadajuce po broju uradjenih zadataka,
160    a ukoliko se i po broju zadataka poklapaju sortirati ih po
161    prezimenu opadajuce. */
```

```

162     sort_prisustvo_pa_zadatke_pa_prezimana(praktikum,
                                              br_studenata);
164     /* Otvaranje datoteke za pisanje */
165     if ((fp = fopen("dat3.txt", "w")) == NULL) {
166         fprintf(stderr, "Neuspješno otvaranje datoteke dat3.txt.\n");
167         exit(EXIT_FAILURE);
168     }
169     /* Upis niza u datoteku */
170     fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
171     fprintf(fp, "pa po broju zadataka,\n");
172     fprintf(fp, "pa po prezimenima leksikografski opadajuće:\n");
173     for (i = 0; i < br_studenata; i++)
174         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
175                 praktikum[i].prezime, praktikum[i].prisustvo,
176                 praktikum[i].zadaci);
177     /* Zatvaranje datoteke */
178     fclose(fp);
179
180     return 0;
181 }

```

### Rešenje 3.25

```

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

6 /* Struktura koja opisuje jednu pesmu */
typedef struct {
8     char *izvodjac;
    char *naslov;
10    int broj_gledanja;
} Pesma;

12
13 /* Funkcija za upoređivanje 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 upoređivanje 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;

```

```
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
51                             informacije citaju */
52     int n;                  /* Ukupan broj pesama */
53     int j, k;
54     char c;
55     int alocirano;          /* Broj mesta alociranih za
56                             propratne informacije o
57                             pesmama */
58     int broj_gledanja;
59
60     /* Pripremamo datoteku za citanje */
61     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
62     if (ulaz == NULL) {
63         printf("Greska pri otvaranju ulazne datoteke!\n");
64         return 0;
65     }
66
67     /* Citamo informacije o pesmama */
68     pesme = NULL;
69     alocirano_za_pesme = 0;
70     i = 0;
71
72     while (1) {
73
74         /* Proveravamo da li smo stigli do kraja datoteke */
75         c = fgetc(ulaz);
76         if (c == EOF) {
77             /* Ako smo dobili kao rezultat EOF, jesmo, nema vise
78                sadrzaja za citanje */
79             break;
80         }
```

```
82 } else {
    /* Ako nismo, vracamo procitani karakter nazad */
    ungetc(c, ulaz);
84 }

86
88 /* Proveravamo da li imamo dovoljno memorije za citanje nove
    pesme */
    if (alocirano_za_pesme == i) {
90
92         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
            alociramo novih KORAK mesta */
            alocirano_za_pesme += KORAK;
94         pesme =
            (Pesma *) realloc(pesme,
96                             alocirano_za_pesme * sizeof(Pesma));

98         /* Proveravamo da li je nova memorija uspesno realocirana */
            if (pesme == NULL) {
100             /* Ako nije ... */
            /* Ispisujemo obavestenje */
102             printf("Problem sa alokacijom memorije!\n");

104             /* I oslobadjamo svu memoriju zauzetu do ovog koraka */
            for (k = 0; k < i; k++) {
106                 free(pesme[k].izvodjac);
                free(pesme[k].naslov);
108             }
            free(pesme);
110             return 0;
        }
112    }

114
116    /* Ako jeste, nastavljamo sa citanjem pesama ... */
    /* Citamo ime izvodjaca */

118    j = 0;                                /* Oznacava poziciju na koju
                                           treba smestiti procitani
                                           karakter */
120
    alocirano = 0;                          /* Oznacava broj alociranih
                                           mesta */
122
    pesme[i].izvodjac = NULL;               /* Memorija koju mozemo
                                           koristiti za smestanje
                                           procitanih karaktera */
124

126    /* Sve dok ne stignemo do prve beline u liniji - beline koja
        se nalazi nakon imena izvodjaca - citamo karaktere iz
        datoteke */
128    while ((c = fgetc(ulaz)) != ' ') {
130
132        /* Proveravamo da li imamo dovoljno memorije za smestanje
```



```
134     procitanog karaktera */
135     if (j == alocirano) {
136         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
137            alociramo novih KORAK mesta */
138         alocirano += KORAK;
139         pesme[i].izvodjac =
140             (char *) realloc(pesme[i].izvodjac,
141                             alocirano * sizeof(char));
142
143         /* Proveravamo da li je nova alokacija uspesna */
144         if (pesme[i].izvodjac == NULL) {
145             /* Ako nije... */
146             /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
147             for (k = 0; k < i; k++) {
148                 free(pesme[k].izvodjac);
149                 free(pesme[k].naslov);
150             }
151             free(pesme);
152             /* I prekidamo sa izvršavanjem programa */
153             return 0;
154         }
155     }
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;               /* Memoriya koju mozemo
184                                         koristiti za smestanje
185                                         procitanih karaktera */
```

```
186 /* Sve dok ne stignemo do zareza - zareza koji se nalazi
    nakon naslova pesme - citamo karaktere iz datoteke */

188 while ((c = fgetc(ulaz)) != ',') {
    /* Proveravamo da li imamo dovoljno memorije za smestanje
190      procitanog karaktera */
    if (j == alocirano) {
192      /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
        alociramo novih KORAK mesta */
194      alocirano += KORAK;
      pesme[i].naslov =
196        (char *) realloc(pesme[i].naslov,
                          alocirano * sizeof(char));

198
      /* Proveravamo da li je nova alokacija uspesna */
200      if (pesme[i].naslov == NULL) {
        /* Ako nije... */
202        /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
        for (k = 0; k < i; k++) {
204          free(pesme[k].izvodjac);
          free(pesme[k].naslov);
206        }
        free(pesme[i].izvodjac);
208        free(pesme);

210        /* I prekidamo izvršavanje programa */
        return 0;
212      }

214    }
    /* Ako imamo dovoljno memorije, smestamo procitani
216      karakter */
    pesme[i].naslov[j] = c;
218    j++;
    /* I nastavljamo dalje sa citanjem */
220  }
  /* Upisujemo terminirajucu nulu na kraju reci */
222  pesme[i].naslov[j] = '\0';

224  /* Citamo razmak */
  fgetc(ulaz);
226

228  /* Citamo broj gledanja */

230  broj_gledanja = 0;

232  /* Sve dok ne stignemo do znaka za novi red - kraja linije -
    citamo karaktere iz datoteke */
234  while ((c = fgetc(ulaz)) != '\n') {
    broj_gledanja = broj_gledanja * 10 + (c - '0');
236  }
```

### 3 Algoritmi pretrage i sortiranja

---

```
238     pesme[i].broj_gledanja = broj_gledanja;
240     /* Prelazimo na citanje sledece pesme */
242     i++;
244     /* Cuvamo informaciju o broju pesama koje smo procitali */
246     n = i;
248     /* Zatvaramo datoteku jer nam nece vise trebati */
250     fclose(ulaz);
252     /* Analiziramo argumente komandne linije */
254     if (argc == 1) {
256         /* Nema dodatnih opcija - sortiramo po broju gledanja */
258         qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
260     } else {
262         if (argc == 2 && strcmp(argv[1], "-n") == 0) {
264             /* Sortiramo po naslovu */
266             qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
268         } else {
270             if (argc == 2 && strcmp(argv[1], "-i") == 0) {
272                 /* Sortiramo po izvodjacu */
274                 qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
276             } else {
278                 printf("Nedozvoljeni argumenti!\n");
280                 free(pesme);
282                 return 0;
284             }
286         }
288     }
290     /* Ispisujemo rezultat */
292     for (i = 0; i < n; i++) {
294         printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
296             pesme[i].broj_gledanja);
298     }
300     /* Oslobadjamo memoriju */
302     for (i = 0; i < n; i++) {
304         free(pesme[i].izvodjac);
306         free(pesme[i].naslov);
308     }
310     free(pesme);
312     /* Prekidamo izvršavanje programa */
314     return 0;
316 }
```

## 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)
22         return 1;
23     else if (b1 < b2)
24         /* Ovo uredjenje favorizujemo jer zelimo rastuci poredak */
25         return -1;
26     else
27         return 0;
28 }
29
30 int compare_int_desc(const void *a, const void *b)
31 {
32     /* Za obrnuti poredak mozemo samo oduzimati a od b */
33     /* return *((int *)b) - *((int *)a); */
34
35     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
36     funkcija */
37     return -compare_int(a, b);
38 }
39
40 /* Test program */
41 int main()
42 {
43     size_t n;
44     int i, x;
45     int a[MAX], *p = NULL;
46
47     /* Unosimo dimenziju */
48     printf("Uneti dimenziju niza: ");
49     scanf("%ld", &n);
50     if (n > MAX)
```

### 3 Algoritmi pretrage i sortiranja

```
51     n = MAX;

53     /* Unosimo elemente niza */
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);

61     /* Prikazujemo sortirani niz */
printf("Sortirani niz u rastucem poretku:\n");
63     for (i = 0; i < n; i++)
        printf("%d ", a[i]);
65     putchar('\n');

67     /* 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");
    p = bsearch(&x, a, n, sizeof(int), &compare_int);
75     if (p == NULL)
        printf("Elementa nema u nizu!\n");
77     else
        printf("Element je nadjen na poziciji %ld\n", p - a);

79     /* Linearna pretraga */
81     printf("Linearna pretraga (lfind): \n");
    p = lfind(&x, a, &n, sizeof(int), &compare_int);
83     if (p == NULL)
        printf("Elementa nema u nizu!\n");
85     else
        printf("Element je nadjen na poziciji %ld\n", p - a);

87     return 0;
89 }
```

#### Rešenje 3.29

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

6 #define MAX 100

8 /* Funkcija racuna broj delilaca broja x */
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: ");
```

### 3 Algoritmi pretrage i sortiranja

```
62  scanf("%ld", &n);
    if (n > MAX)
64      n = MAX;

    /* Unosimo elemente niza */
    printf("Uneti elemente niza:\n");
68  for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

70

    /* Sortiramo niz celih brojeva prema broju delilaca */
72  qsort(a, n, sizeof(int), &compare_no_deviders);

74  /* Prikazujemo sortirani niz */
    printf
76      ("Sortirani niz u rastucem poretku prema broju delilaca:\n");
    for (i = 0; i < n; i++)
78        printf("%d ", a[i]);
    putchar('\n');

80
    return 0;
82 }
```

#### Rešenje 3.30

```
#include <stdio.h>
2  #include <stdlib.h>
   #include <string.h>
4  #include <search.h>
   #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 veka 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 porecenu
   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 slična prethodnoj, osim što elemente ne upoređuje
    leksikografski, već po dužini */
34 int poredi_duzine(const void *a, const void *b)
{
36     return strlen((char *) a) - strlen((char *) b);
}

38
int main()
40 {
    int i;
    size_t n;
    FILE *fp = NULL;
    char niske[MAX_NISKI][MAX_DUZINA];
    char *p = NULL;
    char x[MAX_DUZINA];

    /* Otvaranje datoteke */
    if ((fp = fopen("niske.txt", "r")) == NULL) {
    50     fprintf(stderr, "Neuspješno 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 */
    fclose(fp);
    n = i;

    60
    /* Sortiramo niske leksikografski, tako što 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 traženi nisku */
    printf("Uneti traženu nisku: ");
    74 scanf("%s", x);

    76
    /* Binarna pretraga */
    /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
    78     jer nam je niz sortirani leksikografski. */
    p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),

```



### 3 Algoritmi pretrage i sortiranja

```
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
        printf("Niska nije pronadjena u nizu\n");

88     /* Linearna pretraga */
    /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
90     jer nam je niz sortirani 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
        printf("Niska nije pronadjena u nizu\n");

100    /* Sada ih sortiramo po duzini 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

8 /******
   Niz pokazivaca na karaktere ovog potpisa
10 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  -----
20  Sa leve strane je vertikalno prikazan niz pokazivaca, gde
    je i-ti njegov element pokazivac koji pokazuje na alocirane
    karaktere i-te reci. Njegov tip je char*.

22
24  Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
    elemenata koji trebaju biti uporedjeni (recimo adresu od X
    i adresu od Z), i kako su X i Z tipa char*, onda a i b su
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
38  /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
    leksikografski, vec po duzini */
    int poredi_duzine(const void *a, const void *b)
40  {
42      return strlen(*(char **) a) - strlen(*(char **) b);
    }

44  /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje
    na element u nizu sa kojim se poredi, pa njega treba
46  kastovati na char** i dereferencirati, (videti obrazlozenje
    za prvu funkciju u ovom zadatku, a pokazivac a pokazuje na
    element koji se trazi. U main funkciji je to x, koji je tipa
48  char*, tako da pokazivac a ovde samo kastujemo i ne
    dereferenciramo. */
    int poredi_leksikografski_b(const void *a, const void *b)
50  {
52      return strcmp((char *) a, *(char **) b);
    }

54
56  int main()
    {
58      int i;
      size_t n;
      FILE *fp = NULL;
      char *niske[MAX_NISKI];
      char **p = NULL;
      char x[MAX_DUZINA];

64      /* Otvaranje datoteke */
      if ((fp = fopen("niske.txt", "r")) == NULL) {
66          fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
          exit(EXIT_FAILURE);
68      }
    }

```

### 3 Algoritmi pretrage i sortiranja

---

```
70  /* Citanje sadrzaja datoteke */
72  i = 0;
74  while (fscanf(fp, "%s", x) != EOF) {
76      /* Alociranje dovoljne memorije za i-tu nisku */
78      if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
80          fprintf(stderr, "Greska pri alociranju niske\n");
82          exit(EXIT_FAILURE);
84      }
86      /* Kopiranje procitane niske na svoje mesto */
88      strcpy(niske[i], x);
90      i++;
92  }

94  /* Zatvaranje datoteke */
96  fclose(fp);
98  n = i;

100 /* Sortiramo niske leksikografski, tako sto biblioteckoj
102     funkciji qsort prosledjujemo funkciju kojom se zadaje
104     kriterijum poredjenja 2 niske po duzini */
106 qsort(niske, n, sizeof(char *), &poredi_leksikografski);

108 printf("Leksikografski sortirane niske:\n");
110 for (i = 0; i < n; i++)
112     printf("%s ", niske[i]);
114 printf("\n");

116 /* Unosimo trazeni nisku */
118 printf("Uneti trazenu nisku: ");
120 scanf("%s", x);

122 /* Binarna pretraga */
124 /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
126     jer nam je niz sortiran leksikografski. */
128 p = bsearch(x, niske, n, sizeof(char *),
130             &poredi_leksikografski_b);

132 if (p != NULL)
134     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
136           *p, p - niske);
138 else
140     printf("Niska nije pronadjena u nizu\n");

142 /* Linearna pretraga */
144 /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
146     jer nam je niz sortiran leksikografski. */
148 p = lfind(x, niske, &n, sizeof(char *),
150           &poredi_leksikografski_b);

152 if (p != NULL)
154     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
```

```

122         *p, p - niske);
123     else
124         printf("Niska nije pronadjena u nizu\n");
125
126     /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
127        funkciju poredjenja */
128     qsort(niske, n, sizeof(char *), &poredi_duzine);
129
130     printf("Niske sortirane po duzini:\n");
131     for (i = 0; i < n; i++)
132         printf("%s ", niske[i]);
133     printf("\n");
134
135     /* Oslobadjanje zauzete memorije */
136     for (i = 0; i < n; i++)
137         free(niske[i]);
138
139     exit(EXIT_SUCCESS);
140 }

```

### Rešenje 3.32

```

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

6 #define MAX 500

8 /* Struktura koja nam je neophodna za sve informacije o
   pojedinacnom studentu */
10 typedef struct {
11     char ime[21];
12     char prezime[21];
13     int bodovi;
14 } Student;

16 /* Funkcija poredjenja koju cemo koristiti za sortiranje po
   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;

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
30     prezimenu */
31     return strcmp(prvi->prezime, drugi->prezime);
32 }

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 /* Funkcija za poredjenje koje ce porediti samo po prezimenu
45     Prvi parametar je ono sto trazimo u nizu, ovde je to prezime,
46     * a drugi parametar ce biti element niza cije se prezime
47     poredi. */
48 int compare_zalinearnaprezimena(const void *a, const void *b)
49 {
50     char *prezime = (char *) a;
51     Student *s = (Student *) b;
52     return strcmp(prezime, s->prezime);
53 }

54 }

56 int main(int argc, char *argv[])
57 {
58     Student kolokvijum[MAX];
59     int i;
60     size_t br_studenata = 0;
61     Student *nadjen = NULL;
62     FILE *fp = NULL;
63     int bodovi;
64     char prezime[21];

65     /* Ako je program pozvan sa nedovoljnim brojem argumenata
66         informisemo korisnika kako se program koristi i prekidamo
67         izvorsavanje. */
68     if (argc < 2) {
69         fprintf(stderr,
70             "Program se poziva sa:\n%s datoteka_sarezultatima\n",
71             argv[0]);
72         exit(EXIT_FAILURE);
73     }

74     /* Otvaranje datoteke */
75     if ((fp = fopen(argv[1], "r")) == NULL) {
76         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
77         exit(EXIT_FAILURE);
78     }
79 }
```

```
82  /* Ucitavanje sadrzaja */
84  for (i = 0;
      fscanf(fp, "%s%s%d", kolokvijum[i].ime,
86          kolokvijum[i].prezime,
          &kolokvijum[i].bodovi) != EOF; i++);
88
90  /* Zatvaranje datoteke */
92  fclose(fp);
94  br_studenata = i;
96
98  /* Sortiramo niz studenata po broju bodova, pa unutar grupe
100     studenata sa istim brojem bodova sortiranje se vrši po
        prezimenu */
102  qsort(kolokvijum, br_studenata, sizeof(Student), &compare);
104
106  printf("Studenti sortirani po broju poena opadajuće, ");
108  printf("pa po prezimenu rastuće:\n");
110  for (i = 0; i < br_studenata; i++)
112      printf("%s %s %d\n", kolokvijum[i].ime,
          kolokvijum[i].prezime, kolokvijum[i].bodovi);
114
116  /* Pretraživanje studenata po broju bodova se vrši binarnom
        pretragom jer je niz sortiran po broju bodova. */
118  printf("Unesite broj bodova: ");
120  scanf("%d", &bodovi);
122
124  nadjen =
126      bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
          &compare_za_bsearch);
128
130  if (nadjen != NULL)
132      printf
          ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
          nadjen->ime, nadjen->prezime, nadjen->bodovi);
134  else
136      printf("Nema studenta sa unetim brojem bodova\n");
138
140  /* Pretraga po prezimenu se mora vršiti linearnom pretragom
        jer nam je niz sortiran po bodovima, globalno gledano. */
142  printf("Unesite prezime: ");
144  scanf("%s", prezime);
146
148  nadjen =
150      lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
          &compare_za_linearna_prezimana);
152
154  if (nadjen != NULL)
156      printf
          ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
          nadjen->ime, nadjen->prezime, nadjen->bodovi);
158  else
```

### 3 Algoritmi pretrage i sortiranja

---

```
134     printf("Nema studenta sa unetim prezimenom\n");
136     return 0;
}
```

#### 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 vraca: 1 - ako jesu anagrami 0 - inace */
   int anagrami(char s[], char t[], int n_s, int n_t)
15 {
       /* Ako dve niske imaju razlicitu duzinu => nisu anagrami */
17     if (n_s != n_t)
         return 0;

19     /* Sortiramo niske */
21     qsort(s, strlen(t) / sizeof(char), sizeof(char),
           &uporedi_char);
23     qsort(t, strlen(t) / sizeof(char), sizeof(char),
           &uporedi_char);

25     /* Ako su niske nakon sortiranja iste => jesu anagrami, u
       suprotnom, nisu */
27     return !strcmp(s, t);
29 }

31 int main()
   {
33     char s[MAX], t[MAX];

35     /* Unose se dve niske sa ulaza */
   printf("Unesite dve niske: ");
37     scanf("%s %s", s, t);

39     /* Ispituje se da li su niske anagrami */
   if (anagrami(s, t, strlen(s), strlen(t)))
41     printf("jesu\n");
   else
43     printf("nisu\n");
}
```

```

45     return 0;
}

```

### Rešenje 3.34

```

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

#define MAX 10
#define MAX_DUZINA 32

/* Funkcija poredi dve niske (stringove) */
int uporedi_niske(const void *pa, const void *pb)
{
    return strcmp((char *) pa, (char *) pb);
}

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

    /* Unos broja niski */
    printf("Unesite broj niski:");
    scanf("%d", &n);

    /* Unos niza niski */
    printf("Unesite niske:\n");
    for (i = 0; i < n; i++)
        scanf("%s", S[i]);

    /* Sortiramo niz niski */
    qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

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

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

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

```



```
48     printf("nema\n");
    return 0;
50 }
```

#### Rešenje 3.35

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

5  /* Struktura koja predstavlja jednog studenta */
   typedef struct student {
7     char nalog[8];
     char ime[21];
9     char prezime[21];
     int poeni;
11 } Student;

13
   /* Funkcija poredi studente prema broju poena, rastuce */
15 int uporedi_poeni(const void *a, const void *b)
   {
17     Student s = *(Student *) a;
     Student t = *(Student *) b;
19     return s.poeni - t.poeni;
21 }

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

```

45     else
46         return indeks1 - indeks2;
47 }
48
49 int uporedi_bsearch(const void *a, const void *b)
50 {
51     /* Nalog studenta koji se traži */
52     char *nalog = (char *) a;
53     /* Ključ pretrage */
54     Student s = *(Student *) b;
55
56     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
57     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
58     char smer1 = nalog[1];
59     char smer2 = s.nalog[1];
60     int indeks1 =
61         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] -
62         '0';
63     int indeks2 =
64         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
65         s.nalog[6] - '0';
66     if (godina1 != godina2)
67         return godina1 - godina2;
68     else if (smer1 != smer2)
69         return smer1 - smer2;
70     else
71         return indeks1 - indeks2;
72 }
73
74 int main(int argc, char **argv)
75 {
76     Student *nadjen = NULL;
77     char nalog_trazeni[8];
78     Student niz_studenata[100];
79     int i = 0, br_studenata = 0;
80     FILE *in = NULL, *out = NULL;
81
82     /* Ako je broj argumenata komandne linije različit i od 2 i od
83        3, korisnik nije ispravno pozvao program i prijavljujemo
84        gresku: */
85     if (argc != 2 && argc != 3) {
86         fprintf(stderr,
87             "Greska! Program se poziva sa: ./a.out -opcija (nalog)!\n
88             ");
89         exit(EXIT_FAILURE);
90     }
91
92     /* Otvaranje datoteke za citanje */
93     in = fopen("studenti.txt", "r");
94     if (in == NULL) {
95         fprintf(stderr,
96             "Greska prilikom otvaranja datoteke studenti.txt!\n");

```

```

    exit(EXIT_FAILURE);
97 }

99 /* Otvaranje datoteke za pisanje */
out = fopen("izlaz.txt", "w");
101 if (out == NULL) {
    fprintf(stderr,
103         "Greska prilikom otvaranja datoteke izlaz.txt!\n");
    exit(EXIT_FAILURE);
105 }

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

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

127 /* Sortirani studenti se ispisuju u izlaznu datoteku */
for (i = 0; i < br_studenata; i++)
    fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
129        niz_studenata[i].ime, niz_studenata[i].prezime,
        niz_studenata[i].poeni);
131

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

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

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

```

```

    }
149
    /* Zatvaranje datoteka */
151    fclose(in);
    fclose(out);
153
    return 0;
155 }

```

### Rešenje 3.37

```

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

4  /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
    standardnog ulaza */
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 ispis_i_matricu(int **a, int n, int m)
{
56     int i, j;

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

66
68 /* Funkcija koja alokira memoriju za matricu dimenzija nxm */
69 int **alociraj_memoriju(int n, int m)
{
70     int i, j;
71     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
84         /* Proveravamo da li je doslo do greske prilikom alokacije */
85         if (a[i] == NULL) {
86             /* Ako jeste, ispisujemo poruku */
87             fprintf(stderr, "Problem sa alokacijom memorije!\n");
88
89             /* I oslobadjamo memoriju zauzetu do ovog koraka */
90             for (j = 0; j < i; j++) {
91                 free(a[j]);
92             }
93         }
94     }
95 }
```

```

    free(a);
94     exit(EXIT_FAILURE);
    }
96 }

98     return a;
    }
100
101     /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije
102        nxm */
103     void oslobodi_memoriju(int **a, int n, int m)
104     {
105         int i;
106
107         for (i = 0; i < n; i++) {
108             free(a[i]);
109         }
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         /* Ucitavamo elemente matrice */
128         učitaj_matricu(a, n, m);
129
130         /* Pozivamo funkciju koja sortira vrste matrice prema zbiru */
131         sortiraj_vrste(a, n, m);
132
133         /* Ispisujemo rezultujuću matricu */
134         printf("Sortirana matrica je:\n")
135         ispisi_matricu(a, n, m);
136
137         /* Oslobadjamo memoriju */
138         oslobodi_memoriju(a, n, m);
139
140         /* I prekidamo sa izvršavanjem programa */
141         return 0;
142     }
}
```



## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

**Zadatak 4.1** Napisati program koji koristi 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.

- (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 novo kreiranog čvora.
- (c) Napisati funkciju koja dodaje novi elemenat na početak liste.
- (d) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama `[, ]` i međusobno razdvojene zapetama.
- (e) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- (f) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- (g) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.



## 4 Dinamičke strukture podataka

---

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. **Ana: Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i iterativno rešenje**

### Upotreba programa 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3 3 17 3 1 9 3
  Unosite element koji se trazi u listi: 17
  Unosite element koji se brise iz liste: 3
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]
  Lista: [3, 9, 1, 3, 17, 3, 3, 5, 14, 3, 2]

  Trazeni broj 17 je u listi!
  Lista nakon brisanja: [9, 1, 17, 5, 14, 2]
```

### Upotreba programa 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. 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: [35, 14, 23]

  Element NIJE u listi!
  Lista nakon brisanja: [35, 14, 23]
```

### Upotreba programa 3

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
  Unosite element koji se trazi u listi: 1
  Unosite element koji se brise iz liste: 12
Izlaz:
  Lista: []

  Element NIJE u listi!
  Lista nakon brisanja: []
```

[Rešenje 4.1]

**Zadatak 4.2** Napisati program koji koristi 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.

- Definisati strukturu `Cvor` koja predstavlja čvor liste.
- Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vraća adresu novo kreiranog čvora.
- Napisati funkciju koja dodaje novi element na kraj liste.
- Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama `[ , ]` i međusobno razdvojene zarezima.
- Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. **Ana: Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i iterativno rešenje**

#### *Upotreba programa 1*

```
Poziv: ./a.out
Ulaz:
Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3 3 17 3 1 9 3
Izlaz:
Lista: []
Lista: [2]
Lista: [2, 15]
Lista: [2, 15, 4]
Lista: [2, 15, 4, 8]
Lista: [2, 15, 4, 8, 3]
Lista: [2, 15, 4, 8, 3, 24]
Lista: [2, 15, 4, 8, 3, 24, 11]
Lista: [2, 15, 4, 8, 3, 24, 11, 17]
Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4]
Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4, 3]
Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4, 3, 7]
```

#### *Upotreba programa 2*

```
Poziv: ./a.out
Ulaz:
Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
Izlaz:
Lista: []
```

[Rešenje 4.2]

**Zadatak 4.3** Napisati program koji koristi 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.

- (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 novo kreiranog čvora.
- (c) Napisati funkciju koja dodaje novi element u listu tako da lista ostane rastuće sortirana.
- (d) Napisati funkciju koja oslobađa memoriju koju je zauzela lista.
- (e) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama  
,  
i međusobno razdvojene zapetama.
- (f) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- (g) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- (h) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. **Ana: Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i iterativno rešenje**

*Upotreba programa 1*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3 3 17 3 1 9 3
  Unosite element koji se trazi u listi: 5
  Unosite element koji se brise 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]
  Lista: [1, 2, 3, 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 EOF tj. 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]

```

*Upotreba programa 3*

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
  Unosite element koji se trazi u listi: 1
  Unosite element koji se brise iz liste: 12
Izlaz:
  Lista: []

  Element NIJE u listi!
  Lista nakon brisanja: []

```

[Rešenje 4.3]

**Zadatak 4.4** Napisati program koji koristi dvostruko povezanu listu za čuvanje celih brojeva koji se unose sa standardnog ulaza. Unošenje novih brojeva

u listu se prekida učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. **I ovde isto možda razdvojiti sortiranost od obične liste.**

- (a) Napisati funkciju koja dodaje novi elemenat na početak liste.
- (b) Napisati funkciju koja dodaje novi elemenat na kraj liste.
- (c) Napisati funkciju koja dodaje novi elemenat u listu tako da lista ostane rastuće sortirana.
- (d) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- (e) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- (f) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom implementirati iterativno.

**Zadatak 4.5** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade  $\{$ ,  $[$  i  $($ . Napisati program koji učitava sadržaj datoteke 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!
```

**Zadatak 4.6** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije . **Milena:** A sta ako se ne navede argument komandne linije? Uputstvo: za rešavanje problema koristiti stek implementiran preko listi 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 datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
</html>
Izlaz: Neispravno uparene etikete.
```

### Test 3

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
  </body>
Izlaz: Neispravno uparene etikete.
```

### Test 4

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva sa: ./a.out datoteka.html!
```

### Test 5

```
Poziv: ./a.out datoteka.html
Datoteka.html ne postoji.
Izlaz: Greska prilikom otvaranja datoteke datoteka.html.
```

### Test 6

```
Poziv: ./a.out datoteka.html
Datoteka.html je prazna
Izlaz: Ispravno uparene etikete.
```

**Zadatak 4.7** Milena: Problem sa ovim zadatkom je sto je program najpre na usluzi korisnicima, a zatim na usluzi sluzbeniku i to nekako zbunjuje u formulaiciji. Formulacija mi nije bila jasna bez citanja resenja, pokusala sam da je preciziran, u nastavku je izmenjena formulacija.

Medjutim, ja i dalje nisam bas zadovoljna i zato predlazem da se formulacija izmeni tako da je program stalno na usluzi sluzbeniku. Program ucitava podatke o prijavljenim korisnicima iz datoteke. Sluzbenik odlucuje da li ce da obradjuje redom korisnike, ili ce u nekim situacijama da odlozi rad sa korisnikom i stavi ga na kraj reda. Program ga uvek pita da na osnovu jmbg-a i zahteva odluci da li ce ga staviti na kraj reda, ako hoce, on ide na kraj reda, ako nece, onda sluzbenik daje odgovor na zahtev i jmbg, zahtev i odgovor se upisuju u izlaznu datoteku.

Napisati program kojim se simulira rad jednog šaltera na kojem se prvo zakažu termini, a potom službenik uslužuje korisnike redom, kako su se prijavljivali.

Korisnik se prijavljuje unošenjem svog jmbg broja (niska koja sadrži 13 karaktera) i zahteva (niska koja sadrži najviše 999 karaktera). Prijavljivanje korisnika se prekida unošenjem karaktera za kraj ulaza (EOF).

Službenik redom proziva korisnike čitanjem njihovog jmbg broja, a zatim odlu-

čuje da li korisnika vraća na kraj reda ili ga odmah uslužuje. Službeniku se postavlja pitanje **Da li korisnika vracate na kraj reda?** i ukoliko on da odgovor **Da**, korisnik se vraća na kraj reda. Ukoliko odgovor nije **Da**, tada službenik čita korisnikov zahtev. Posle svakog 10 usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu.

Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.

**Zadatak 4.8** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
{
    unsigned broj_pojavljivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;
```

#### Test 1

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    A sutra ce biti jos lepsi.
    <a link="http://www.google.com"> Link 1</a>
    <a link="http://www.math.rs"> Link 2</a>
  </body>
</html>

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

#### Test 2

```
Poziv: ./a.out
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:
```

**Zadatak 4.9** Milena: malo me muci u ovom zadatku sto nema neki smisao. Naime, ako se samo vrsi učitavanje iz datoteka i ispisivanje, onda su ove liste zapravo visak jer isti rezultat moze da se dobije i bez koriscenja listi. Zato mi fali da program uradi nesto sto ne bi mogao da uradi bez koriscenja listi, npr da na osnovu unetog broja ispisuje svaki n-ti broj rezultujuce liste pa to u nekoj petlji da korisnik moze da ispisuje za razlicite unete n ili tako nesto...

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. 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 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

```

**Zadatak 4.10** Napisati funkciju koja formira listu studenata tako što se podaci o studentima učitavaju 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. Napisati rekurzivnu funkciju koja određuje da li neki student pripada listi ili ne. Ispisati zatim odgovarajuću poruku i rekurzivno osloboditi memoriju koju je data lista zauzimala. Student se traži na osnovu broja indeksa, koji se zadaje sa standardnog ulaza.

*Test 1*

```

Poziv: ./a.out studenti.txt
Datoteka:      Ulaz:      Izlaz:
123/2014 Marko Lukic      3/2014      da: Ana Sokic
3/2014  Ana Sokic        235/2008      ne
43/2013 Jelena Ilic      41/2009      da: Marija Zaric
41/2009 Marija Zaric
13/2010 Milovan Lazic

```

*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

```
Poziv: ./a.out studenti.txt
studenti.txt prazna
Izlaz: ??? videti sta ce tacno biti
```

**Zadatak 4.11** Neka su date 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 koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. *Milena: Sta ako je neka lista duza? To precizirati. I ovde me muci sto nedostaje neki smisao zadatku, nesto sto ne bi moglo da se uradi da nismo koristili liste.*

### 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 4.12** Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

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

Napisati program koji u datoteku `Rezultat.txt` upisuje nađeni strogo rastući podniz. Milena: I ovde me muci sto bi zadatak mogao da se resi i bez koriscenja listi...

Milena: Prirodni oblik testa ovde bi bio horizontalan, a ne ovako vertikaln.

*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 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 4.13** 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. 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: u implementaciji koristiti kružnu listu.

### Test 1

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

**Zadatak 4.14** 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, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavao 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: u implementaciji koristiti dvostruko povezanu kružnu listu.

*Test 1*

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

*Test 1*

```
Ulaz: 2 7
Izlaz: n mora biti uvek vece
       od k, a 2 < 7!
```

## 4.2 Stabla

**Zadatak 4.15** 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 alocira 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`.
- Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili `NULL` ukoliko takav čvor ne postoji.
- Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.

- (f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor** 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!
  Briše se broj: 7
  Rezultujuće 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
  Rezultujuce stablo: -3 -2 6 8 13 24

```

[Rešenje 4.15]

**Zadatak 4.16** 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 prema 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 4.16]

**Zadatak 4.17** 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
Izlaz:
      Broj je: 011/3240-987
Ulaz:
      Unesite ime i prezime: Marko Markovic
Izlaz:
      Broj nije u imeniku!
Ulaz:
      Unesite ime i prezime: KRAJ
```

[Rešenje 4.17]

**Zadatak 4.18** 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 4.18]

\* **Zadatak 4.19** 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 zadanog 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 4.19]

**Zadatak 4.20** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. *Napomena: 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. Programe 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 4.20]

\* **Zadatak 4.21** 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

```

[Rešenje 4.21]

**Zadatak 4.22** 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 4.22]

**Zadatak 4.23** Dato je binarno pretraživačko stablo celih brojeva.

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

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara  $i$  i  $x$  pročitati kao argumente komandne linije.

*Test 2*

```

Poziv: ./a.out 2 15
Ulaz:
10 5 15 3 2 4 30 12 14 13
Izlaz:
broj cvorova: 10
broj listova: 4
pozitivni listovi: 2 4 13 30
zbir cvorova: 108
najveci element: 30
dubina stabla: 5
broj cvorova na 2. nivou: 3
elementi na 2. nivou: 3 12 30
maksimalni na 2. nivou: 30
zbir na 2. nivou: 45
zbir elemenata manjih ili jednakih od 15: 7

```

[Rešenje 4.23]

**Zadatak 4.24** Dato je binarno pretraživačko 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.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gorenavedene funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza.

*Test 1*

```

Poziv: ./a.out
Ulaz:
10 5 15 3 2 4 30 12 14 13
Izlaz:
Cvor sa maksimalnim desnim proizvodom: 10
Cvor sa najmanjom levom sumom: 2
Putanja do najdubljeg cvora: 10 15 12 14 13
Putanja do najmanjeg cvora: 10 5 3 2

```

[Rešenje 4.24]

**Zadatak 4.25** 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
```

[Rešenje 4.25]

**\* Zadatak 4.26** 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.
```

[Rešenje 4.26]

### 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 4.27** AVL-stablo je binarno stablo pretrage kod koga 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 4.27]

**Zadatak 4.28** Binarno stablo se naziva HEAP ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablina. 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.

*Test 1*

```

Poziv: ./a.out
Ulaz:
    100 19 36 17 3 25 1 2 7
Izlaz:
    Stablo je heap.

```

[Rešenje 4.28]

## 4.3 Rešenja

### Rešenje 4.1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* NAPOMENA:
5     Jednostruko povezana lista je struktura podataka
6     koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
7     podatak (odredjenog tipa) i pokazivac na sledeci cvor u
8     sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
9     liste (bez glave) je takodje lista, i naziva se rep liste.
10    Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
11    baratiranja listom mi cuvamo samo pokazivac na glavu liste.
12    Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
13    sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
14    pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
15    preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
16    element u listi nema sledeci element: u tom slucaju se

```



```
18      njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
      lista se predstavlja NULL pokazivacem.

20      Prednost koriscenja povezanih lista u odnosu na dinamicki
      niz je u tome sto se elementi mogu efikasno umetati i brisati
22      sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
      premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
24      ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
      elementi moraju obradjivati redom (iteracijom kroz listu).

26

28      Prilikom promene liste (dodavanje novog elementa, brisanje
      elementa,
      premestanje elemenata, itd.) postoji mogucnost da glava liste bude
      promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
30      U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
      promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
32      pozivajucoj funkciji vrati azurirana informacija o adresi glave
      liste.

34

36      Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
      pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
      cuva
      adresu glave i koju, eventualno, treba azurirati.
38      Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
      promeni promenljivu pozivajuce funkcije direktno. Npr:
40          funkcija_zapromenu(&pok, ...);
      */

42
43      /* Struktura koja predstavlja cvor liste */
44      typedef struct cvor {
45          /* Podatak koji cvor sadrzi */
46          int vrednost;
47          /* Pokazivac na sledeci cvor liste */
48          struct cvor *sledeci;
49      } Cvor;

50

51
52      /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
53      novog cvora inicijalizuje na broj, dok pokazivac na
54      sledeci cvor u novom cvoru postavlja na NULL.
55      Funkcija vraca pokazivac na novokreirani cvor ili NULL
56      ako alokacija nije uspesno izvrшена. */
57      Cvor *napravi_cvor(int broj) {
58          Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
59          if( novi == NULL )
60              return NULL;

61
62          /* Inicijalizacija polja u novom cvoru */
63          novi->vrednost = broj;
64          novi->sledeci = NULL;
```

```

66     return novi;
67 }
68
69
70
71
72 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
73    ciji se pocetni cvor nalazi na adresi adresa_glave. */
74 void oslobodi_listu(Cvor ** adresa_glave) {
75     Cvor *pomocni = NULL;
76
77     /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
78        */
79     while (*adresa_glave != NULL) {
80         /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
81            a tek onda osloboditi element koji predstavlja glavu liste */
82         pomocni = (*adresa_glave)->sledeci;
83         free(*adresa_glave);
84         /* Sledeci element je nova glava liste */
85         *adresa_glave = pomocni;
86     }
87 }
88
89
90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
91    ukoliko
92    alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
93    memorija
94    za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
95 void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
96     /* Ukoliko je novi NULL */
97     if ( novi == NULL ) {
98         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
99     }
100    /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
101       */
102    oslobodi_listu(adresa_glave);
103    exit(EXIT_FAILURE);
104 }
105
106 /* Funkcija dodaje novi cvor na pocetak liste.
107    Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na
108    pocetak */
109 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
110     /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
111        alokaciji */
112     Cvor *novi = napravi_cvor(broj);
113     prover_i_alokaciju(adresa_glave, novi);

```

```
112      /* Uvezujemo novi cvor na pocetak */
114      novi->sledeci = *adresa_glave;
116      /* Nov cvor je sada nova glava liste */
      *adresa_glave = novi;
118  }

120
122  /* Funkcija pronalazi i vraća pokazivac na poslednji element liste,
      ili NULL ukoliko je lista prazna */
124  Cvor* pronadji_poslednji (Cvor* glava) {
      /* ako je lista prazna, nema ni poslednjeg cvor
      i u tom slucaju vracamo NULL.*/
126      if( glava == NULL)
          return NULL;
128
      /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo
      pokazivac
130      glava na taj sledeci element. Kada izađemo iz petlje,
      glava ce pokazivati na element liste koji nema sledeceg,
132      tj. poslednji element liste je. Zato vracamo vrednost
      pokazivaca glava.

134      glava je argument funkcije i njegove promene nece se odraziti
      na
      vrednost pokazivaca glava u pozivajucoj funkciji. */
136      while (glava->sledeci != NULL)
          glava = glava->sledeci;
138
      return glava;
140  }

142
144  /* Funkcija trazi u listi element cija je vrednost jednaka datom
      broju.
      Vraca pokazivac na cvor liste u kome je sadržan traženi broj
      ili NULL u slucaju da takav element ne postoji u listi. */
146  Cvor* pretrazi_listu(Cvor * glava, int broj) {
      for ( ; glava != NULL; glava = glava->sledeci)
148          /* Pronasli smo*/
          if (glava->vrednost == broj)
150              return glava;

152      /* Nema traženog broja u listi i vracamo NULL*/
      return NULL;
154  }

156
158
      /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
```

```

160 Funkcija azurira pokazivac na glavu liste (koji moze biti
    promenjen u slucaju da se obrise stara glava) */
162 void obrisi_element(Cvor ** adresa_glave, int broj) {
    Cvor *tekuci = NULL;
164     Cvor *pomocni = NULL;

166     /* Brisemo sa pocetka liste sve eventualne cvorove
        koji su jednaki datom broju, i azuriramo pokazivac na glavu */
168     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
        /* Sacuvamo adresu repa liste pre oslobadjanja glave */
170         pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
172         *adresa_glave = pomocni;
    }

174     /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
176     if (*adresa_glave == NULL)
        return;

178     /* Od ovog trenutka se u svakom koraku nalazimo
        na tekucem cvoru koji je razlicit od trazenog
        broja (kao i svi levo od njega). Poredimo
180     vrednost sledeceg cvora (ako postoji) sa trazenim
        brojem i brisemo ga ako je jednak, a prelazimo na
182     sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
        dok ne dodjemo do poslednjeg cvora. */
184     tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL)
186     {
        if (tekuci->sledeci->vrednost == broj) {
188             /* tekuci->sledeci treba obrisati,
                zbog toga sacuvamo njegovu adresu u pomocni */
190             pomocni = tekuci->sledeci;
            /* Tekucem preusmerimo pokazivac sledeci
                tako sto preskakemo njegovog trenutnog sledeceg.
192             Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
            tekuci->sledeci = tekuci->sledeci->sledeci;
194             /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
                */
            free(pomocni);
196             } else {
                /* Ne treba brisati sledeceg, prelazimo na sledeci */
200                 tekuci = tekuci->sledeci;
            }
198         }
202     }
    return;
204 }

206 /* 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 potrebe da azuriza
    pokazivac
208     na glavu liste iz pozivajuce funkcije. */

```

```
void ispisi_listu(Cvor * glava)
210 {
    putchar('[');
212     for ( ; glava != NULL; glava = glava->sledeci)
    {
214         printf("%d", glava->vrednost);
        if( glava->sledeci != NULL )
216             printf(", ");
    }

218     printf("]\n");
220 }

222

224 /* Glavni program u kome testiramo sve funkcije za rad sa listama */
226 int main() {
    Cvor *glava = NULL; /* na pocetku imamo praznu listu */
228     Cvor *trazeni = NULL;
    int broj;

230     /* Testiramo dodavanje na pocetak*/
232     printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+D)
        \n");
    printf("\n\tLista: ");
234     ispisi_listu(glava);

236     while(scanf("%d",&broj)>0)
    {
238         dodaj_na_pocetak_liste(&glava, broj);
        printf("\n\tLista: ");
240         ispisi_listu(glava);
    }

242     printf("\nUnesite element koji se trazi u listi: ");
244     scanf("%d", &broj);

246     trazeni=pretrazi_listu(glava, broj);
    if(trazeni==NULL)
248         printf("Element NIJE u listi!\n");
    else
250         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

252     /* brisemo elemente iz liste cije polje vrednost je jednako
        broju procitanom sa ulaza */
254     printf("\nUnesite element koji se brise iz liste: ");
256     scanf("%d", &broj);

258     obrisi_element(&glava, broj);
```

```

260     printf("Lista nakon brisanja: ");
      ispisi_listu(glava);
262
      oslobodi_listu(&glava);
264
      return 0;
266 }

```

## Rešenje 4.2

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   /* NAPOMENA:
5   Jednostruko povezana lista je struktura podataka
   koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
7   podatak (odredjenog tipa) i pokazivac na sledeci cvor u
   sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
9   liste (bez glave) je takodje lista, i naziva se rep liste.
   Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
11  baratanja listom mi cuvamo samo pokazivac na glavu liste.
   Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
13  sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
   pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
15  preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
   element u listi nema sledeci element: u tom slucaju se
17  njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
   lista se predstavlja NULL pokazivcem.
19
   Prednost koriscenja povezanih lista u odnosu na dinamicki
21  niz je u tome sto se elementi mogu efikasno umetati i brisati
   sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
23  premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
   ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
25  elementi moraju obradivati redom (iteracijom kroz listu).
27
   Prilikom promene liste (dodavanje novog elementa, brisanje
   elementa,
29  premestanje elemenata, itd.) postoji mogucnost da glava liste bude
   promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
   U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
31  promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
   pozivajucoj funkciji vrati azurirana informacija o adresi glave
33  liste.
35
   Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
   pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
   cuva
37  adresu glave i koju, eventualno, treba azurirati.
   Sada pozvana funkcija moze interno da preko dobijenog pokazivaca

```

```
39     promeni promenljivu pozivajuće funkcije direktno. Npr:
      funkcija_za_promenu(&pok, ...);
41 */
43 /* Struktura koja predstavlja cvor liste */
      typedef struct cvor {
45         /* Podatak koji cvor sadrzi */
         int vrednost;
47         /* Pokazivac na sledeci cvor liste */
         struct cvor *sledeci;
49     } Cvor;
51
53 /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
      novog cvora inicijalizuje na broj, dok pokazivac na
55 sledeci cvor u novom cvoru postavlja na NULL.
      Funkcija vraća pokazivac na novokreirani cvor ili NULL
      ako alokacija nije uspesno izvršena. */
      Cvor * napravi_cvor(int broj) {
57         Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
         if( novi == NULL )
61             return NULL;
63
         /* Inicijalizacija polja u novom cvoru */
         novi->vrednost = broj;
65         novi->sledeci = NULL;
         return novi;
67     }
69
71 /* Funkcija oslobadja dinamičku memoriju zauzetu za elemente liste
      čiji se početni cvor nalazi na adresi adresa_glave. */
73 void oslobodi_listu(Cvor ** adresa_glave) {
75     Cvor *pomocni = NULL;
77
         /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
            */
         while (*adresa_glave != NULL) {
79             /* Potrebno je najpre zapamtiti adresu sledećeg elementa,
                a tek onda osloboditi element koji predstavlja glavu liste */
             pomocni = (*adresa_glave)->sledeci;
             free(*adresa_glave);
83             /* Sledeci element je nova glava liste */
             *adresa_glave = pomocni;
85         }
87     }
89 }
```

```

/* Funkcija proverava uspesnost alokacije memorije za cvor novi i
   ukoliko
91   alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
   memorija
   za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
93 void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
   /* Ukoliko je novi NULL */
95   if ( novi == NULL ) {
       fprintf(stderr, "Neuspela alokacija za nov cvor\n");
97   /* Oslobadjamo svu dinamički alociranu memoriju i prekidamo program
       */

       oslobodi_listu(adresa_glave);
       exit(EXIT_FAILURE);
101   }
103 }

/* Funkcija pronalazi i vraća pokazivac na poslednji element liste,
   ili NULL ukoliko je lista prazna */
105 Cvor* pronadji_poslednji (Cvor* glava) {
107   /* ako je lista prazna, nema ni poslednjeg cvor
   i u tom slucaju vracamo NULL.*/
109   if( glava == NULL)
111       return NULL;

113   /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo
   pokazivac
   glava na taj sledeci element. Kada izađemo iz petlje,
115   glava ce pokazivati na element liste
   koji nema sledeceg, tj, poslednji element liste je. Zato vracamo
117   vrednost pokazivaca glava.

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

123   return glava;

125 }

127

129 /* Funkcija dodaje novi cvor na kraj liste. */
131 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj) {
   Cvor *novi = napravi_cvor(broj);
133   /* Proveravamo da li je doslo do greske prilikom alokacije
   memorije */
   prover_i_alokaciju(adresa_glave, novi);

135   /* U slucaju prazne liste */

```



```
137     if (*adresa_glave == NULL) {
138         /* Glava nove liste je upravo novi cvor i ujedno i cela
139         lista.
140         * Azuriramo vrednost na koju pokazuje adresa_glave i tako
141         azuriramo
142         * i pokazivacku promenljivu u pozivajucoj funkciji. */
143         *adresa_glave = novi;
144     /* Vracamo se iz funkcije */
145     return;
146 }
147
148 /* Ako lista nije prazna, pronalazimo poslednji element liste */
149 Cvor* poslednji = pronadji_poslednji(*adresa_glave);
150
151 /* Dodajemo novi cvor na kraj preusmeravanjem pokazivaca */
152 poslednji->sledeci = novi;
153 }
154
155 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
156 Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
157 ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
158 pokazivac
159 na glavu liste iz pozivajuce funkcije. */
160 void ispisi_listu(Cvor * glava)
161 {
162     putchar('[');
163     for ( ; glava != NULL; glava = glava->sledeci)
164     {
165         printf("%d", glava->vrednost);
166         if( glava->sledeci != NULL )
167             printf(", ");
168     }
169     printf("]\n");
170 }
171
172
173
174
175 /* Glavni program u kome testiramo sve funkcije za rad sa listama */
176 int main() {
177     Cvor *glava = NULL; /* na pocetku imamo praznu listu */
178     int broj;
179
180     /* Testiramo dodavanje na kraj liste */
181     printf("\nUnesite elemente liste! (za kraj unesite EOF tj. CTRL+D)
182     \n");
183     printf("\n\tLista: ");
184     ispisi_listu(glava);
```

```

185     while(scanf("%d",&broj)>0) {
186         dodaj_na_kraj_liste(&glava, broj);
187         printf("\n\tLista: ");
188         ispisi_listu(glava);
189     }
191     oslobodi_listu(&glava);
193     return 0;
194 }

```

### Rešenje 4.3

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* NAPOMENA:
5   Jednostruko povezana lista je struktura podataka
6   koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
7   podatak (odredjenog tipa) i pokazivac na sledeci cvor u
8   sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
9   liste (bez glave) je takodje lista, i naziva se rep liste.
10  Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
11  baratiranja listom mi cuvamo samo pokazivac na glavu liste.
12  Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
13  sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
14  pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
15  preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
16  element u listi nema sledeci element: u tom slucaju se
17  njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
18  lista se predstavlja NULL pokazivacem.
19
20  Prednost koriscenja povezanih lista u odnosu na dinamicki
21  niz je u tome sto se elementi mogu efikasno umetati i brisati
22  sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
23  premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
24  ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
25  elementi moraju obradivati redom (iteracijom kroz listu).
26
27
28  Prilikom promene liste (dodavanje novog elementa, brisanje
29  elementa,
30  premestanje elemenata, itd.) postoji mogucnost da glava liste bude
31  promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
32  U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
33  promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
34  pozivajucoj funkciji vrati azurirana informacija o adresi glave
35  liste.
36
37  Pozvana funkcija koja vrsi promenu na listi prihvata kao argument

```

```
36     pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
      cuva
      adresu glave i koju, eventualno, treba azurirati.
38     Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
      promeni promenljivu pozivajuce funkcije direktno. Npr:
40     funkcija_za_promenu(&pok, ...);
      */
42
      /* Struktura koja predstavlja cvor liste */
44     typedef struct cvor {
      /* Podatak koji cvor sadrzi */
46         int vrednost;
      /* Pokazivac na sledeci cvor liste */
48         struct cvor *sledeci;
    } Cvor;
50
52
      /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
      novog cvora inicijalizuje na broj, dok pokazivac na
54     sledeci cvor u novom cvoru postavlja na NULL.
      Funkcija vraca pokazivac na novokreirani cvor ili NULL
      ako alokacija nije uspesno izvorsena. */
56     Cvor * napravi_cvor(int broj) {
      Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60         if( novi == NULL )
            return NULL;
62
      /* Inicijalizacija polja u novom cvoru */
64         novi->vrednost = broj;
        novi->sledeci = NULL;
66         return novi;
    }
68
70
      /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
      ciji se pocetni cvor nalazi na adresi adresa_glave. */
72     void oslobodi_listu(Cvor ** adresa_glave) {
      Cvor *pomocni = NULL;
74
      /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
      */
76
      while (*adresa_glave != NULL) {
78         /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
          a tek onda osloboditi element koji predstavlja glavu liste */
80         pomocni = (*adresa_glave)->sledeci;
          free(*adresa_glave);
82         /* Sledeci element je nova glava liste */
          *adresa_glave = pomocni;
84     }
}
```

```

86 }
88
90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
    alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
92 za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94     /* Ukoliko je novi NULL */
    if ( novi == NULL ) {
96         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
    /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
        */
98
        oslobodi_listu(adresa_glave);
100        exit(EXIT_FAILURE);
    }
102 }
104
106 /* Funkcija dodaje novi cvor na pocetak liste.
    Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na
    pocetak */
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
    /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
    alokaciji */
110    Cvor *novi = napravi_cvor(broj);
    prover_i_alokaciju(adresa_glave, novi);
112
    /* Uvezujemo novi cvor na pocetak */
114    novi->sledeci = *adresa_glave;
    /* Nov cvor je sada nova glava liste */
116    *adresa_glave = novi;
118 }
120
122 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
    nov element sa vrednoscu broj.*/
122 Cvor * pronadji_mesto_umetanja(Cvor* glava, int broj ) {
    /*Ako je lista prazna onda nema takvog mesta i vracamo NULL */
124    if(glava == NULL)
        return NULL;
126
    /* Krecemo se kroz listu sve dok se ne dodje do elementa
    ciji je sledeci element veci ili jednak od novog elementa,
    ili dok se ne dodje do poslednjeg elementa.
128
130    Zbog lenjog izracunavanja izraza u C-u prvi deo konjukcije
132    mora biti provera da li smo dosli do poslednjeg elementa liste

```

```

134     pre nego sto proverimo vrednost njegovog sledeceg elementa,
        jer u slucaju poslednjeg, sledeci ne postoji, pa ni vrednost.*/
        while (glava->sledeci != NULL && glava->sledeci->vrednost < broj
        )
136             glava = glava->sledeci;

138     /* Iz petlje smo mogli izaci jer smo dosli do poslednjeg elementa
        ili smo se zaustavili ranije na elementu ciji sledeci ima
140     vrednost vecu od broj */
        return glava;
142 }

144
146 void dodaj_iza(Cvor* tekuci, Cvor* novi) {
        /* Novi element dodajemo iza tekuceg elementa */
        novi->sledeci = tekuci->sledeci;
148     tekuci->sledeci = novi;
    }

150
152 /* Funkcija dodaje novi element u sortiranu listu
    tako da nova lista ostane sortirana.*/
    void dodaj_sortirano(Cvor ** adresa_glave, int broj) {
154         /* U slucaju prazne liste glava nove liste je upravo novi cvor */
        if ( *adresa_glave == NULL ) {
156             Cvor *novi = napravi_cvor(broj);
            /* Proveravamo da li je doslo do greske prilikom alokacije
            memorije */
158             prover_i_alokaciju(adresa_glave, novi);
            *adresa_glave = novi;
160             return;
        }

162
        /* Lista nije prazna*/
164         /* Ako je broj manji ili jednak vrednosti u glavi liste,
            onda ga dodajemo na pocetak liste */
166         if ( (*adresa_glave)->vrednost >= broj ) {
            dodaj_na_pocetak_liste(adresa_glave, broj);
168             return;
        }

170
        /* U slucaju da je glava liste element manji od novog elementa,
172         tada pronalazimo element liste iza koga treba da se umetne nov
            broj */
        Cvor* pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);

174
        Cvor *novi = napravi_cvor(broj);
176         /* Proveravamo da li je doslo do greske prilikom alokacije
            memorije */
        prover_i_alokaciju(adresa_glave, novi);

178
        /* Uvezujemo novi cvor iza pomocnog */
180         dodaj_iza(pomocni, novi);
    }

```

```

182 }
184
186 /* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju.
    Funkcija se u pretrazi oslanja na cinjenicu da je lista
    koja se pretrazuje rastuce sortirana.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj
    ili NULL u slučaju da takav element ne postoji u listi. */
190 Cvor* pretrazi_listu(Cvor * glava, int broj) {
    /* U konjukciji koja cini uslov ostanka u petlji, bitan je
    redosled! */
192     for ( ; glava != NULL && glava->vrednost <= broj ; glava = glava
        ->sledeci)
        /* Pronasli smo*/
194         if (glava->vrednost == broj)
            return glava;
196
        /* Nema traženog broja u listi i vracamo NULL*/
198     return NULL;
199 }
200
202
204 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na cinjenicu da je prosledjena lista sortirana
    rastuce.
    Funkcija azurira pokazivac na glavu liste, koji može biti
    promenjen u slučaju da se obriše stara glava liste. */
206 void obrisi_element(Cvor ** adresa_glave, int broj) {
208     Cvor *tekuci = NULL ;
    Cvor *pomocni = NULL ;
210
    /* Brisemo sa pocetka liste sve eventualne cvorove
    koji su jednaki datom broju, i azuriramo pokazivac na glavu */
212     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj
        ) {
214         /* Sacuvamo adresu repa liste pre oslobadjanja glave */
        pomocni = (*adresa_glave)->sledeci;
216         free(*adresa_glave);
        *adresa_glave = pomocni;
218     }
220
    /* Ako je nakon toga lista ostala prazna ili glava liste sadrži
    vrednost
    koja je veca od broja, kako je lista sortirana rastuce nema
    potrebe
    broj traziti u repu liste i zato prekidamo funkciju */
222     if ( *adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
224         return;

```

```

226     /* Od ovog trenutka se u svakom koraku nalazimo u tekucem cvoru
        cija
        vrednost je manja od trazenog broja (kao i svi levo od njega).
228     Poredimo vrednost sledeceg cvora (ako postoji) sa trazenim
        brojem
        i brisemo ga ako je jednak, a prelazimo na sledeci cvor ako je
230     razlicit. Ovaj postupak ponavljamo dok ne dodjemo do
        poslednjeg cvora
        ili prvog cvora cija vrednost je veca od trazenog broja. */
232     tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <=
        broj)
234         if (tekuci->sledeci->vrednost == broj) {
        /* tekuci->sledeci treba obrisati,
        zbog toga sacuvamo njegovu adresu u pomocni */
236         pomocni = tekuci->sledeci;
        /* Tekucem preusmerimo pokazivac sledeci
        tako sto preskacemo njegovog trenutnog sledeceg.
        Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
240         tekuci->sledeci = tekuci->sledeci->sledeci;
        /* Sada mozemo
        * slobodno i da oslobodimo cvor sa vrednoscu broj */
242         free(pomocni);
        } else {
244         /* Ne treba brisati sledeceg, jer je manji od trazenog
        i prelazimo na sledeci */
246         tekuci = tekuci->sledeci;
        }
248     return;
250 }

252
254 /* 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 potrebe da azuriza
    pokazivac
256 na glavu liste iz pozivajuce funkcije. */
void ispisi_listu(Cvor * glava)
258 {
    putchar('[');
260     for ( ; glava != NULL; glava = glava->sledeci)
    {
        printf("%d", glava->vrednost);
262     if( glava->sledeci != NULL )
        printf(", ");
264     }

266     printf("]\n");
268 }

270
272

```

```
274 /* Glavni program u kome testiramo sve funkcije za rad sa listama */
int main() {
276     Cvor *glava = NULL; /* na pocetku imamo praznu listu */
    Cvor *trazeni = NULL;
278     int broj;

280     /* Testiramo dodavanje u listu tako da ona bude rastuce sortirana
    */
    printf("\nUnosite elemente liste! (za kraj unesite EOF tj. CTRL+D
    )\n");
282     printf("\n\tLista: ");
    ispisi_listu(glava);

284     while(scanf("%d",&broj)>0)
286     {
        dodaj_sortirano(&glava, broj);
288         printf("\n\tLista: ");
        ispisi_listu(glava);
290     }

292     printf("\nUnesite element koji se trazi u listi: ");
    scanf("%d", &broj);

294     trazeni = pretrazi_listu(glava, broj);
    if(trazeni == NULL)
296         printf("Element NIJE u listi!\n");
    else
298         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
300

302     /* Brisemo elemente iz liste cije polje vrednost je jednako broju
    procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
304     scanf("%d", &broj);

306     obrisi_element(&glava, broj);

308     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);
310

312     oslobodi_listu(&glava);

314     return 0;
}
```

#### Rešenje 4.4

#### Rešenje 4.5



Rešenje 4.6

Rešenje 4.7

Rešenje 4.8

Rešenje 4.9

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* NAPOMENA:
5  Jednostruko povezana lista je struktura podataka
6  koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
7  podatak (određenog tipa) i pokazivac na sledeći cvor u
8  sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
9  liste (bez glave) je takodje lista, i naziva se rep liste.
10 Lista koja ne sadrži cvorove naziva se prazna lista. Prilikom
11 baratanja listom mi cuvamo samo pokazivac na glavu liste.
12 Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
13 sledećeg elementa, pa mu samim tim mozemo pristupiti. Kada mu
14 pristupimo, u njemu je sadržana adresa sledećeg elementa, pa
15 preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
16 element u listi nema sledeći element: u tom slučaju se
17 njegov pokazivac na sledeći postavlja na NULL. Takodje, prazna
18 lista se predstavlja NULL pokazivcem.
19
20 Prednost koriscenja povezanih lista u odnosu na dinamički
21 niz je u tome što se elementi mogu efikasno umetati i brisati
22 sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
23 premestanjem elemenata. Nedostatak ovakvog pristupa je to što
24 ne mozemo nasumično pristupiti proizvoljnom elementu, već se
25 elementi moraju obradjivati redom (iteracijom kroz listu).
26
27 Prilikom promene liste (dodavanje novog elementa, brisanje
28 elementa,
29 premestanje elemenata, itd.) postoji mogućnost da glava liste bude
30 promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
31 U tom slučaju se pokazivac na glavu liste mora azurirati. Kada
32 promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
33 pozivajućoj funkciji vrati azurirana informacija o adresi glave
34 liste.
35
36 Pozvana funkcija koja vrši promenu na listi prihvata kao argument
37 pokazivac na pokazivačku promenljivu koja u pozivajućoj funkciji
38 čuva
39 adresu glave i koju, eventualno, treba azurirati.
40 Sada pozvana funkcija može interno da preko dobijenog pokazivaca
```

```
40     promeni promenljivu pozivajuće funkcije direktno. Npr:
        funkcija_za_promenu(&pok, ...);
42 */
44 /* Struktura koja predstavlja cvor liste */
typedef struct cvor {
46     /* Podatak koji cvor sadrzi */
    int vrednost;
    /* Pokazivac na sledeci cvor liste */
48     struct cvor *sledeci;
} Cvor;
50
52
54 /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
    novog cvora inicijalizuje na broj, dok pokazivac na
    sledeci cvor u novom cvoru postavlja na NULL.
56 Funkcija vraća pokazivac na novokreirani cvor ili NULL
    ako alokacija nije uspesno izvršena. */
58 Cvor * napravi_cvor(int broj) {
    Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60     if( novi == NULL )
        return NULL;
62
    /* Inicijalizacija polja u novom cvoru */
64     novi->vrednost = broj;
    novi->sledeci = NULL;
66     return novi;
}
68
70
72 /* Funkcija oslobadja dinamičku memoriju zauzetu za elemente liste
    ciji se pocetni cvor nalazi na adresi adresa_glave. */
74 void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;
76
    /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
    */
78     while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
80         a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
82         /* Sledeci element je nova glava liste */
84         *adresa_glave = pomocni;
    }
86 }
88
```

```

90  /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
    alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
92  za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
void proveri_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94  /* Ukoliko je novi NULL */
    if ( novi == NULL ) {
96      fprintf(stderr, "Neuspela alokacija za nov cvor\n");
    /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
       */
98
        oslobodi_listu(adresa_glave);
100       exit(EXIT_FAILURE);
    }
102 }

104

106 /* Funkcija dodaje novi cvor na pocetak liste.
    Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na
    pocetak */
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
    /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
       alokaciji */
110     Cvor *novi = napravi_cvor(broj);
    proveri_alokaciju(adresa_glave, novi);
112
    /* Uvezujemo novi cvor na pocetak */
114     novi->sledeci = *adresa_glave;
    /* Nov cvor je sada nova glava liste */
116     *adresa_glave = novi;
118 }

120

122 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL ukoliko je lista prazna */
Cvor* pronadji_poslednji (Cvor* glava) {
124     /* ako je lista prazna, nema ni poslednjeg cvor
       i u tom slucaju vracamo NULL.*/
126     if( glava == NULL)
        return NULL;
128
    /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo
       pokazivac
130     glava na taj sledeci element. Kada izadjemo iz petlje,
       glava ce pokazivati na element liste koji nema sledeceg,
132     tj. poslednji element liste je. Zato vracamo vrednost
       pokazivaca glava.

134     glava je argument funkcije i njegove promene nece se odraziti

```

```

na
    vrednost pokazivaca glava u pozivajucoj funkciji. */
136 while (glava->sledeci != NULL)
    glava = glava->sledeci;
138
    return glava;
140 }

142
/* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju.
    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj
    ili NULL u slucaju da takav element ne postoji u listi. */
144
Cvor* pretrazi_listu(Cvor * glava, int broj) {
    for ( ; glava != NULL; glava = glava->sledeci)
146         /* Pronasli smo */
        if (glava->vrednost == broj)
148             return glava;
150
    /* Nema trazenog broja u listi i vracamo NULL */
    return NULL;
152
}
154

156

158
/* 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) */
160
void obrisi_element(Cvor ** adresa_glave, int broj) {
    Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;
162

    /* Brisemo sa pocetka liste sve eventualne cvorove
        koji su jednaki datom broju, i azuriramo pokazivac na glavu */
164
    while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj
    ) {
        /* Sacuvamo adresu repa liste pre oslobadjanja glave */
166
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        *adresa_glave = pomocni;
168
    }

    /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
170
    if ( *adresa_glave == NULL)
        return;
172

    /* Od ovog trenutka se u svakom koraku nalazimo
        na tekucem cvoru koji je razlicit od trazenog
        broja (kao i svi levo od njega). Poredimo
        vrednost sledeceg cvora (ako postoji) sa trazanim
        brojem i brisemo ga ako je jednak, a prelazimo na
174
176
178
180
182

```

```

184         sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
           dok ne dodjemo do poslednjeg cvora. */
186     tekuci = *adresa_glave;
187     while (tekuci->sledeci != NULL)
188     {
189         if (tekuci->sledeci->vrednost == broj) {
190             /* tekuci->sledeci treba obrisati,
191                zbog toga sacuvamo njegovu adresu u pomocni */
192             pomocni = tekuci->sledeci;
193             /* Tekucem preusmerimo pokazivac sledeci
194                tako sto preskacemo njegovog trenutnog sledeceg.
195                Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
196             tekuci->sledeci = tekuci->sledeci->sledeci;
197             /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
198                */
199             free(pomocni);
200         } else {
201             /* Ne treba brisati sledeceg, prelazimo na sledeci */
202             tekuci = tekuci->sledeci;
203         }
204     }
205     return;
206 }
207
208 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
209    Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
210    ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
211    pokazivac
212    na glavu liste iz pozivajuće funkcije. */
213 void ispisi_listu(Cvor * glava)
214 {
215     putchar('[');
216     for ( ; glava != NULL; glava = glava->sledeci)
217     {
218         printf("%d", glava->vrednost);
219         if( glava->sledeci != NULL )
220             printf(", ");
221     }
222     printf("]\n");
223 }
224
225 /* Glavni program u kome testiramo sve funkcije za rad sa listama */
226 int main() {
227     Cvor *glava = NULL; /* na pocetku imamo praznu listu */
228     Cvor *trazeni = NULL;
229     int broj;
230
231     /* Testiramo dodavanje na pocetak*/
232     printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+D
233           )\n");

```

```

234     printf("\n\tLista: ");
        ispisi_listu(glava);

236     while(scanf("%d",&broj)>0)
    {
238         dodaj_na_pocetak_liste(&glava, broj);
        printf("\n\tLista: ");
240         ispisi_listu(glava);
    }

242     printf("\nUnesite element koji se trazi u listi: ");
244     scanf("%d", &broj);

246     trazenipretrazi_listu(glava, broj);
    if(trazenipretrazi_listu(glava, broj)==NULL)
248         printf("Element NIJE u listi!\n");
    else
250         printf("Trazeni broj %d je u listi!\n", trazenipretrazi_listu(glava, broj)->vrednost);

252     /* brisemo elemente iz liste cije polje vrednost je jednako
254        broju procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
256     scanf("%d", &broj);

258     obrisi_element(&glava, broj);

260     printf("Lista nakon brisanja: ");
        ispisi_listu(glava);

262     oslobodi_listu(&glava);

264     return 0;
266 }

```

## Rešenje 4.10

## Rešenje 4.11

## Rešenje 4.12

```

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

4 /* NAPOMENA:
    Jednostruko povezana lista je struktura podataka
6 koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
    podatak (odredjenog tipa) i pokazivac na sledeci cvor u
8 sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak

```

## 4 Dinamičke strukture podataka

---

```
10  liste (bez glave) je takodje lista, i naziva se rep liste.
12  Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
    baratanja listom mi cuvamo samo pokazivac na glavu liste.
14  Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
    sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
16  pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
    preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
18  element u listi nema sledeci element: u tom slucaju se
    njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
    lista se predstavlja NULL pokazivacem.

20  Prednost koriscenja povezanih lista u odnosu na dinamicki
    niz je u tome sto se elementi mogu efikasno umetati i brisati
22  sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
    premanjanjem elemenata. Nedostatak ovakvog pristupa je to sto
24  ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
    elementi moraju obradjivati redom (iteracijom kroz listu).

26

28  Prilikom promene liste (dodavanje novog elementa, brisanje
    elementa,
    premanjanje elemenata, itd.) postoji mogucnost da glava liste bude
30  promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
    U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
32  promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
    pozivajucoj funkciji vrati azurirana informacija o adresi glave
    liste.

34

36  Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
    pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
    cuva
    adresu glave i koju, eventualno, treba azurirati.
38  Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
    promeni promenljivu pozivajuce funkcije direktno. Npr:
40      funkcija_za_promenu(&pok, ...);
    */

42  /* Struktura koja predstavlja cvor liste */
44  typedef struct cvor {
    /* Podatak koji cvor sadrzi */
    int vrednost;
    /* Pokazivac na sledeci cvor liste */
    struct cvor *sledeci;
48  } Cvor;

50

52
54  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
    novog cvora inicijalizuje na broj, dok pokazivac na
    sledeci cvor u novom cvoru postavlja na NULL.
56  Funkcija vraca pokazivac na novokreirani cvor ili NULL
    ako alokacija nije uspesno izvorsena. */
```

```

58 Cvor * napravi_cvor(int broj) {
    Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60     if( novi == NULL )
        return NULL;

62     /* Inicijalizacija polja u novom cvoru */
64     novi->vrednost = broj;
    novi->sledeci = NULL;
66     return novi;
    }

68

70

72 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
    ciji se pocetni cvor nalazi na adresi adresa_glave. */
74 void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;

76     /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
        */
78     while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
80         a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
82         free(*adresa_glave);
        /* Sledeci element je nova glava liste */
84         *adresa_glave = pomocni;
    }
86 }

88

90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
    alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
92 za listu ciji pocetni cvor se nalazi na adresi adresa_glave. */
void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94     /* Ukoliko je novi NULL */
    if ( novi == NULL ) {
96         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
            */
98         oslobodi_listu(adresa_glave);
100         exit(EXIT_FAILURE);
    }
102 }

104

```



```
106 /* Funkcija dodaje novi cvor na pocetak liste.  
    Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na  
    pocetak */  
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {  
    /* Kreiramo nov cvor i proveravamo da li je bilo greske pri  
    alokaciji */  
110    Cvor *novi = napravi_cvor(broj);  
    prover_i_alokaciju(adresa_glave, novi);  
112  
    /* Uvezujemo novi cvor na pocetak */  
114    novi->sledeci = *adresa_glave;  
    /* Nov cvor je sada nova glava liste */  
116    *adresa_glave = novi;  
    }  
118  
120  
    /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,  
    ili NULL ukoliko je lista prazna */  
122 Cvor* pronadji_poslednji (Cvor* glava) {  
    /* ako je lista prazna, nema ni poslednjeg cvor  
    i u tom slucaju vracamo NULL.*/  
124    if( glava == NULL)  
        return NULL;  
126  
    /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo  
    pokazivac  
130    glava na taj sledeci element. Kada izadjemo iz petlje,  
    glava ce pokazivati na element liste koji nema sledeceg,  
132    tj. poslednji element liste je. Zato vracamo vrednost  
    pokazivaca glava.  
134  
    glava je argument funkcije i njegove promene nece se odraziti  
    na  
    vrednost pokazivaca glava u pozivajucoj funkciji. */  
136    while (glava->sledeci != NULL)  
        glava = glava->sledeci;  
138  
    return glava;  
140 }  
142  
    /* Funkcija trazi u listi element cija je vrednost jednaka datom  
    broju.  
144    Vraca pokazivac na cvor liste u kome je sadržan traženi broj  
    ili NULL u slucaju da takav element ne postoji u listi. */  
146 Cvor* pretrazi_listu(Cvor * glava, int broj) {  
    for ( ; glava != NULL; glava = glava->sledeci)  
148        /* Pronasli smo */  
        if (glava->vrednost == broj)  
150            return glava;
```

```
152     /* Nema trazenog broja u listi i vracamo NULL*/
153     return NULL;
154 }
155
156
157
158 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
159 Funkcija azurira pokazivac na glavu liste (koji moze biti
160 promenjen u slucaju da se obrise stara glava) */
161 void obrisi_element(Cvor ** adresa_glave, int broj) {
162     Cvor *tekuci = NULL;
163     Cvor *pomocni = NULL;
164
165     /* Brisemo sa pocetka liste sve eventualne cvorove
166     koji su jednaki datom broju, i azuriramo pokazivac na glavu */
167     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj
168     ) {
169         /* Sacuvamo adresu repa liste pre oslobadjanja glave */
170         pomocni = (*adresa_glave)->sledeci;
171         free(*adresa_glave);
172         *adresa_glave = pomocni;
173     }
174
175     /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
176     if ( *adresa_glave == NULL)
177         return;
178
179     /* Od ovog trenutka se u svakom koraku nalazimo
180     na tekucem cvoru koji je razlicit od trazenog
181     broja (kao i svi levo od njega). Poredimo
182     vrednost sledeceg cvora (ako postoji) sa trazenim
183     brojem i brisemo ga ako je jednak, a prelazimo na
184     sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
185     dok ne dodjemo do poslednjeg cvora. */
186     tekuci = *adresa_glave;
187     while (tekuci->sledeci != NULL)
188         if (tekuci->sledeci->vrednost == broj) {
189             /* tekuci->sledeci treba obrisati,
190             zbog toga sacuvamo njegovu adresu u pomocni */
191             pomocni = tekuci->sledeci;
192             /* Tekucem preusmerimo pokazivac sledeci
193             tako sto preskakemo njegovog trenutnog sledeceg.
194             Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
195             tekuci->sledeci = tekuci->sledeci->sledeci;
196             /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
197             */
198             free(pomocni);
199         } else {
200             /* Ne treba brisati sledeceg, prelazimo na sledeci */
201             tekuci = tekuci->sledeci;
202         }
203 }
```

```
202     return;
203 }
204
205 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
206    Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
207    ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
208    pokazivac
209    na glavu liste iz pozivajuće funkcije. */
210 void ispisi_listu(Cvor * glava)
211 {
212     putchar('[');
213     for ( ; glava != NULL; glava = glava->sledeci)
214     {
215         printf("%d", glava->vrednost);
216         if( glava->sledeci != NULL )
217             printf(", ");
218     }
219     printf("]\n");
220 }
221
222
223
224 /* Glavni program u kome testiramo sve funkcije za rad sa listama */
225 int main() {
226     Cvor *glava = NULL; /* na pocetku imamo praznu listu */
227     Cvor *trazeni = NULL;
228     int broj;
229
230     /* Testiramo dodavanje na pocetak*/
231     printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+D
232            )\n");
233     printf("\n\tLista: ");
234     ispisi_listu(glava);
235
236     while(scanf("%d",&broj)>0)
237     {
238         dodaj_na_pocetak_liste(&glava, broj);
239         printf("\n\tLista: ");
240         ispisi_listu(glava);
241     }
242
243     printf("\nUnesite element koji se trazi u listi: ");
244     scanf("%d", &broj);
245
246     trazeni=pretrazi_listu(glava, broj);
247     if(trazeni==NULL)
248         printf("Element NIJE u listi!\n");
249     else
250         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
```

```

252     /* brisemo elemente iz liste cije polje vrednost je jednako
253     broju procitanom sa ulaza */
254     printf("\nUnesite element koji se brise iz liste: ");
255     scanf("%d", &broj);
256
257     obrisi_element(&glava, broj);
258
259     printf("Lista nakon brisanja: ");
260     ispisi_listu(glava);
261
262     oslobodi_listu(&glava);
263
264     return 0;
265 }

```

## Rešenje 4.13

## Rešenje 4.14

## Rešenje 4.15

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* a) Struktura kojom se predstavlja cvor binarnog
5  pretrazivackog stabla */
6  typedef struct cvor {
7      int broj;
8      struct cvor *levo;
9      struct cvor *desno;
10 }Cvor;
11
12 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
13 inicijalizuje polja strukture i vraća pokazivac na novi
14 cvor */
15 Cvor * napravi_cvor(int broj)
16 {
17     /* Alociramo memoriju */
18     Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
19     if (novi == NULL)
20         return NULL;
21     /* Inicijalizujemo polja novog cvora. */
22     novi->broj = broj;
23     novi->levo = NULL;
24     novi->desno = NULL;
25
26     /* Vracamo adresu novog cvora. */
27     return novi;

```

```

}
29
/* Funkcija koja proverava uspesnost kreiranja novog cvora
31   stabla */
void prover_i_alokaciju(Cvor * novi_cvor)
33 {
    /* Ukoliko je cvor neuspesno kreiran */
35     if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa
37     */
        fprintf(stderr, "Malloc greska za novi cvor!\n");
39         exit(EXIT_FAILURE);
    }
41 }

/* c) Funkcija koja dodaje zadati broj u stablo */
43 void dodaj_u_stablo(Cvor ** adresa_korena, int broj){
45
    /* Ako je stablo prazno */
47     if (*adresa_korena == NULL) {
        /* Kreiramo novi cvor */
        Cvor * novi = napravi_cvor(broj);
        prover_i_alokaciju(novi);
51
        /* I proglašavamo ga korenom stabla */
        *adresa_korena = novi;
53
        return;
55     }
57

59     /* U suprotnom trazimo odgovarajucu poziciju za zadati broj */

61     /* Ako je zadata vrednost manja od vrednosti korena */
    if (broj < (*adresa_korena)->broj)
63         /* Dodajemo broj u levo podstablo */
        dodaj_u_stablo(&(*adresa_korena)->levo, broj);
65     else
        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa ga
67     dodajemo u desno podstablo */
        dodaj_u_stablo(&(*adresa_korena)->desno, broj);
69 }

/*
71 d) Funkcija koja proverava da li se zadati broj nalazi
    u stablu
73 */

75 Cvor * pretrazi_stablo(Cvor * koren, int broj)
{
77     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u
    * njemu */
79     if (koren == NULL)
```

```

    return NULL;
81
/* Ako je trazena vrednost sadrazana u korenu */
83 if (koren->broj == broj) {
    /* Prekidamo pretragu */
85     return koren;
}
87
/* Inace, ako je broj manji od vrednosti sadrzane u korenu */
89 if (broj < koren->broj)
    /* Pretragu nastavljamo u levom podstablu */
91     return pretrazi_stablo(koren->levo, broj);
else
93     /* U suprotnom, pretragu nastavljamo u desnom podstablu */
    return pretrazi_stablo(koren->desno, broj);
95 }

97 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost
u stablu */
99 Cvor * pronadji_najmanji(Cvor * koren)
{
101     /* Ako je stablo prazno, prekidamo pretragu */
    if (koren == NULL)
103         return NULL;

105     /* Vrednosti koje su manje od vrednosti u korenu stabla
        nalaze se levo od njega */
107
109     /* Ako je koren cvor koji nema levo podstablo, onda on
        sadrzi najmanju vrednost */
    if (koren->levo == NULL)
111         return koren;

113     /* Inace, pretragu treba nastaviti u levom podstablu */
    return pronadji_najmanji(koren->levo);
115 }

117 /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u stablu
*/
119 Cvor * pronadji_najveci(Cvor * koren)
{
121     /* Ako je stablo prazno, prekidamo pretragu */
    if (koren == NULL)
123         return NULL;

125     /* Vrednosti koje su veće od vrednosti u korenu stabla
        nalaze se desno od njega */
127
129     /* Ako je koren cvor koji nema desno podstablo, onda on
        sadrzi najveću vrednost */
    if (koren->desno == NULL)
131         return koren;

```

```
133  /* Inace, pretragu treba nastaviti u desnom podstablu */
      return pronadji_najveci(koren->desno);
135
136  }
137
138  /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
139  void obrisi_element(Cvor ** adresa_korena, int broj)
140  {
141      Cvor * pomocni_cvor = NULL;
142
143      /* ako je stablo prazno, brisanje nije primenljivo pa mozemo
      prekinuti rad funkcije */
144      if (*adresa_korena == NULL)
145          return;
146
147      /* Ako je vrednost koju treba obrisati manja od vrednosti
148      u korenu stabla,
149      ona se eventualno nalazi u levom podstablu,
150      pa treba rekurzivno primeniti postupak na
151          levo
152      podstablo. Koren ovako modifikovanog stabla je nepromenjen. */
153      if (broj < (*adresa_korena)->broj) {
154          obrisi_element(&(*adresa_korena)->levo, broj);
155          return;
156      }
157
158      /* Ako je vrednost koju treba obrisati veca od vrednosti u
159      korenu stabla,
160      ona se eventualno nalazi u
161          desnom
162      podstablu
163      pa treba rekurzivno primeniti
164          postupak na
165      desno
166      podstablo. Koren ovako
167          modifikovanog stabla je
168      nepromenjen. */
169      if ((*adresa_korena)->broj < broj) {
170          obrisi_element(&(*adresa_korena)->desno, broj);
171          return;
172      }
173
174      /* Slede podslucajevi vezani za slucaj kada je vrednost
175      u
176      korenu jednaka broju koji se brise (tj. slucaj
177          kada
178      treba obrisati koren) */
179
180      /* Ako koren nema sinova, tada se on prosto brise, i
181          rezultat je prazno stablo (vracamo NULL) */
182      if ((*adresa_korena)->levo == NULL
```

```

185         &&(*adresa_korena)->desno == NULL) {
186             free(*adresa_korena);
187             *adresa_korena = NULL;
188             return;
189         }
190
191         /* Ako koren ima samo levog sina, tada se brisanje vrši
192            tako sto obrisemo koren, a novi koren postaje levi sin */
193         if ((*adresa_korena)->levo != NULL
194             &&(*adresa_korena)->desno == NULL) {
195             pomocni_cvor = (*adresa_korena)->levo;
196             free(*adresa_korena);
197             *adresa_korena = pomocni_cvor;
198             return;
199         }
200
201         /* Ako koren ima samo desnog sina, tada se brisanje vrši
202            tako sto obrisemo koren, a novi koren postaje desni sin
203            */
204         if ((*adresa_korena)->desno != NULL
205             &&(*adresa_korena)->levo == NULL) {
206             pomocni_cvor = (*adresa_korena)->desno;
207             free(*adresa_korena);
208             *adresa_korena = pomocni_cvor;
209             return;
210         }
211
212         /* Slučaj kada koren ima oba sina. Tada se brisanje vrši
213            na sledeći način:
214            - najpre se potraži sledbenik
215              korena (u smislu poretka) u stablu. To je upravo po
216              vrednosti najmanji cvor u desnom podstablu.
217            On se može
218            pronaci npr. funkcijom pronadji_najmanji().
219            Nakon
220            toga se u koren smesti vrednost tog cvora, a
221              u
222            taj cvor
223            se smesti vrednost korena (tj. broj koji
224              se
225            briše).
226
227            - Onda se prosto rekursivno pozove
228              funkcija
229            za brisanje
230            na desno podstablo. S obzirom
231              da u njemu
232            treba
233            obrisati
234            najmanji element, a on
235              zasigurno ima
236            najviše

```



```
237     jednog potomka, jasno je da ce
        njegovo
239     brisanje biti
        obavljeno na
        jedan od
241     jednostavnijih
        nacina koji su
243     gore opisani. */
        pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
245     (*adresa_korena)->broj = pomocni_cvor->broj;
        pomocni_cvor->broj = broj;
247     obrisi_element(&(*adresa_korena)->desno, broj);
    }

249     /* h) Funkcija ispisuje stablo u infiksnoj notaciji ( Levo
        postablo - Koren - Desno podstablo ) */
251     void ispisi_stablo_infiksno(Cvor * koren)
253     {
        /* Ako stablo nije prazno */
255     if (koren != NULL) {
        /* Prvo ispisujemo sve cvorove levo od korena */
257         ispisi_stablo_infiksno(koren->levo);

259         /* Ispisujemo vrednost u korenu */
        printf("%d ", koren->broj);

261         /* Na kraju ispisujemo cvorove desno od korena */
263         ispisi_stablo_infiksno(koren->desno);
    }

265 }

267     /* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren - Levo
        podstablo - Desno podstablo ) */
269     void ispisi_stablo_prefiksno(Cvor * koren)
271     {
        /* Ako stablo nije prazno */
273     if (koren != NULL) {

275         /* Prvo ispisujemo vrednost u korenu */
        printf("%d ", koren->broj);

277         /* Ispisujemo sve cvorove levo od korena */
279         ispisi_stablo_prefiksno(koren->levo);

281         /* Na kraju ispisujemo sve cvorove desno od korena */
        ispisi_stablo_prefiksno(koren->desno);
283     }

285 }

287     /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo
```

```

    podstablo - Desno postablo - Koren) */
289 void ispisi_stablo_postfiksno(Cvor * koren)
{
291     /* Ako stablo nije prazno */
    if (koren != NULL) {
293
        /* Prvo ispisujemo sve cvorove levo od korena */
295         ispisi_stablo_postfiksno(koren->levo);

        /* Ispisujemo sve cvorove desno od korena */
297         ispisi_stablo_postfiksno(koren->desno);

        /* Na kraju ispisujemo vrednost u korenu */
301         printf("%d ", koren->broj);
    }
303
305 }

307 /*
    k) Funkcija koja oslobadja memoriju zauzetu stablom.
309 */

311 void oslobodi_stablo(Cvor ** adresa_korena)
{
313     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*adresa_korena == NULL)
315         return;

317     /* U suprotnom rekurzivno oslobadjamo memoriju koje zauzima najpre
    levo, a zatim i desno podstablo */
319     oslobodi_stablo(&(*adresa_korena)->levo);
    oslobodi_stablo(&(*adresa_korena)->desno);
321

    /* Oslobadjamo memoriju koju zauzima koren */
323     free(*adresa_korena);

325     /* I proglašavamo stablo praznim */
    *adresa_korena = NULL;
327 }

329 int main()
{
331     Cvor * koren;
    int n;
333     Cvor * trazeni_cvor;

335     /* Proglašavamo stablo praznim */
    koren = NULL;
337

    /* Dodajemo vrednosti u stablo */
339     printf("Unesite brojeve (CTRL+D za kraj unosa): ");

```

```
341     while (scanf("%d", &n) != EOF) {
342         dodaj_u_stablo(&koren, n);
343     }
344     /* Generisemo trazene ispise: */
345     printf("\nInfiksni ispis: ");
346     ispisi_stablo_infiksno(koren);
347     printf("\nPrefiksni ispis: ");
348     ispisi_stablo_prefiksno(koren);
349     printf("\nPostfiksni ispis: ");
350     ispisi_stablo_postfiksno(koren);
351
352     /* Demonstriramo rad funkcije za pretragu */
353     printf("\nTrazi se broj: ");
354     scanf("%d", &n);
355     trazeni_cvor = pretrazi_stablo(koren, n);
356     if (trazeni_cvor == NULL)
357         printf("Broj se ne nalazi u stablu!\n");
358     else
359         printf("Broj se nalazi u stablu!\n");
360
361     /* Demonstriramo rad funkcije za brisanje */
362     printf("Brise se broj: ");
363     scanf("%d", &n);
364     obrisi_element(&koren, n);
365     printf("Rezultujuce stablo: ");
366     ispisi_stablo_infiksno(koren);
367     printf("\n");
368
369     /* Oslobadjamo memoriju zauzetu stablom */
370     oslobodi_stablo(&koren);
371
372     /* Prekidamo sa programom */
373     return 0;
374 }
375 }
```

### Rešenje 4.16

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX 50
7
8  /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
9  pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
11     char *rec;
```

```

13     int brojac;
14     struct cvor *levo;
15     struct cvor *desno;
16 } Cvor;

17 /* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec) {
19
20     /* Alociramo memoriju za novi cvor */
21     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
22     if (novi_cvor == NULL)
23         return NULL;

24     /* Alociramo memoriju za zadatu rec: potrebno je rezervirati
25     memoriju za svaki karakter reci ukljucujuci i terminirajucu nulu */
26     novi_cvor->rec = (char*)malloc((strlen(rec)+1)*sizeof(char));
27     if( novi_cvor->rec == NULL) {
28         free(novi_cvor);
29         return NULL;
30     }

31     /* Inicijalizujemo polja u novom cvoru */
32     strcpy(novi_cvor->rec, rec);
33     novi_cvor->brojac = 1;
34     novi_cvor->levo = NULL;
35     novi_cvor->desno = NULL;

36     /* Vracamo adresu novog cvora */
37     return novi_cvor;
38 }

39 /* 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
45         izvršavanje programa */
46         fprintf(stderr, "Malloc greska za novi cvor!\n");
47         exit(EXIT_FAILURE);
48     }
49 }

50 /* Funkcija koja dodaje novu rec u stablo. */
51 void dodaj_u_stablo(Cvor** adresa_korena, char *rec) {
52
53     /* Ako je stablo prazno */
54     if( *adresa_korena == NULL) {
55         /* Kreiramo novi cvor */
56         Cvor* novi = napravi_cvor(rec);
57         prover_i_alokaciju(novi);
58         /* i proglašavamo ga korenom stabla */

```

```

        *adresa_korena = novi;
65     return;
    }

67     /* U suprotnom trazimo odgovarajucu poziciju za novu rec */

69     /* Ako je rec leksikografski manju od reci u korenu ubacujemo
71     je u levo podstablo */
    if (strcmp(rec, (*adresa_korena)->rec) < 0)
73         dodaj_u_stablo(&(*adresa_korena)->levo, rec);
    else
75         /* Ako je rec leksikografski veca od reci u korenu
ubacujemo je u desno podstablo */
77         if (strcmp(rec, (*adresa_korena)->rec) > 0)
            dodaj_u_stablo(&(*adresa_korena)->desno, rec);
79         else
            /* Ako je rec jednaka reci u korenu, uvecavamo njen
81     broj pojavljivanja */
            (*adresa_korena)->brojac++;
83     }

85     /* Funkcija koja oslobadja memoriju zauzetu stablom */
87     void oslobodi_stablo(Cvor** adresa_korena) {

89         /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
        if(*adresa_korena == NULL)
91             return;

93         /* Inace ...*/
        /* Oslobadjamo memoriju zauzetu levim podstablom */
95         oslobodi_stablo(&(*adresa_korena)->levo);
        /* Oslobadjamo memoriju zauzetu desnim podstablom */
97         oslobodi_stablo(&(*adresa_korena)->desno);

99         /* Oslobadjamo memoriju zauzetu korenom */
        free((*adresa_korena)->rec);
101        free(*adresa_korena);

103        /* Proglasavamo stablo praznim */
        *adresa_korena = NULL;
105    }

107

109    /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
sa najvećim brojem pojavljivanja) */
111    Cvor *nadj_i_najfrekventniju_rec(Cvor * koren) {

113        Cvor *max, *max_levo, *max_desno;

115        /* Ako je stablo prazno, prekidamo sa pretragom */

```

```

117     if (koren == NULL)
        return NULL;

119     /* Pronalazimo najfrekventniju reci u levom podstablu */
    max_levo = nadji_najfrekventniju_rec(koren->levo);

121     /* Pronalazimo najfrekventniju reci u desnom podstablu */
123     max_desno = nadji_najfrekventniju_rec(koren->desno);

125     /* Trazimo maksimum vrednosti pojavljivanja reci iz
    levog podstabla, korena i desnog podstabla */
127     max = koren;
    if (max_levo != NULL && max_levo->brojac > max->brojac)
129         max = max_levo;
    if (max_desno != NULL && max_desno->brojac > max->brojac)
131         max = max_desno;

133     /* Vracamo adresu cvora sa najvećim brojcem */
    return max;
135 }

137
139 /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
    pracene brojem pojavljivanja */
void prikazi_stablo(Cvor * koren) {
141
    /* Ako je stablo prazno, završavamo sa ispisom */
143     if (koren == NULL)
        return;

145
    /* Zbog leksikografskog poretka, prvo ispisujemo sve reci iz
    levog podstabla */
147     prikazi_stablo(koren->levo);
    /* Zatim ispisujemo rec iz korena */
149     printf("%s: %d\n", koren->rec, koren->brojac);
    /* I nastavljamo sa ispisom reci iz desnog podstabla */
151     prikazi_stablo(koren->desno);
153 }

155 /* Funkcija ucitava sledeću rec iz zadate datoteke i upisuje je
    u niz rec. Maksimalna dužina reci je određena argumentom max.
    Funkcija vraća EOF ako nema više reci ili 0 u suprotnom.
    Rec je niz malih ili velikih slova.*/
157
159 int procitaj_rec(FILE * f, char rec[], int max) {
    /* karakter koji citamo */
161     int c;
    /* indeks pozicije na koju se smesta procitani karakter */
163     int i = 0;

165     /* Sve dok ima mesta za još jedan karakter u nizu
        i dokle god nismo stigli do kraja datoteke... */
167     while (i < max - 1 && (c = fgetc(f)) != EOF) {

```

```

169         /* Proveravamo da li je procitani karakter slovo */
171         if (isalpha(c))
173             /* Ako jeste, smestamo ga u niz - pritom vrsimo
konverziju u mala slova jer program treba da bude neosetljiv na
razliku izmedju malih i velikih slova */
175             rec[i++] = tolower(c);
177         else
179             /* Ako nije, proveravamo da li smo procitali barem jedno
slovo nove rece */
181             /* Ako jesmo prekidamo sa citanjem */
183             if (i > 0)
185                 break;
187             /* U suprotnom idemo na sledecu iteraciju */
189         }

191     /* Dodajemo na rec terminirajucu nulu */
193     rec[i] = '\0';

195     /* Vracamo 0 ako smo procitali rec, EOF u suprotnom */
197     return i > 0 ? 0 : EOF;
199 }

201 int main(int argc, char **argv) {
203     Cvor *koren = NULL, *max;
205     FILE *f;
207     char rec[MAX];

209     /* Proveravamo da li je navedeno ime datoteke prilikom
pokretanja programa */
211     if (argc < 2) {
213         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
215         exit(EXIT_FAILURE);
217     }

219     /* Otvaramo datoteku iz koje citamo reci */
221     if ((f = fopen(argv[1], "r")) == NULL) {
223         fprintf(stderr, "fopen() greska pri otvaranju %s\n", argv[1]);
225         exit(EXIT_FAILURE);
227     }

229     /* Ucitavamo reci iz datoteke i smestamo u binarno stablo pretrage.
*/
231     while (procitaj_rec(f, rec, MAX) != EOF)
233         dodaj_u_stablo(&koren, rec);

235     /* Posto smo zavrшили sa citanjem reci zatvaramo datoteku */
237     fclose(f);

239     /* Prikazujemo sve reci iz teksta i brojeve njihovih
pojavlјivanja. */

```

```

221 prikazi_stablo(koren);

223 /* Pronalazimo najfrekventniju rec */
max = najdi_najfrekventniju_rec(koren);

225 /* Ako takve reci nema... */
if (max== NULL)
227     /* Ispisujemo odgovarajuće obavestjenje */
    printf("U tekstu nema reci!\n");
229 else
    /* Inace, ispisujemo broj pojavljivanja reci */
231     printf("Najcesca rec: %s (pojavljuje se %d puta)\n", max->rec,
max->brojac);
233

235 /* Oslobadjamo dinamicki alociran prostor za stablo */
oslobodi_stablo(&koren);
237

239 /* Završavamo sa programom */
return 0;
}

```

### Rešenje 4.17

```

#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
/* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
38 void proveri_alokaciju(Cvor * novi_cvor)
40 {
    /* Ukoliko je cvor neuspesno kreiran */
42     if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
           programa */
44         fprintf(stderr, "Malloc greska za novi cvor!\n");
46         exit(EXIT_FAILURE);
    }
48 }

50 /* Funkcija koja dodaje novu osobu i njen broj telefona u
   stablo. */
52 void
dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
54              char *telefon)
{
56     /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
58         /* Kreiramo novi cvor */
        Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
60         proveri_alokaciju(novi);
        /* i proglašavamo ga korenom stabla */
62         *adresa_korena = novi;
        return;
64     }

    /* U suprotnom trazimo odgovarajucu poziciju za novi unos */
66     /* Kako pretragu treba vrsiti po imenu i prezimenu, stablo
       treba da bude pretrazivacko po ovom polju */
68     /* Ako je zadato ime i prezime leksikografski manje od imena i
       prezimena sadržanog u korenu, podatke dodajemo u levo
       podstablo */
70     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
72         < 0)
        dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
74                      telefon);
    else
76         /* 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)
```

```

80     dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
81                  telefon);
82 }

84 /* Funkcija koja oslobadja memoriju zauzetu stablom */
85 void oslobodi_stablo(Cvor ** adresa_korena)
86 {
87     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
88     if (*adresa_korena == NULL)
89         return;
90     /* Inace ... */
91     /* Oslobadjamo memoriju zauzetu levim podstablom */
92     oslobodi_stablo(&(*adresa_korena)->levo);
93     /* Oslobadjamo memoriju zauzetu desnim podstablom */
94     oslobodi_stablo(&(*adresa_korena)->desno);
95     /* Oslobadjamo memoriju zauzetu korenom */
96     free(*adresa_korena);
97     /* Proglasavamo stablo praznim */
98     *adresa_korena = NULL;
99 }

100 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
101 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
102    koristiti za proveru da li je stablo lepo kreirano ili ne */
103 void prikazi_stablo(Cvor * koren)
104 {
105     /* Ako je stablo prazno, završavamo sa ispisom */
106     if (koren == NULL)
107         return;
108     /* Zbog leksikografskog poretka, prvo ispisujemo podatke iz
109        levog podstabla */
110     prikazi_stablo(koren->levo);
111     /* Zatim ispisujemo podatke iz korena */
112     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
113     /* I nastavljamo sa ispisom podataka iz desnog podstabla */
114     prikazi_stablo(koren->desno);
115 }

116

117 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
118    ime i prezime i broj telefona u odgovarajuće nizove.
119    Maksimalna dužina imena i prezimena određena je konstantom
120    MAX_IME_PREZIME, a maksimalna dužina broja telefona
121    konstantom MAX_CIFARA. Funkcija vraća EOF ako nema više
122    kontakata ili 0 u suprotnom. */
123 int procitaj_kontakt(FILE * f, char *ime_i_prezime,
124                     char *telefon)
125 {
126     int c;
127     int i = 0;
128     /* Linije datoteke koje obradjujemo su formata Ime Prezime
129        BrojTelefona */
130     /* Preskacemo eventualne praznine sa pocetka linije datoteke */

```

```

132 while ((c = fgetc(f)) != EOF && isspace(c));
/* Prvo procitano slovo upisujemo u ime i prezime */
134 if (!feof(f))
    ime_i_prezime[i++] = c;
136 /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
cifre, tako da cemo citanje forsirati sve dok ne naidjemo na
138 cifru. Pri tom cemo voditi racuna da li ima dovoljno mesta za
smestanje procitanog karaktera i da slucajno ne dodjemo do
140 kraja datoteke */
while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
142     if (!isdigit(c))
        ime_i_prezime[i++] = c;
144     else if (i > 0)
        break;
146 }
/* Upisujemo terminirajucu nulu na mesto poslednjeg procitanog
148 blanko karaktera */
ime_i_prezime[--i] = '\0';
150 /* I pocinjemo sa citanjem broja telefona */
i = 0;
152 /* Upisujemo cifru koju smo vec procitali */
telefon[i++] = c;
154 /* I citamo peostale cifre - naznaka kraja ce nam biti pojava
karaktera cije prisustvo nije dozvoljeno u broju telefona */
156 while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
    if (c == '/' || c == '-' || isdigit(c))
158         telefon[i++] = c;
    else
160         break;
/* Upisujemo terminirajucu nulu */
162 telefon[i] = '\0';
/* Vracamo 0 ako smo procitali kontakt, EOF u suprotnom */
164 return !feof(f) ? 0 : EOF;
}

166 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
prezimenom */
168 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
170 {
    /* Ako je imenik prazan, završavamo sa pretragom */
172     if (koren == NULL)
        return NULL;
174     /* Ako je trazeno ime i prezime sadržano u korenu, takodje
završavamo sa pretragom */
176     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
        return koren;
178     /* Ako je zadato ime i prezime leksikografski manje od
vrednosti u korenu pretragu nastavljamo levo */
180     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
        return pretrazi_imenik(koren->levo, ime_i_prezime);
182     else
        /* u suprotnom, pretragu nastavljamo desno */

```

```

184     return pretrazi_imenik(koren->desno, ime_i_prezime);
185 }
186
187 int main(int argc, char **argv)
188 {
189     char ime_datoteke[MAX_IME_DATOTEKE];
190     Cvor *koren = NULL;
191     Cvor *trazeni;
192     FILE *f;
193     char ime_i_prezime[MAX_IME_I_PREZIME];
194     char telefon[MAX_CIFARA];
195     char c;
196     int i;
197
198     /* Ucitavamo ime datoteke i pripremamo je za citanje */
199     printf("Unesite ime datoteke: ");
200     scanf("%s", ime_datoteke);
201     if ((f = fopen(ime_datoteke, "r")) == NULL) {
202         fprintf(stderr, "fopen() greska prilikom otvaranja
203 %s\n", ime_datoteke);
204         exit(EXIT_FAILURE);
205     }
206
207     /* Ucitavamo podatke iz datoteke i smestamo kontakte u binarno
208        stablo pretrage. */
209     while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
210         dodaj_u_stablo(&koren, ime_i_prezime, telefon);
211
212     /* Posto smo zavrшили sa citanjem podataka zatvaramo datoteku */
213     fclose(f);
214
215     /* Omogucavamo pretragu imenika */
216     while (1) {
217         /* Ucitavamo ime i prezime */
218         printf("Unesite ime i prezime: ");
219         i = 0;
220         while ((c = getchar()) != '\n')
221             ime_i_prezime[i++] = c;
222         ime_i_prezime[i] = '\0';
223         /* Ako je korisnik uneo naznaku za kraj pretrage,
224            obustavljamo funkcionalnost */
225         if (strcmp(ime_i_prezime, "KRAJ") == 0)
226             break;
227         /* Inace, ispisujemo rezultat pretrage */
228         trazeni = pretrazi_imenik(koren, ime_i_prezime);
229         if (trazeni == NULL)
230             printf("Broj nije u imeniku!\n");
231         else
232             printf("Broj je: %s \n", trazeni->telefon);
233     }
234
235     /* Oslobadjamo memoriju zauzetu imenikom */

```

```
236   oslobodi_stablo(&koren);
238   /* Završavamo sa programom */
      return 0;
240 }
```

### Rešenje 4.18

```
#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 uspeh, 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
  /* Funkcija kojom se kreira cvor drveta */
22 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
      double matematika, double jezik)
24 {
26     /* Alociramo memoriju za novi cvor */
     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
28     if (novi == NULL)
         return NULL;
30
     /* Inicijalizujemo polja strukture */
32     strcpy(novi->ime, ime);
     strcpy(novi->prezime, prezime);
34     novi->uspeh = uspeh;
     novi->matematika = matematika;
36     novi->jezik = jezik;
     novi->levo = NULL;
38     novi->desno = NULL;
40
     /* Vracamo adresu kreiranog cvora */
     return novi;
42 }
```

```
44 /* Funkcija kojom se proverava uspesnost alociranja memorije */
void prover_i_alokaciju(Cvor * novi_cvor)
46 {
    /* Ako alokacije nije uspesna */
    if (novi_cvor == NULL) {
    48 /* Ispisujemo poruku i prekidamo sa izvršavanjem */
        fprintf(stderr, "Malloc greska za novi cvor!\n");
    50 exit(EXIT_FAILURE);
    52 }
    54 }
    56 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
void oslobodi_stablo(Cvor ** koren)
    58 {
    60 /* Ako je stablo prazno, nema potrebe za oslobadjanjem
        memorije */
    62 if (*koren == NULL)
    64     return;
    66 /* oslobadjamo memoriju zauzetu levim podstablom */
    oslobodi_stablo(&(*koren)->levo);
    68 /* oslobadjamo memoriju zauzetu desnim podstablom */
    70 oslobodi_stablo(&(*koren)->desno);
    72 /* oslobadjamo memoriju zauzetu korenom */
    free(*koren);
    74 /* proglasavamo stablo praznim */
    76 *koren = NULL;
    78 }
    78 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
    80 double uspeh, double matematika,
    82 double jezik)
    {
    84 /* Ako je stablo prazno */
    86 if (*koren == NULL) {
        /* Kreiramo novi cvor */
    88 Cvor *novi =
            napravi_cvor(ime, prezime, uspeh, matematika, jezik);
    90 prover_i_alokaciju(novi);
    92 /* I proglasavamo ga korenom stabla */
        *koren = novi;
    94 return;
    }
```

```
96     }

98     /* Inace, dodajemo cvor u stablo tako da bude sortiran po
    ukupnom broju poena */
100     if (uspeh + matematika + jezik >
        (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
102         dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
                        matematika, jezik);
104     else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
                        matematika, jezik);
106     }
108

110     /* Funkcija ispisuje sadrzaj stabla - ukoliko je vrednost
    argumenta polozili jednaka 0 ispisuju se informacije o
112     ucenicima koji nisu polozili prijemni, a ako je vrednost
    argumenta razlicita od nule, ispisuju se informacije o
114     ucenicima koji su polozili prijemni */
    void stampaj(Cvor * koren, int polozili)
116     {

118         /* Stablo je prazno - prekidamo sa ispisom */
        if (koren == NULL)
120             return;

122         /* Stampamo informacije iz levog podstabla */
        stampaj(koren->levo, polozili);
124

        /* Stampamo informacije iz korenog cvora */
126         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);
        else if (!polozili && koren->matematika + koren->jezik < 10)
132            printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
                koren->prezime, koren->uspeh, koren->matematika,
134                koren->jezik,
                koren->uspeh + koren->matematika + koren->jezik);
136

        /* Stampamo informacije iz desnog podstabla */
138        stampaj(koren->desno, polozili);
    }
140

142     /* Funkcija koja odredjuje koliko studenata nije polozilo
    prijemni ispit */
144     int nisu_polozili(Cvor * koren)
    {
146

        /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
```

```

148     if (koren == NULL)
149         return 0;
150
151     /* Pretragu vrsimo i u levom i u desnom podstablu - ako uslov
152        za polaganje nije ispunjen za koreni cvor, broj studenata
153        uvecavamo za 1 */
154     if (koren->matematika + koren->jezik < 10)
155         return 1 + nisu_položili(koren->levo) +
156             nisu_položili(koren->desno);
157
158     return nisu_položili(koren->levo) +
159         nisu_položili(koren->desno);
160 }
161
162 int main(int argc, char **argv)
163 {
164     FILE *in;
165     Cvor *koren;
166     char ime[MAX], prezime[MAX];
167     double uspeh, matematika, jezik;
168
169     /* Otvaramo datoteku sa rezultatima sa prijemnog za citanje */
170     in = fopen("prijemni.txt", "r");
171     if (in == NULL) {
172         fprintf(stderr, "Greska prilikom citanja podataka!\n");
173         exit(EXIT_FAILURE);
174     }
175
176     /* Citamo podatke i dodajemo ih u stablo */
177     koren = NULL;
178     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
179                 &matematika, &jezik) != EOF) {
180         dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
181                     jezik);
182     }
183
184     /* Zatvaramo datoteku */
185     fclose(in);
186
187     /* Stampamo prvo podatke o ucenicima koji su položili prijemni
188        */
189     stampaj(koren, 1);
190
191     /* Liniju iscrtavamo samo ako postoje učenici koji nisu
192        položili prijemni */
193     if (nisu_položili(koren) != 0)
194         printf("-----\n");
195
196     /* Stampamo podatke o ucenicima koji nisu položili prijemni */
197     stampaj(koren, 0);

```



## 4 Dinamičke strukture podataka

---

```
200  /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);
202
    /* Završavamo sa programom */
204  return 0;
206 }
```

### Rešenje 4.19

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

5  #define MAX_NISKA 51
   #define MAX_DATUM 3

7
   /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i
9   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)
29       return NULL;

31   /* Inicijalizujemo polja strukture */
   strcpy(novi->ime, ime);
33   strcpy(novi->prezime, prezime);
   novi->dan = dan;
35   novi->mesec = mesec;
   novi->godina = godina;
37   novi->levo = NULL;
   novi->desno = NULL;
39

   /* Vracamo adresu novog cvora */
41   return novi;
```

```
43 }
44
45 /* Funkcija koja proverava uspesnost alokacije */
46 void prover_i_alokaciju(Cvor * novi_cvor)
47 {
48     /* Ako memorija nije uspesno alocirana */
49     if (novi_cvor == NULL) {
50         /* Ispisujemo poruku i prekidamo izvršavanje programa */
51         fprintf(stderr, "Malloc greska za novi cvor!\n");
52         exit(EXIT_FAILURE);
53     }
54 }
55
56 /* Funkcija koja oslobadja memoriju zauzetu stablom */
57 void oslobodi_stablo(Cvor ** koren)
58 {
59     /* Stablo je prazno */
60     if (*koren == NULL)
61         return;
62
63     /* Oslobadjamo memoriju zauzetu levim podstablom (ako postoji)
64     */
65     if ((*koren)->levo)
66         oslobodi_stablo(&(*koren)->levo);
67
68     /* Oslobadjamo memoriju zauzetu desnim podstablom (ako
69     postoji) */
70     if ((*koren)->desno)
71         oslobodi_stablo(&(*koren)->desno);
72
73     /* Oslobadjamo memoriju zauzetu korenom */
74     free(*koren);
75
76     /* Proglasavamo stablo praznim */
77     *koren = NULL;
78 }
79
80 /* Funkcija koja dodaje novi cvor u stablo - stablo treba da
81    bude uredjeno po datumu */
82 void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
83                    int dan, int mesec, int godina)
84 {
85     /* Ako je stablo prazno */
86     if (*koren == NULL) {
87
88         /* Kreiramo novi cvor */
89         Cvor *novi_cvor =
90             napravi_cvor(ime, prezime, dan, mesec, godina);
91         prover_i_alokaciju(novi_cvor);
92     }
93 }
```

```
95     /* I proglašavamo ga korenom */
96     *koren = novi_cvor;
97
98     return;
99 }
100
101 /* Kako se ne unosi godina za pretragu, stablo uredjujemo samo
102    po mesecu (i danu u okviru istog meseca) */
103 if (mesec < (*koren)->mesec)
104     dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
105                   godina);
106 else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
107     dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
108                   godina);
109 else
110     dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec,
111                   godina);
112 }
113
114 /* Funkcija vrši pretragu stabla i vraća cvor sa traženim
115    datumom (null ako takav ne postoji). u promenljivu pom ce
116    biti smesten prvi datum (dan i mesec) veci od traženog
117    datuma (null ako takav ne postoji)
118
119 */
120 Cvor *pretrazi(Cvor * koren, int dan, int mesec)
121 {
122     /* Stablo je prazno, obustavljamo pretragu */
123     if (koren == NULL)
124         return NULL;
125
126     /* Nasli smo traženi datum u stablu */
127     if (koren->dan == dan && koren->mesec == mesec)
128         return koren;
129
130     /* Ako je mesec traženog datuma manji od meseca sadržanog u
131        korenu ili ako su meseci isti ali je dan traženog datuma
132        manji od aktuelnog datuma, pretražujemo levo podstablo -
133        pre toga svakako proveravamo da li leva grana postoji - ako
134        ne postoji treba da vratimo prvi sledeći, a to je bas
135        vrednost uocenog korena */
136     if (mesec < koren->mesec
137         || (mesec == koren->mesec && dan < koren->dan)) {
138         if (koren->levo == NULL)
139             return koren;
140         else
141             return pretrazi(koren->levo, dan, mesec);
142     }
143
144     /* inace, nastavljamo pretragu u desnom delu */
145     return pretrazi(koren->desno, dan, mesec);
```

```
147 }
149 int main(int argc, char **argv)
150 {
151     FILE *in;
152     Cvor *koren;
153     Cvor *slavljenik;
154     char ime[MAX_NISKA], prezime[MAX_NISKA];
155     int dan, mesec, godina;
157     /* Proveravamo da li je zadato ime ulazne datoteke */
158     if (argc < 2) {
159         /* Ako nije, ispisujemo poruku i prekidamo sa izvršavanjem
160            programa */
161         printf("Nedostaje ime ulazne datoteke!\n");
162         return 0;
163     }
165     /* Inace, pripremamo datoteku za citanje */
166     in = fopen(argv[1], "r");
167     if (in == NULL) {
168         fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
169         exit(EXIT_FAILURE);
170     }
171     /* I popunjavamo podacima stablo */
172     koren = NULL;
173     while (fscanf
174         (in, "%s %s %d.%d.%d.", ime, prezime, &dan, &mesec,
175          &godina) != EOF)
176         dodaj_u_stablo(&koren, ime, prezime, dan, mesec, godina);
178     /* I zatvaramo datoteku */
179     fclose(in);
181     /* Omogucavamo pretragu podataka */
182     while (1) {
183         /* Ucitavamo novi datum */
184         printf("Unesite datum: ");
185         if (scanf("%d.%d.", &dan, &mesec) == EOF)
186             break;
187         /* Pretražujemo stablo */
188         slavljenik = pretrazi(koren, dan, mesec);
189         /* Ispisujemo pronadjene podatke */
190         if (slavljenik == NULL) {
191             printf("Nema podataka o ovim ni o sledecem rođendanu.\n");
192             continue;
193         }
194     }
195 }
```

```
199      /* Slučaj kada smo pronašli prave podatke */
200      if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
201          printf("Slavljenik: %s %s\n", slavljenik->ime,
202                slavljenik->prezime);
203          continue;
204      }
205
206      /* Slučaj kada smo pronašli podatke o prvom sledećem
207         rođendanu */
208      printf("Slavljenik: %s %s %d.%d.\n", slavljenik->ime,
209          slavljenik->prezime, slavljenik->dan,
210          slavljenik->mesec);
211  }
212
213  /* Oslobadjamo memoriju zauzetu stablom */
214  oslobodi_stablo(&koren);
215
216  /* Prekidamo sa izvršavanjem programa */
217  return 0;
218 }
```

### Rešenje 4.20

```
1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura kojom se predstavlja cvor binarnog pretraživackog stabla
5     */
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 vraća pokazivac na novi cvor */
13  Cvor* napravi_cvor(int broj);
14
15  /* Funkcija koja proverava uspešnost kreiranja novog cvora stabla. */
16  void proveri_alokaciju( Cvor* novi_cvor);
17
18  /* Funkcija koja dodaje zadati broj u stablo */
19  void dodaj_u_stablo(Cvor** adresa_korena, int broj);
20
21  /* Funkcija koja proverava da li se zadati broj nalazi u stablu */
22  Cvor* pretrazi_stablo(Cvor * koren, int broj);
23
24  /* Funkcija koja pronalazi cvor koji sadrži najmanju vrednost
25     u stablu */
26  Cvor * pronadji_najmanji(Cvor * koren);
```

```

28 /* Funkcija koja pronalazi cvor koji sadrzi najveći vrednost u stablu
   */
30 Cvor * pronadji_najveci(Cvor * koren);

32 /* Funkcija koja briše cvor stabla koji sadrži zadati broj */
void obrisi_element(Cvor** adresa_korena, int broj);

34 /* Funkcija koja ispisuje sadržaj stabla u infiksnoj notaciji (levo
36 podstablo - koren - desno podstablo) */
void prikazi_stablo(Cvor * koren);

38 /* Funkcija koja oslobađa memoriju zauzetu stablom */
40 void oslobodi_stablo(Cvor** adresa_korena);

42 #endif

```

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "stabla.h"
   /* Funkcija kojom se kreira novi cvor stabla koji sadrži zadatu
5   vrednost */
   Cvor *napravi_cvor(int broj)
7   {
   /* Alociramo memoriju za novi cvor */
9   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   if (novi == NULL)
11      return NULL;
   /* Inicijalizujemo polja cvora */
13   novi->broj = broj;
   novi->levo = NULL;
15   novi->desno = NULL;
   /* Vracamo adresu novog cvora */
17   return novi;
   }

19
   /* Funkcija koja proverava uspesnost kreiranja novog cvora
21   stabla */
   void prover_i_alokaciju(Cvor * novi_cvor)
23   {
   /* Ukoliko je cvor neuspesno kreiran */
25   if (novi_cvor == NULL) {
   /* Ispisuje se odgovarajuća poruka i prekida izvršavanje
27   programa */
   fprintf(stderr, "Malloc greska za novi cvor!\n");
29   exit(EXIT_FAILURE);
   }
31   }

33 /* Funkcija koja dodaje novi broj u stablo. */
   void dodaj_u_stablo(Cvor ** koren, int broj)
35   {
   /* Ako je stablo prazno */

```

```
37  if (*koren == NULL) {
    /* Kreiramo novi cvor */
39  Cvor *novi = napravi_cvor(broj);
    prover_i_alokaciju(novi);
41  /* i proglašavamo ga korenom stabla */
    *koren = novi;
43  return;
}
45  /* U suprotnom tražimo odgovarajuću poziciju za novi broj */
/* Ako je broj manji od vrednosti sadržane u korenu, ubacujemo
47  ga u levo podstablo */
if (broj < (*koren)->broj)
49  dodaj_u_stablo(&(*koren)->levo, broj);
else
51  /* Inace, ubacujemo broj u desno podstablo */
    dodaj_u_stablo(&(*koren)->desno, broj);
53 }

55 /* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** koren)
57 {
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
59  if (*koren == NULL)
        return;
61  /* Inace ... */
    /* Oslobadjamo memoriju zauzetu levom podstablom */
63  if ((*koren)->levo)
        oslobodi_stablo(&(*koren)->levo);
65  /* Oslobadjamo memoriju zauzetu desnom podstablom */
    if ((*koren)->desno)
        oslobodi_stablo(&(*koren)->desno);
67  /* Oslobadjamo memoriju zauzetu korenom */
    free(*koren);
69  /* Proglašavamo stablo praznim */
    *koren = NULL;
71 }

73 Cvor *pronadji_najmanji(Cvor * koren)
75 {
    /* ako je stablo prazno, prekidamo pretragu */
77  if (koren == NULL)
        return NULL;
79  /* vrednosti koje su manje od vrednosti u korenu stabla nalaze
    se levo od njega */
81  /* ako je koren cvor koji nema levo podstablo, onda on sadrži
    najmanju vrednost */
83  if (koren->levo == NULL)
        return koren;
85  /* inace, pretragu treba nastaviti u levom podstablu */
    return pronadji_najmanji(koren->levo);
87 }
```

```

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 pozivajucoj
106    funkciji, jer u ovoj funkciji koren moze 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 stablue,
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. */
127     if ((*adresa_korena)->broj > n) {
128         obrisi_element(&(*adresa_korena)->levo, n);
129         return;
130     }
131     /* Slede podslucajevi vezani za slucaj kada je vrednost u
132        korenu jednaka broju koji se brise (tj. slucaj kada treba
133        obrisati koren) */
134     /* Ako koren nema sinova, tada se on prosto brise, i rezultat
135        je prazno stablo (vracamo NULL) */
136     if ((*adresa_korena)->levo == NULL
137         && (*adresa_korena)->desno == NULL) {
138         free(*adresa_korena);
139         *adresa_korena = NULL;
140         return;

```



```

141 }
142 /* Ako koren ima samo levog sina, tada se brisanje vrši tako
143    sto obrisemo koren, a novi koren postaje levo sin */
144 if ((*adresa_korena)->levo != NULL
145     && (*adresa_korena)->desno == NULL) {
146     pomocni = (*adresa_korena)->levo;
147     free(*adresa_korena);
148     *adresa_korena = pomocni;
149     return;
150 }
151 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako
152    sto obrisemo koren, a novi koren postaje desno sin */
153 if ((*adresa_korena)->desno != NULL
154     && (*adresa_korena)->levo == NULL) {
155     pomocni = (*adresa_korena)->desno;
156     free(*adresa_korena);
157     *adresa_korena = pomocni;
158     return;
159 }
160 /* Slučaj kada koren ima oba sina. Tada se brisanje vrši na
161    sledeći način: - najpre se potraži sledbenik korena (u
162    smislu poretka) u stablu. To je upravo po vrednosti
163    najmanji cvor u desnom podstablu. On se može pronaći npr.
164    funkcijom pronadji_najmanji(). - Nakon toga se u koren
165    smesti vrednost tog cvora, a u taj cvor se smesti vrednost
166    korena (tj. broj koji se briše). - Onda se prosto
167    rekurzivno pozove funkcija za brisanje na desno podstablu.
168    S obzirom da u njemu treba obrisati najmanji element, a on
169    definitivno ima najviše jednog potomka, jasno je da će
170    njegovo brisanje biti obavljeno na jedan od jednostavnijih
171    načina koji su gore opisani. */
172 pomocni = pronadji_najmanji((*adresa_korena)->desno);
173 (*adresa_korena)->broj = pomocni->broj;
174 pomocni->broj = n;
175 obris_element(&(*adresa_korena)->desno, n);
176 }
177
178 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje
179    elemente u rastućem poretku) */
180 void prikazi_stablo(Cvor * koren)
181 {
182     /* izlaz iz rekurzije */
183     if (koren == NULL)
184         return;
185     prikazi_stablo(koren->levo);
186     printf("%d ", koren->broj);
187     prikazi_stablo(koren->desno);
188 }
189
190 Cvor *pretrazi_stablo(Cvor * koren, int broj)
191 {
192     /* ako je stablo prazno, vrednost se sigurno ne nalazi u njemu

```

```

193  */
194  if (koren == NULL)
195      return NULL;
196  /* ako je trazena vrednost sadrazana u korenu */
197  if (koren->broj == broj) {
198      /* prekidamo pretragu */
199      return koren;
200  }
201  /* inace, ako je broj manji od vrednosti sadrzane u korenu */
202  if (broj < koren->broj)
203      /* pretragu nastavljamo u levom podstablu */
204      return pretrazi_stablo(koren->levo, broj);
205  else
206      /* u suprotnom, pretragu nastavljamo u desnom podstablu */
207      return pretrazi_stablo(koren->desno, broj);
208  }

```

```

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

/* Ukljucujemo biblioteku za rad sa stablima - pogledati uvodni
   zadatak ove glave */
#include "stabla.h"

/* Funkcija koja proverava da li su dva stabla koja sadrze cele
   brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
int identitet(Cvor * koren1, Cvor * koren2)
{
    /* Ako su oba stabla prazna, jednaka su */
    if (koren1 == NULL && koren2 == NULL)
        return 1;

    /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
       jednaka */
    if (koren1 == NULL || koren2 == NULL)
        return 0;

    /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
       vrednosti, mozemo da zakljucimo da se razlikuju */
    if (koren1->broj != koren2->broj)
        return 0;

    /* inace, proveravamo da li vazi i jednakost u levih
       podstabala i desnih podstabala */
    return (identitet(koren1->levo, koren2->levo)
            && identitet(koren1->desno, koren2->desno));
}

int main()
{

```

```
36     int broj;
37     Cvor *koren1, *koren2;
38
39     koren1 = NULL;
40     /* učitavamo elemente prvog stabla */
41     printf("Prvo stablo: ");
42     scanf("%d", &broj);
43     while (broj != 0) {
44         dodaj_u_stablo(&koren1, broj);
45         scanf("%d", &broj);
46     }
47
48     koren2 = NULL;
49     /* učitavamo elemente drugog stabla */
50     printf("Drugo stablo: ");
51     scanf("%d", &broj);
52     while (broj != 0) {
53         dodaj_u_stablo(&koren2, broj);
54         scanf("%d", &broj);
55     }
56
57     /* pozivamo funkciju koja ispituje identitet stabala */
58     if (identitet(koren1, koren2))
59         printf("Stabla jesu identicna.\n");
60     else
61         printf("Stabla nisu identicna.\n");
62
63     /* oslobadjamo memoriju zauzetu stablima */
64     oslobodi_stablo(&koren1);
65     oslobodi_stablo(&koren2);
66
67     /* završavamo sa radom programa */
68     return 0;
69 }
70 }
```

### Rešenje 4.21

```
#include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključujemo 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     /* Izlaz iz rekurzije: ako je stablo prazno nema sta da se kopira
11     */
12     if (koren == NULL){
13         *duplikat = NULL;
```

```

14     return;
15 }
16 /* Dupliramo koren stabla i postavljamo ga da bude koren novog
17 stabla */
18 *duplikat =napravi_cvor(koren->broj);
19 prover_i_alokaciju(*duplikat);
20
21 /* Rekurzivno dupliramo levo podstablo i njegovu adresu cuvamo
22 u pokazivacu na levo podstablo korena duplikata. */
23 kopiraj_stablo(koren->levo, &(*duplikat)->levo);
24
25 /* Rekurzivno dupliramo desno podstablo i njegovu
26 adresu cuvamo u pokazivacu na desno podstablo korena duplikata. */
27 kopiraj_stablo(koren->desno, &(*duplikat)->desno);
28 }
29
30 /* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
31 dobija modifikacijom prvog stabla */
32 void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2) {
33     /* Ako drugo stablo nije prazno */
34     if (koren2 != NULL) {
35         /* dodajemo njegov koren u prvo stablo */
36         dodaj_u_stablo(adresa_korena1, koren2->broj);
37         /* rekurzivno racunamo uniju levog i desnog podstabla drugog
38         stabla sa prvim stablom */
39         kreiraj_uniju(adresa_korena1, koren2->levo);
40         kreiraj_uniju(adresa_korena1, koren2->desno);
41     }
42 }
43
44 /* Funkcija izracunava presek dva stabla - rezultujuce stablo se
45 dobija modifikacijom prvog stabla */
46 void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2) {
47     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
48     if (*adresa_korena1 == NULL)
49         return ;
50
51     /* Kreiramo presek levog i desnog podstabla sa drugim stablom, tj.
52     iz levog i desnog podstabla prvog stabla brisemo sve one elemente
53     koji ne postoje u drugom stablu */
54     kreiraj_presek(&(*adresa_korena1)->levo, koren2);
55     kreiraj_presek(&(*adresa_korena1)->desno, koren2);
56
57     /* Ako se koren prvog stabla ne nalazi u drugom stablu tada ga
58     uklanjamo iz prvog stabla */
59     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
60         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
61 }
62
63 /* Funkcija izracunava razliku dva stabla - rezultujuce stablo se
64 dobija modifikacijom prvog stabla */
65 void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2) {

```

```

66  /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
    if (*adresa_korena1 == NULL)
        return ;
68  /* 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 */
70  kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
72  kreiraj_razliku(&(*adresa_korena1)->desno, koren2);

74  /* Ako se koren prvog stabla nalazi i u drugom stablu
    tada ga uklanjamo iz prvog stabla */
76  if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
    obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
78 }

80 int main() {
    Cvor* koren1;
82  Cvor* koren2;
    Cvor *pomocni = NULL;
84  int n;

86

    /* Ucitavamo elemente prvog stabla: */
88  koren1=NULL;
    printf("Prvo stablo: ");
90  while( scanf("%d",&n) != EOF) {
        dodaj_u_stablo(&koren1,n);
92  }

94  /* Ucitavamo elemente drugog stabla: */
    koren2=NULL;
96  printf("Drugo stablo: ");
    while( scanf("%d",&n) != EOF) {
        dodaj_u_stablo(&koren2,n);
98  }

100

    /* Kreiramo uniju stabala: prvo napravimo kopiju prvog stabla kako
    bi mogli da ga iskoristimo i za preostale operacije */
102  kopiraj_stablo(koren1,&pomocni);
104  kreiraj_uniju(&pomocni, koren2);
    printf("Unija: ");
106  prikazi_stablo(pomocni);
    putchar('\n');
108  /* Oslobadjamo stablo za rezultatom operacije */
    oslobodi_stablo(&pomocni);
110

112  /* Kreiramo presek stabala: prvo napravimo kopiju prvog stabla kako
    bi mogli da ga iskoristimo i za preostale operacije;
    */
114  kopiraj_stablo(koren1,&pomocni);
116  kreiraj_presek(&pomocni, koren2);

```

```

118 printf("Presek: ");
    prikazi_stablo(pomocni);
    putchar('\n');
120 /* Oslobadjamo stablo za rezultatom operacije */
    oslobodi_stablo(&pomocni);

122

124 /* Kreiramo razliku stabala: prvo napravimo
    kopiju prvog stabla kako
126 bi mogli da ga iskoristimo i za preostale operacije;
    */
128 kopiraj_stablo(koren1,&pomocni);
    kreiraj_razliku(&pomocni, koren2);
130 printf("Razlika: ");
    prikazi_stablo(pomocni);
132 putchar('\n');
    /* Oslobadjamo stablo za rezultatom operacije */
134 oslobodi_stablo(&pomocni);

136

138 /* Oslobadjamo i polazna stabla */
    oslobodi_stablo(&koren2);
    oslobodi_stablo(&koren1);

140

142 /* Završavamo sa programom */
    return 0;
}

```

### Rešenje 4.22

```

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. */

13 int kreiraj_niz(Cvor * koren, int a[])
    {
15     int r, s;

17     /* Drvo je prazno - u niz je smesteno 0 elemenata */
    if (koren == NULL)
19         return 0;

21     /* Dodajemo u niz elemente iz levog podstabla */

```

```
    r = kreiraj_niz(koren->levo, a);
23
    /* Tekuca vrednost promenljive r je broj elemenata koji su
25        upisani u niz i na osnovu nje mozemo odrediti indeks novog
        elementa */
27
    /* Smestamo vrednost iz korena */
29    a[r] = koren->broj;

31    /* Dodajemo elemente iz desnog podstabla */
    s = kreiraj_niz(koren->desno, a + r + 1);
33

    /* Racunamo indeks na koji treba smestiti naredni element */
35    return r + s + 1;
}
37

39    /* Funkcija sortira niz tako sto najpre elemente niza smesti u
        stablo, a zatim kreira novi niz prolazeci kroz stablo sa
41        leva u desno.

43        Ovaj nacin sortiranja primer sortiranja koje nije "u mestu "
        kao sto je to slucaj sa ostalim prethodno opisanim
45        algoritmima sortiranja, jer se sortiranje vrši u pomocnoj
        dinamičkoj strukturi, a ne razmenom elemenata niza. */
47

void sortiraj(int a[], int n)
49{
    int i;
51    Cvor *koren;

53    /* Kreiramo stablo smestanjem elemenata iz niza u stablo */
    koren = NULL;
55    for (i = 0; i < n; i++)
        dodaj_u_stablo(&koren, a[i]);
57

    /* Infiksnim obilaskom stabla elemente iz stabla prepisujemo u
59    niz a */
    kreiraj_niz(koren, a);
61

    /* Vise nam stablo nije potrebno i oslobadjamo memoriju */
63    oslobodi_stablo(&koren);
}
65

67 int main()
{
69     int a[MAX];
    int n, i;
71

    /* Ucitavamo dimenziju i elemente niza */
73    printf("n: ");
```

```

scanf("%d", &n);
75 if (n < 0 || n > MAX) {
    printf("Greska: pogresna dimenzija niza!\n");
77     return 0;
}

79 printf("a: ");
81 for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

83
/* Pozivamo funkciju za sortiranje */
85 sortiraj(a, n);

87 /* Ispisujemo rezultat */
for (i = 0; i < n; i++)
89     printf("%d ", a[i]);
printf("\n");

91
/* Prekidamo sa programom */
93 return 0;
}

```

### Rešenje 4.23

```

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 */
    if (koren == NULL)
13         return 0;

15     /* U suprotnom je broj cvorova stabla jednak zbiru broja
       cvorova u levom podstablu i broja cvorova u desnom
17     podstablu - 1 dodajemo zato sto treba racunati i koren */
    return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) +
19     1;
}

21
/* b) Funkcija koja izracunava broj listova stabla */
23 int broj_listova(Cvor * koren)
{

25     /* Ako je stablo prazno, broj listova je nula */
27     if (koren == NULL)

```



```
        return 0;

29
    /* Proveravamo da li je tekuci cvor list */
31    if (koren->levo == NULL && koren->desno == NULL)
        /* i ako jeste vracamo 1 - to ce kasnije zbog rekurzivnih
33        poziva uvecati broj listova za 1 */
        return 1;
35
    /* U suprotnom prebrojavamo listove koje se nalaze u
37    podstablima */
    return broj_listova(koren->levo) + broj_listova(koren->desno);
39 }

41 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
void pozitivni_listovi(Cvor * koren)
43 {
45     /* Slucaj kada je stablo prazno */
    if (koren == NULL)
47         return;

49     /* Ako je cvor list i sadrzi pozitivnu vrednost */
    if (koren->levo == NULL && koren->desno == NULL
51        && koren->broj > 0)
        /* Stampamo ga */
53        printf("%d ", koren->broj);

55     /* Nastavljamo sa stampanjem pozitivnih listova u podstablima */
    pozitivni_listovi(koren->levo);
57    pozitivni_listovi(koren->desno);
    }
59

61 /* d) Funkcija koja izracunava zbir cvorova stabla */
int zbir_cvorova(Cvor * koren)
63 {
65     /* Ako je stablo prazno, zbir cvorova je 0 */
    if (koren == NULL)
67         return 0;

69     /* Inace, zbir cvorova stabla izracunavamo kao zbir korena i
    svih elemenata u podstablima */
71    return koren->broj + zbir_cvorova(koren->levo) +
        zbir_cvorova(koren->desno);
73 }

75 /* e) Funkcija koja izracunava najveći element stabla. */
Cvor *najveci_element(Cvor * koren)
77 {
79     /* Ako je stablo prazno, obustavljamo pretragu */
```

```

81     if (koren == NULL)
        return NULL;

83     /* Zbog prirode pretrazivackog stabla, sigurni smo da su
        vrednosti vece od korena u desnom podstablu */

85     /* Ako desnog podstabla nema */
87     if (koren->desno == NULL)
        /* Najveca vrednost je koren */
89     return koren;

91     /* Inace, najvecu vrednost trazimo jos desno */
    return najveći_element(koren->desno);
93 }

95 /* f) Funckija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
97 {

99     /* Dubina praznog stabla je 0 */
    if (koren == NULL)
101     return 0;

103     /* Izracunavamo dubinu levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

105     /* Izracunavamo dubinu desnog podstabla */
107     int dubina_desno = dubina_stabla(koren->desno);

109     /* dubina stabla odgovara vecoj od dubina podstabala - 1
        dodajemo jer racunamo i koren */
111     return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
113 }

115 /* g) Funckija koja izracunava broj cvorova na i-tom nivou */
int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
117 {

119     /* ideja je da ste spustamo kroz drvo sve dok ne stignemo do
        trazenog nivoa */

121     /* Ako nema vise cvorova, ne mozemo da se spustamo niz stablo */
    if (koren == NULL)
123     return 0;

125     /* Ako smo stigli do trazenog nivoa, vracamo 1 - to ce kasnije
        zbog rekurzivnih poziva uvecati broj pojavljivanja za 1 */
127     if (i == 0)
        return 1;

129     /* inace, spustamo se jedan nivo nize i u levom i u desnom
        postablu */
131

```

```
133     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
134         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
135 }
136
137 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
138 void ispis_nivo(Cvor * koren, int i)
139 {
140     /* ideja je slicna ideji iz prethodne funkcije */
141
142     /* nema vise cvorova, ne mozemo da se spustamo kroz stablo */
143     if (koren == NULL)
144         return;
145
146     /* ako smo na trazenom nivou - ispisujemo vrednost */
147     if (i == 0) {
148         printf("%d ", koren->broj);
149         return;
150     }
151     /* inace, spustamo se jedan nivo nize i u levom i u desnom
152     podstablu */
153     ispis_nivo(koren->levo, i - 1);
154     ispis_nivo(koren->desno, i - 1);
155 }
156
157 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom
158     nivou stabla */
159 Cvor *max_nivo(Cvor * koren, int i)
160 {
161     /* Ako je stablo prazno, obustavljamo pretragu */
162     if (koren == NULL)
163         return NULL;
164
165     /* Ako smo na trazenom nivou, takodje prekidamo pretragu */
166     if (i == 0)
167         return koren;
168
169     /* Pronalazimo maksimum sa i-tog nivoa levog podstabla */
170     Cvor *a = max_nivo(koren->levo, i - 1);
171
172     /* Pronalazimo maksimum sa i-tog nivoa desnog podstabla */
173     Cvor *b = max_nivo(koren->desno, i - 1);
174
175     /* Trazimo i vracamo maksimum izracunatih vrednosti */
176     if (a == NULL && b == NULL)
177         return NULL;
178     if (a == NULL)
179         return b;
180     if (b == NULL)
181         return a;
182     return a->broj > b->broj ? a : b;
183 }
```

```
185 }
186
187 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
188 int zbir_nivo(Cvor * koren, int i)
189 {
190     /* Ako je stablo prazno, zbir je nula */
191     if (koren == NULL)
192         return 0;
193
194     /* Ako smo na trazenom nivou, vracamo vrednost */
195     if (i == 0)
196         return koren->broj;
197
198     /* Inace, spustamo se jedan nivo nize i trazimo sume iz levog
199        i desnog podstabla */
200     return zbir_nivo(koren->levo, i - 1) + zbir_nivo(koren->desno,
201                                                       i - 1);
202 }
203
204 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje
205     su manje ili jednake od date vrednosti x */
206 int suma(Cvor * koren, int x)
207 {
208     /* Ako je stablo prazno, zbir je nula */
209     if (koren == NULL)
210         return 0;
211
212     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
213        prirode pretrazivackog stabla treba obici i levo i desno
214        podstablo */
215     if (koren->broj < x)
216         return koren->broj + suma(koren->levo,
217                                   x) + suma(koren->desno, x);
218
219     /* Inace, racunamo samo sumu vrednosti iz levog podstabla jer
220        medju njima jedino moze biti onih koje zadovoljavaju uslov */
221     return suma(koren->levo, x);
222 }
223
224 int main(int argc, char **argv)
225 {
226     /* Analiziramo argumente komandne linije */
227     if (argc != 3) {
228         fprintf(stderr, "Greska! Program se poziva sa: ./a.out nivo
229 broj_zapretragu\n");
230         exit(EXIT_FAILURE);
231     }
232 }
```

```
237     int i = atoi(argv[1]);
238     int x = atoi(argv[2]);
239
240     /* Kreiramo stablo */
241     Cvor *koren = NULL;
242     int broj;
243     while (scanf("%d", &broj) != EOF)
244         dodaj_u_stablo(&koren, broj);
245
246     /* ispisujemo rezultat rada funkcija */
247     printf("broj cvorova: %d\n", br_cvorova(koren));
248     printf("broj listova: %d\n", br_listova(koren));
249     printf("pozitivni listovi: ");
250     pozitivni_listovi(koren);
251     printf("zbir cvorova: %d\n", suma_cvorova(koren));
252
253     if (najveci_element(koren) == NULL)
254         printf("najveci element: ne postoji\n");
255     else
256         printf("najveci element: %d\n",
257             najveci_element(koren)->broj);
258
259     printf("dubina stabla: %d\n", dubina_stabla(koren));
260     printf("\n");
261     printf("broj cvorova na %d. nivou: %d\n", i,
262         cvorovi_nivo(koren, i));
263     printf("elementi na %d. nivou: ", i);
264     ispis_nivo(koren, i);
265     printf("\n");
266     if (max_nivo(koren, i) == NULL)
267         printf("Nema elemenata na %d. nivou!\n", i);
268     else
269         printf("maksimalni na %d. nivou: %d\n", i,
270             max_nivo(koren, i)->broj);
271
272     printf("zbir na %d. nivou: %d\n", i, zbir_nivo(koren, i));
273     printf("zbir elemenata manjih ili jednakih od %d: %d\n", x,
274         suma(koren, x));
275
276     /* Oslobadjamo memoriju zauzetu stablom */
277     oslobodi_stablo(&koren);
278
279     /* Prekidamo izvršavanje programa */
280     return 0;
281 }
```

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 izračunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
9  {
11     /* Dubina praznog stabla je 0 */
       if (koren == NULL)
13         return 0;
15
       /* Izračunavamo dubinu levog podstabla */
       int dubina_levo = dubina_stabla(koren->levo);
17
       /* Izračunavamo dubinu desnog podstabla */
       int dubina_desno = dubina_stabla(koren->desno);
19
       /* Dubina stabla odgovara većoj od dubina podstabala - 1
          dodajemo jer računamo i koren */
21       return dubina_levo >
           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
23   }
25
   /* Funkcija koja ispisuje sve elemente na i-tom nivou */
   void ispisi_nivo(Cvor * koren, int i)
27   {
29       /* Ideja je slična ideji iz prethodne funkcije */
31
       /* Nema više cvorova, ne možemo da se spustamo kroz stablo */
33       if (koren == NULL)
           return;
35
       /* Ako smo na traženom nivou - ispisujemo vrednost */
37       if (i == 0) {
           printf("%d ", koren->broj);
39           return;
       }
41       /* Inače, spustamo se jedan nivo niže i u levom i u desnom
          podstablu */
       ispisi_nivo(koren->levo, i - 1);
       ispisi_nivo(koren->desno, i - 1);
43   }
45
   /* Funkcija koja ispisuje stablo po nivoima */
   void ispisi_stablo_po_nivoima(Cvor * koren)
47   {
49
51       int i;
```

```
53  /* Prvo izracunavamo dubinu stabla */
    int dubina;
55  dubina = dubina_stabla(koren);

57  /* Ispisujemo nivo po nivo stabla */
    for (i = 0; i < dubina; i++) {
59      printf("%d. nivo: ", i);
        ispisi_nivo(koren, i);
61      printf("\n");
    }
63 }

65 int main(int argc, char **argv)
{
67     Cvor *koren;
    int broj;
69
    /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
71     koren = NULL;
    while (scanf("%d", &broj) != EOF) {
73         dodaj_u_stablo(&koren, broj);
    }
75
    /* Ispisujemo stablo po nivoima */
77     ispisi_stablo_po_nivoima(koren);

79     /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);
81
    /* Prekidamo izvorsavanje programa */
83     return 0;
}
```

### Rešenje 4.26

### Rešenje 4.27

```
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  {

11     /* Dubina praznog stabla je 0 */
        if (koren == NULL)
```

```

13     return 0;

15     /* Izracunavamo dubinu levog podstabla */
16     int dubina_levo = dubina_stabla(koren->levo);

17     /* Izracunavamo dubinu desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);

21     /* Dubina stabla odgovara vecoj od dubina podstabala - 1
22        dodajemo jer racunamo i koren */
23     return dubina_levo >
24            dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
25 }

27 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za
28    AVL stablo */
29 int avl(Cvor * koren)
30 {
31     int dubina_levo, dubina_desno;

33     /* Ako je stablo prazno, zaustavljamo brojanje */
34     if (koren == NULL) {
35         return 0;
36     }

37     /* Izracunavamo dubinu levog podstabla korena */
38     dubina_levo = dubina_stabla(koren->levo);

39     /* Izracunavamo dubinu desnog podstabla korena */
40     dubina_desno = dubina_stabla(koren->desno);

43     /* Ako je uslov za AVL stablo ispunjen */
44     if (abs(dubina_desno - dubina_levo) <= 1) {
45         /* Racunamo broj avl cvorova u levom i desnom podstablu i
46            uvecavamo za jedan iz razloga sto koren ispunjava uslov */
47         return 1 + avl(koren->levo) + avl(koren->desno);
48     } else {
49         /* Inace, racunamo samo broj avl cvorova u podstablama */
50         return avl(koren->levo) + avl(koren->desno);
51     }
52 }

55 int main(int argc, char **argv)
56 {
57     Cvor *koren;
58     int broj;

59     /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
60     koren = NULL;
61     while (scanf("%d", &broj) != EOF) {
62         dodaj_u_stablo(&koren, broj);
63     }

```



```
65  /* Racunamo i ispisujemo broj AVL cvorova */
67  printf("%d\n", avl(koren));

69  /* Oslobadjamo memoriju zauzetu stablom */
   oslobodi_stablo(&koren);

71

   /* Prekidamo izvršavanje programa */
73  return 0;
   }
```

### Rešenje 4.28

```
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. */
21 if (koren == NULL) {
   /* posto je 0 najmanji pozitivan broj, može nam poslužiti
23 kao indikator */
   return 0;
25 }

27 /* Ukoliko je stablo list ... */
   if (koren->levo == NULL && koren->desno == NULL) {
29     /* ... vraćamo njegovu vrednost */
     return koren->broj;
31 }

33 /* Proveravamo svojstvo za levo podstablo. */
   max_levo = heap(koren->levo);
35

   /* Proveravamo svojstvo za desno podstablo. */
37 max_desno = heap(koren->desno);
   /* Ako levo ili desno podstablo uocenog cvora nije heap, onda
```

```
39     nije ni celo stablo. */
40     if (max_levo == -1 || max_desno == -1) {
41         return -1;
42     }
43
44     /* U suprotnom proveravamo da li svojstvo vazi za uoceni
45        cvor. */
46     if (koren->broj > max_levo && koren->broj > max_desno) {
47         /* ako vazi, vracamo vrednost korena */
48         return koren->broj;
49     }
50
51     /* u suprotnom zakljucujemo da stablo nije heap */
52     return -1;
53 }
54
55 int main(int argc, char **argv)
56 {
57     Cvor *koren;
58     int heap_indikator;
59
60     /* 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
72     /* pozivamo funkciju kojom proveravamo da li je stablo heap */
73     heap_indikator = heap(koren);
74
75     /* i ispisujemo rezultat */
76     if (heap_indikator == -1) {
77         printf("Zadato tablo nije heap\n");
78     } else {
79         printf("Zadato stablo je heap!\n");
80     }
81
82     /* Oslobadjamo memoriju zauzetu stablom. */
83     oslobodi_stablo(&koren);
84
85     /* Završavamo sa programom */
86     return 0;
87 }
```



## Glava 5

# Razno

### 5.1 Zadaci

### 5.2 Rešenja



## Glava 6

# Ispitni rokovi

### 6.1 Programiranje 2, praktični deo ispita, jun 2015.

#### Zadatak 6.1

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*

```
ž
Sadraj datoteke:
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
Izlaz:
Ispit
Matematika
Programiranje
```

#### *Test 2*

```
ž
Sadraj datoteke:
2
maksimalano
poena
Izlaz:
```

#### *Test 3*

```
Problem:
datoteka
ne postoji
Izlaz:
-1
```

[Rešenje 6.1]

#### Zadatak 6.2

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 6.2]

**Zadatak 6.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

*Test 1*

```
Ulaz:
4
5
1 2 3 4 5
-1 2 -3 4 -5
-5 -4 -3 -2 1
-1 0 0 0 0
Izlaz:
0
```

*Test 2*

```
Ulaz:
2
3
0 0 -5
1 2 -4
Izlaz:
```

*Test 3*

```
Ulaz:
-2
Izlaz (na stderr):
-1
```

[Rešenje 6.3]

## 6.2 Programiranje 2, praktični deo ispita, jul 2015.

**Zadatak 6.4**

Napisati program koji kao prvi arugment komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja)

koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera. Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

Test 1

```
|| Poziv: ./a.out fajl.txt test
|| Datoteka: Ovo je test primer.
||           U njemu se rec test javlja
||           vise puta. testtesttest
|| Izlaz: 5
```

Test 2

```
|| Poziv: ./a.out
|| Izlaz (na stderr): -1
```

Test 3

```
|| Poziv: ./a.out fajl.txt foo
|| Datoteka: (ne postoji)
|| Izlaz (na stderr): -1
```

Test 4

```
|| Poziv: ./a.out fajl.txt .
|| Datoteka: (prazna)
|| Izlaz: 0
```

[Rešenje 6.4]

### Zadatak 6.5

Na početku datoteke "trouglovi.txt" nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

Test 1

```
|| 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
```

Test 2

```
|| Datoteka: 3
||           1.2 3.2 1.1 4.3
|| Izlaz:    -1
```

Test 3

```
|| Datoteka: (nema datoteke)
|| Izlaz:    -1
```

Test 4

```
|| Datoteka: 0
|| Izlaz:
```



[Rešenje 6.5]

**Zadatak 6.6** 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.

*Test 1*

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

*Test 2*

```
|| Ulaz:
|| 2 5 3 6 1 0 4 7 9
|| Izlaz:
|| 2
```

*Test 3*

```
|| Ulaz:
|| 0 4 2 5
|| Izlaz:
|| 0
```

*Test 4*

```
|| Ulaz:
|| 3
|| Izlaz:
|| 0
```

*Test 5*

```
|| Ulaz:
|| -1 4 5 1 7
|| Izlaz:
|| 0
```

[Rešenje 6.6]

## 6.3 Rešenja

### Rešenje 6.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define MAX 50
5
6 void greska(){
7     printf("-1\n");
8     exit(EXIT_FAILURE);
9 }
10
11 int main(int argc, char* argv[]){
12
13     FILE* ulaz;
14     char** linije;
15     int i, j, n;
16
17     /* Proveravamo argumente komandne linije.
18     */
```

```
19     if(argc!=2){
20         greska();
21     }
22
23     /* Otvaramo datoteku cije ime je navedeno kao argument komandne
24     linije neposredno nakon imena programa koji se poziva. */
25     ulaz=fopen(argv[1], "r");
26     if(ulaz==NULL){
27         greska();
28     }
29
30     /* Ucitavamo broj linija. */
31     fscanf(ulaz, "%d", &n);
32
33     /* Alociramo memoriju na osnovu ucitanog broja linija.*/
34     linije=(char**)malloc(n*sizeof(char*));
35     if(linije==NULL){
36         greska();
37     }
38     for(i=0; i<n; i++){
39         linije[i]=malloc(MAX*sizeof(char));
40         if(linije[i]==NULL){
41             for(j=0; j<i; j++){
42                 free(linije[j]);
43             }
44             free(linije);
45             greska();
46         }
47     }
48
49     /* Ucitavamo svih n linija iz datoteke. */
50     for(i=0; i<n; i++){
51         fscanf(ulaz, "%s", linije[i]);
52     }
53
54     /* Ispisujemo u odgovarajucem poretku ucitane linije koje
55     zadovoljavaju kriterijum. */
56     for(i=n-1; i>=0; i--){
57         if(isupper(linije[i][0])){
58             printf("%s\n", linije[i]);
59         }
60     }
61
62     /* Oslobadjamo memoriju koju smo dinamicki alocirali. */
63     for(i=0; i<n; i++){
64         free(linije[i]);
65     }
66
67     free(linije);
68
69     /* Zatvaramo datoteku. */
70     fclose(ulaz);
```

```
69
    /* Završavamo sa programom. */
71    return 0;
73 }
```

### Rešenje 6.2

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

4
int sumirajN (Cvor * koren, int n){
6     if(koren==NULL){
            return 0;
8     }

10     if(n==0){
            return koren->broj;
12     }

14     return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
}

16

18 int main(){
    Cvor* koren=NULL;
20     int n;
    int nivo;

22     /* Citamo vrednost nivoa */
24     scanf("%d", &nivo);

26

28     while(1){

        /* Citamo broj sa standardnog ulaza */
30         scanf("%d", &n);

        /* Ukoliko je korisnik uneo 0, prekidamo dalje citanje. */
32         if(n==0){
            break;
34         }

        /* A ako nije, dodajemo procitani broj u stablo. */
36         dodaj_u_stablo(&koren, n);
38

40     }

42     /* Ispisujemo rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren,nivo));
```

```

44     /* Oslobadjamo memoriju */
45     oslobodi_stablo(&koren);
46
47     /* Prekidamo izvršavanje programa */
48     return 0;
49 }

```

---

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"
4
5  Cvor* napravi_cvor(int b ) {
6      Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
7      if( novi == NULL)
8          return NULL;
9
10     /* Inicijalizacija polja novog Cvora */
11     novi->broj = b;
12     novi->levo = NULL;
13     novi->desno = NULL;
14
15     return novi;
16 }
17
18 void oslobodi_stablo(Cvor** adresa_korena) {
19     /* Prazno stablo i nema sta da se oslobadja */
20     if( *adresa_korena == NULL)
21         return;
22
23     /* Rekurzivno oslobadjamo najpre levo, a onda i desno podstablo*/
24     if( (*adresa_korena)->levo )
25         oslobodi_stablo(&(*adresa_korena)->levo);
26     if( (*adresa_korena)->desno)
27         oslobodi_stablo(&(*adresa_korena)->desno);
28
29     free(*adresa_korena);
30     *adresa_korena = NULL;
31 }
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*/

```

```

45     if( *adresa_korena == NULL){
        Cvor* novi = napravi_cvor(broj);
        prover_i_alokaciju(novi);
47     *adresa_korena = novi; /* Kreirani Cvor novi ce biti od
        sada koren stabla*/
        return;
49     }

51     /* Brojeve smestamo u uredjeno binarno stablo, pa
        ako je broj koji ubacujemo manji od broja koji je u korenu */
53     if( broj < (*adresa_korena)->broj)
        /* Dodajemo u levo podstablo */
55         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
        /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
        dodajemo nov Cvor desno od korena */
57     else
        dodaj_u_stablo(&(*adresa_korena)->desno, broj);
59 }

```

```

1 #ifndef __STABLA_H__
2 #define __STABLA_H__ 1

4 /* Struktura kojom se predstavlja Cvor drveteta */
typedef struct dcvor{
6     int broj;
    struct dcvor* levo, *desno;
8 } Cvor;

10 /* Funkcija alokira prostor za novi Cvor drveteta, inicijalizuje polja
    strukture i vraca pokazivac na nov Cvor */
12 Cvor* napravi_cvor(int b );

14 /* Oslobadjamo dinamički alocirani prostor za stablo
    * Nakon oslobadjanja se u pozivajucoj funkciji koren
    * postavlja NULL, jer je stablo prazno */
16 void oslobodi_stablo(Cvor** adresa_korena);
18

20 /* Funkcija proverava da li je novi Cvor ispravno alocirani,
    * 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 pozivajucoj funkciji.
    */
28 void dodaj_u_stablo(Cvor** adresa_korena, int broj);

30 #endif

```

## Rešenje 6.3

```
1  #include <stdio.h>
2  #define MAX 50
3
4
5
6  int main(){
7      int m[MAX][MAX];
8      int v, k;
9      int i, j;
10     int max_broj_negativnih, max_indeks_kolone;
11     int broj_negativnih;
12
13     /* Ucitavamo dimenzije matrice */
14     scanf("%d", &v);
15     scanf("%d", &k);
16
17     if(v<0 || v>MAX || k<0 || k>MAX){
18         fprintf(stderr, "-1\n");
19         return 0;
20     }
21
22     /* Ucitavamo elemente matrice */
23     for(i=0; i<v; i++){
24         for(j=0; j<k; j++){
25             scanf("%d", &m[i][j]);
26         }
27     }
28
29     /*Pronalazimo kolonu koja sadrzi najveći broj negativnih
30     elemenata */
31     max_indeks_kolone=0;
32
33     max_broj_negativnih=0;
34     for(i=0; i<v; i++){
35         if(m[i][0]<0){
36             max_broj_negativnih++;
37         }
38     }
39
40     for(j=0; j<k; j++){
41         broj_negativnih=0;
42         for(i=0; i<v; i++){
43             if(m[i][j]<0){
44                 broj_negativnih++;
45             }
46             if(broj_negativnih>max_broj_negativnih){
47                 max_indeks_kolone=j;
48             }
49         }
50     }
```

```
50     }

52     /* Ispisujemo trazeni rezultat */
    printf("%d\n", max_indeks_kolone);

54     /* Završavamo program */
56     return 0;
}
```

### Rešenje 6.4

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #define MAX 128
5
   int main(int argc, char **argv) {
7     FILE *f;
       int brojac = 0;
9     char linija[MAX], *p;

11    if (argc != 3) {
        fprintf(stderr, "-1\n");
13        exit(EXIT_FAILURE);
    }

15    if ((f = fopen(argv[1], "r")) == NULL) {
17        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
19    }

21    while (fgets(linija, MAX, f) != NULL) {
        p = linija;
23        while (1) {
            p = strstr(p, argv[2]);
25            if (p == NULL)
                break;
            brojac++;
            p = p + strlen(argv[2]);
29        }
    }

31    fclose(f);

33    printf("%d\n", brojac);

35    return 0;
37 }
```

## Rešenje 6.5

```

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
21 int poredi(const void *a, const void *b) {
22     trougao x = *(trougao*)a;
23     trougao y = *(trougao*)b;
24     double xp = povrsina(x);
25     double yp = povrsina(y);
26     if (xp < yp)
27         return 1;
28     if (xp > yp)
29         return -1;
30     return 0;
31 }
32
33 int main() {
34     FILE *f;
35     int n, i;
36     trougao *niz;
37
38     if ((f = fopen("truglovi.txt", "r")) == NULL) {
39         fprintf(stderr, "-1\n");
40         exit(EXIT_FAILURE);
41     }
42
43     if (fscanf(f, "%d", &n) != 1) {
44         fprintf(stderr, "-1\n");
45         exit(EXIT_FAILURE);
46     }
47
48     if ((niz = malloc(n * sizeof(trougao))) == NULL) {
49         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;
}

```

```

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

5  Cvor *napravi_cvor(int broj)
   {
7
   /* Dinamicki kreiramo cvor */
9       Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

11
   /* U slucaju greske ... */
       if (novi == NULL) {
13         fprintf(stderr, "-1\n");
         exit(1);
15     }

17
   /* Inicijalizacija */
       novi->vrednost = broj;
19       novi->levi = NULL;
       novi->desni = NULL;

21
   /* Vracamo adresu novog cvora */
23       return novi;
   }

25
void dodaj_u_stablo(Cvor **koren, int broj)

```

```

27 {
29 /* Izlaz iz rekurzije: ako je stablo bilo prazno,
   novi koren je upravo novi cvor */
31 if (*koren == NULL) {
   *koren = napravi_cvor(broj);
33 return;
   }
35
37 /* Ako je stablo neprazno, i koren sadrzi manju vrednost
   od datog broja, broj se umece u desno podstablo,
   rekurzivnim pozivom */
39 if ((*koren)->vrednost < broj)
   dodaj_u_stablo(&(*koren)->desni, broj);
41 /* Ako je stablo neprazno, i koren sadrzi vecu vrednost
   od datog broja, broj se umece u levo podstablo,
   rekurzivnim pozivom */
43 else if ((*koren)->vrednost > broj)
45     dodaj_u_stablo(&(*koren)->levi, broj);
47 }

49 void prikazi_stablo(Cvor * koren)
50 {
51 /* Izlaz iz rekurzije */
   if (koren == NULL)
53     return;

55     prikazi_stablo(koren->levi);
   printf("%d ", koren->vrednost);
57     prikazi_stablo(koren->desni);
   }

59 Cvor* ucitaj_stablo() {
61     Cvor *koren = NULL;
   int x;
63     while (scanf("%d", &x) == 1)
   dodaj_u_stablo(&koren, x);
65     return koren;
   }

67 void oslobodi_stablo(Cvor **koren)
68 {
71 /* Izlaz iz rekurzije */
   if (*koren == NULL)
73     return;

75     oslobodi_stablo(&(*koren)->levi);
   oslobodi_stablo(&(*koren)->desni);
77     free(*koren);

```

```

79     *koren = NULL;
    }

1  #ifndef __STABLA_H__
    #define __STABLA_H__ 1
3
    /* Struktura koja predstavlja cvor stabla */
5  typedef struct cvor {
        int vrednost;    /* Vrednost koja se cuva */
7      struct cvor *levi;    /* Pokazivac na levo podstablo */
        struct cvor *desni; /* Pokazivac na desno podstablo */
9  } Cvor;

11 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira
    dinamicki, funkcijom malloc(). U slucaju greske program
13 se prekida i ispisuje se poruka o gresci. U slucaju
    uspeha inicijalizuje se vrednost datim brojem, a pokazivaci
15 na podstabla se inicijalizuju na NULL. Funkcija vraća
    adresu novokreiranog cvora */
17 Cvor *napravi_cvor(int broj);

19 /* Funkcija dodaje novi cvor u stablo sa datim korenom.
    Ukoliko broj vec postoji u stablu, ne radi nista.
    Cvor se kreira funkcijom napravi_cvor(). */
21 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
    vraća
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 6.6

```

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

4  int f3(Cvor *koren, int n) {
        if (koren == NULL || n < 0)
6          return 0;
        if (n == 0) {

```

```
8      if (koren -> levi == NULL && koren -> desni != NULL)
          return 1;
10     if (koren -> levi != NULL && koren -> desni == NULL)
          return 1;
12     return 0;
    }
14     return f3(koren -> levi, n - 1) + f3(koren -> desni, n - 1);
}
16
17 int main() {
18     Cvor *koren;
19     int n;
20
21     scanf("%d", &n);
22     koren = ucitaj_stablo();
23
24     printf("%d\n", f3(koren, n));
25
26     oslobodi_stablo(&koren);
27
28     return 0;
}
```