

Univerzitet u Beogradu  
Matematički fakultet

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

## Zbirka programa

Beograd, 2015.



# Predgovor

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

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

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

*Autori*

---

# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>3</b>
1.1	Podela koda po datotekama . . . . .	3
1.2	Algoritmi za rad sa bitovima . . . . .	6
1.3	Rekurzija . . . . .	12
1.4	Rešenja . . . . .	19
<b>2</b>	<b>Pokazivači</b>	<b>63</b>
2.1	Pokazivačka aritmetika . . . . .	63
2.2	Višedimenzioni nizovi . . . . .	67
2.3	Dinamička alokacija memorije . . . . .	72
2.4	Pokazivači na funkcije . . . . .	77
2.5	Rešenja . . . . .	79
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>119</b>
3.1	Pretraživanje . . . . .	119
3.2	Sortiranje . . . . .	124
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	133
3.4	Rešenja . . . . .	139
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>213</b>
4.1	Liste . . . . .	213
4.2	Stabla . . . . .	224
4.3	Rešenja . . . . .	233
<b>5</b>	<b>Ispitni rokovi</b>	<b>323</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	323
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	325
5.3	Rešenja . . . . .	327



# Glava 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

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

- (a) Definirati strukturu `KompleksanBroj` koja predstavlja kompleksan broj i sadrži realan i imaginaran deo kompleksnog broja.
- (b) Napisati funkciju `ucitaj_kompleksan_broj` koja učitava kompleksan broj sa standardnog ulaza.
- (c) Napisati funkciju `ispisi_kompleksan_broj` koja ispisuje kompleksan broj na standardni izlaz u odgovarajućem formatu (npr. broj čiji je realan deo 2 a imaginarni -3 ispisati kao  $(2 - 3i)$  na standardni izlaz).
- (d) Napisati funkciju `realan_deo` koja računa vrednosti realnog dela broja.
- (e) Napisati funkciju `imaginaran_deo` koja računa vrednosti imaginarnog dela broja.
- (f) Napisati funkciju `modulo` koja računa modulo kompleksnog broja.
- (g) Napisati funkciju `konjugovan` koja računa konjugovano-kompleksni broj svog argumenta.
- (h) Napisati funkciju `saberi` koja sabira dva kompleksna broja.
- (i) Napisati funkciju `oduzmi` koja oduzima dva kompleksna broja.
- (j) Napisati funkciju `mnozi` koja množi dva kompleksna broja.

## 1 Uvodni zadaci

---

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

Napisati program koji testira prethodno napisane funkcije tako što redom:

- (a) pozivanjem funkcije `ucitaj_kompleksan_broj` omogućava da se kompleksan broj  $z_1$  unosi sa standardnog ulaza,
- (b) ispisuje realni deo, imaginarni deo i moduo kompleksnog broja  $z_1$ ,
- (c) pozivanjem funkcije `ucitaj_kompleksan_broj` omogućava da se kompleksan broj  $z_2$  unosi sa standardnog ulaza,
- (d) ispisuje konjugovano kompleksan broj i argument broja  $z_2$ ,
- (e) ispisuje zbir, razliku i proizvod brojeva  $z_1$  i  $z_2$ .

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite realan i imaginaran deo kompleksnog broja:  1 -3
(1.00 - 3.00 i)
realan_deo: 1
imaginaran_deo: -3.000000
moduo 3.162278
Unesite realan i imaginaran deo kompleksnog broja:  -1 4
(-1.00 + 4.00 i)
Njegov konjugovano kompleksan broj: (-1.00 - 4.00 i)
Argument kompleksnog broja: 1.815775
(1.00 - 3.00 i) + (-1.00 + 4.00 i) = (1.00 i)
(1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)
(1.00 - 3.00 i) * (-1.00 + 4.00 i) = (11.00 + 7.00 i)
```

[Rešenje 1.1]

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

### Primer 1

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

[Rešenje 1.2]

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



- (a) Definirati strukturu `Polinom` koja predstavlja polinom (stepena najviše 20). Struktura sadrži stepen i niz koeficijenata. Redosled navođenja koeficijenata u nizu treba da bude takav da na nultoj poziciji u nizu bude koeficijent uz slobodan član, na prvoj koeficijent uz prvi stepen, itd.
- (b) Napisati funkciju koja ispisuje polinom na standardni izlaz u što lepšem obliku.
- (c) Napisati funkciju koja učitava polinom sa standardnog ulaza.
- (d) Napisati funkciju za izračunavanje vrednosti polinoma u datoj tački koristeći Hornerov algoritam.
- (e) Napisati funkciju koja sabira dva polinoma.
- (f) Napisati funkciju koja množi dva polinoma.

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

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite polinom (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
3 1.2 3.5 2.1 4.2
Unesite tacku u kojoj racunate vrednost polinoma
5
Vrednost polinoma u tacki je 252.20
Unesite polinom (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
2 2.1 0 -3.9
Zbir polinoma je: 1.20x^3+5.60x^2+2.10x+0.30
Proizvod polinoma je: 2.52x^5+7.35x^4-0.27x^3-4.83x^2-8.19x-16.38
Unesite izvod polinoma koji zelite:
2
2. izvod prvog polinoma je: 7.20x+7.00
```

[Rešenje 1.3]

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

- (a) Definirati strukturu za reprezentovanje razlomaka.
- (b) Napisati funkcije za učitavanje i ispis razlomaka.

- Napisati funkcije koje vraćaju brojilac i imenilac.
- Napisati funkciju koja vraća vrednost razlomka kao `double` vrednost.
- Napisati funkciju koja izračunava recipročnu vrednost razlomka.
- Napisati funkciju koja skraćuje dati razlomak.
- Napisati funkcije koje sabiraju, oduzimaju, množe i dele dva razlomka.

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

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite imenilac i brojilac prvog razlomka: 1 2
Unesite imenilac i brojilac drugog razlomka: 3 1
Zbir je 5/6
Razlika je 1/6
Zbir je 5/6
Kolicnik je 3/2
```

## 1.2 Algoritmi za rad sa bitovima

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

### Test 1

```
ULAZ:  
    0x7F  
IZLAZ:  
    0000000000000000000000000000111111
```

### Test 2

```
ULAZ:  
    0x80  
IZLAZ:  
    00000000000000000000000001000000
```

### Test 3

```
ULAZ:
    0x00FF00FF
IZLAZ:
    00000000111111110000000011111111
```

## Test 4

```
ULAZ:  
    0xFFFFFFFF  
IZLAZ:  
    11111111111111111111111111111111111111111111111
```

### Test 5

```
ULAZ:
0xABCDE123
IZLAZ:
10101011110011011110000100100011
```

[Rešenje 1.5]

### Zadatak 1.6

Napisati funkcije `count_bits1` i `count_bits2` koje broje bitove sa vrednošću 1 u binarnom zapisu celog broja  $x$ . Prebrojavanje bitova ostvariti na dva načina:

- (a) formiranjem odgovarajuće maske i njenim pomeranjem
- (b) formiranjem odgovarajuće maske i pomeranjem promenljive  $x$ .

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

### Test 1

```
ULAZ:
0x7F
IZLAZ:
Broj jedinica u zapisu je
funkcija count_bits1: 7
funkcija count_bits2: 7
```

### Test 2

```
ULAZ:
0x80
IZLAZ:
Broj jedinica u zapisu je
funkcija count_bits1: 1
funkcija count_bits2: 1
```

### Test 3

```
ULAZ:
0x00FF00FF
IZLAZ:
Broj jedinica u zapisu je
funkcija count_bits1: 16
funkcija count_bits2: 16
```

### Test 4

```
ULAZ:
0xFFFFFFFF
IZLAZ:
Broj jedinica u zapisu je
funkcija count_bits1: 32
funkcija count_bits2: 32
```

### Test 5

```
ULAZ:
0xABCDE123
IZLAZ:
Broj jedinica u zapisu je
funkcija count_bits1: 17
funkcija count_bits2: 17
```

[Rešenje 1.6]

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

## 1 Uvodni zadaci

---

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 heksadekadsnom formatu.

### Test 1

```
ULAZ:
0x7F
IZLAZ:
Najveci:
11111110000000000000000000000000
Najmanji:
000000000000000000000000111111
```

### Test 2

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

### Test 3

```
ULAZ:
0x00FF00FF
IZLAZ:
Najveci:
11111111111111111000000000000000
Najmanji:
00000000000000001111111111111111
```

### Test 4

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

### Test 5

```
ULAZ:
0xABCD123
IZLAZ:
Najveci:
11111111111111111000000000000000
Najmanji:
00000000000000001111111111111111
```

[Rešenje 1.7]

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

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

- 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.*

[illegible]

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

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

```
ULAZ:
    B10011A7 5
IZLAZ:
    x = 1011000100000000001000110100111
    rotate_left(2969571751, 5) = 00100000000000100011010011110110
    rotate_right(2969571751, 5) = 00111101100010000000000010001101
    rotate_right_signed(2969571751, 5) = 00111101100010000000000010001101
```

[Rešenje 1.9]

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

*Test 1*

```
|| ULAZ:
|| 255
|| IzLAZ:
|| 0000000000000000000000001001010101
|| 10101010010000000000000000000000
```

[Rešenje 1.10]

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

*Test 1*

```
|| ULAZ:
|| 10
|| IzLAZ:
|| 0
```

*Test 2*

```
|| ULAZ:
|| 1024
|| IzLAZ:
|| 0
```

*Test 3*

```
|| ULAZ:
|| 2147377146
|| IzLAZ:
|| 1
```

*Test 4*

```
|| ULAZ:
|| 1111111115
|| IzLAZ:
|| 0
```

[Rešenje 1.11]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ:   11 IZLAZ:   1           </pre>	<pre> ULAZ:   1024 IZLAZ:   0           </pre>	<pre> ULAZ:   2147377146 IZLAZ:   22           </pre>

[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>Primer 1</i>	<i>Primer 2</i>	<i>Primer 2</i>
<pre> POZIV: ./a.out 1 2 INTERAKCIJA PROGRAMA:   11   13           </pre>	<pre> POZIV: ./a.out 1 2 INTERAKCIJA PROGRAMA:   1024   1024           </pre>	<pre> POZIV: ./a.out 12 12 INTERAKCIJA PROGRAMA:   12345   12345           </pre>

**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>
<pre> ULAZ:   11 IZLAZ:   0000000B           </pre>	<pre> ULAZ:   1024 IZLAZ:   00000400           </pre>	<pre> ULAZ:   12345 IZLAZ:   00003039           </pre>

[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.

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

**Zadatak 1.16** ++ Napisati funkciju koja 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.

Test 1	Test 2	Test 3
<pre> ULAZ: 123 IZLAZ: 0 </pre>	<pre> ULAZ: 3245 IZLAZ: 2 </pre>	<pre> ULAZ: 100328 IZLAZ: 1 </pre>

### 1.3 Rekurzija

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

Test 1	Test 2	Test 3
<pre> ULAZ: 2 10 IZLAZ: 1024 </pre>	<pre> ULAZ: 5 3 IZLAZ: 125 </pre>	<pre> ULAZ: 9 4 IZLAZ: 6561 </pre>

[Rešenje 1.17]

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

- fukciju `paran` koja proverava da li je broj cifara nekog broja paran i vraća 1 ako jeste, a 0 inače;
- i fukciju `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 proizvoljan broj  $n$  ( $n \leq 12$ ) unet sa standardnog ulaza.

*Primer 1*

```
|| INTERAKCIJA PROGRAMA:
|| Unesite n (<= 12): 5
|| 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 proizvoljan broj  $n$  ( $n \in \mathbb{N}$ ) unet sa standardnog ulaza.

*Primer 1*

```
|| INTERAKCIJA PROGRAMA:
|| Unesite koeficijente 2 3
|| Unesite koji clan niza se racuna 5
|| F(5) = 61
```

[Rešenje 1.20]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 123 IZLAZ: 6 </pre>	<pre> ULAZ: 23156 IZLAZ: 17 </pre>	<pre> ULAZ: 1432 IZLAZ: 10 </pre>
<i>Test 4</i>	<i>Test 5</i>	
<pre> ULAZ: 1 IZLAZ: 1 </pre>	<pre> ULAZ: 0 IZLAZ: 0 </pre>	

[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.

<i>Test 1</i>	<i>Test 2</i>
<pre> ULAZ: 3 2 1 4 21 IZLAZ: 21 </pre>	<pre> ULAZ: 2 -1 0 -5 -10 IZLAZ: 2 </pre>
<i>Test 3</i>	<i>Test 4</i>
<pre> ULAZ: 1 11 3 5 8 1 IZLAZ: 11 </pre>	<pre> ULAZ: 5 IZLAZ: 5 </pre>

[Rešenje 1.23]

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

*Test 1*

```

|| ULAZ:
|| 3 1 2 3 1 2 3
|| IZLAZ:
|| 14

```

*Test 2*

```

|| ULAZ:
|| 2 3 5 2 6
|| IZLAZ:
|| 36

```

*Test 3*

```

|| ULAZ:
|| 0
|| IZLAZ:
|| 0

```

[Rešenje 1.24]

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

*Test 1*

```

|| ULAZ:
|| 4 1 2 3 4
|| IZLAZ:
|| 1

```

*Test 2*

```

|| ULAZ:
|| 11 3 2 11 14 11 43 1
|| IZLAZ:
|| 2

```

*Test 3*

```

|| ULAZ:
|| 1 3 21 5 6
|| IZLAZ:
|| 0

```

[Rešenje 1.25]

**Zadatak 1.26** Napisati rekurzivnu funkciju **tri\_uzastopna\_clana** kojom se proverava da li su tri zadata broja uzastopni članovi niza. Potom, napisati program koji je testira. Sa standardnog ulaza se unose najpre tri tražena broja,

## 1 Uvodni zadaci

a zatim elementi niza, sve do kraja ulaza. Pretpostaviti da neće biti uneto više od 256 brojeva.

Test 1

ULAZ:									
	1	2	3	4	1	2	3	4	5
IZLAZ:									
	da								

Test 2

ULAZ:	1	2	3	11	1	2	4	3	6
IzLAZ:	ne								

[Rešenje 1.26]

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

Test 1

ULAZ:	0x7F
IZLAZ:	7

Test 2

ULAZ:	0x80
IZLAZ:	1

Test 3

ULAZ:	0x00FF00FF
IZLAZ:	16

Test 4

ULAZ:	0xFFFFFFFF
IZLAZ:	32

[Rešenje 1.27]

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

[illegible]

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

Test 1	Test 2	Test 3
<pre> ULAZ: 5 IZLAZ: 5 </pre>	<pre> ULAZ: 125 IZLAZ: 7 </pre>	<pre> ULAZ: 8 IZLAZ: 1 </pre>

[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	Test 2	Test 3
<pre> ULAZ: 5 IZLAZ: 5 </pre>	<pre> ULAZ: 16 IZLAZ: 1 </pre>	<pre> ULAZ: 18 IZLAZ: 2 </pre>

[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	Test 2	
<div>ULAZ: programiranje</div> <div>IZLAZ: ne</div>	<div>ULAZ: anavolimilovana</div> <div>IZLAZ: da</div>	
Test 3	Test 4	Test 5
<div>ULAZ: a</div> <div>IZLAZ: da</div>	<div>ULAZ: aba</div> <div>IZLAZ: da</div>	<div>ULAZ: aa</div> <div>IZLAZ: da</div>

[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.

### Primer 1

```

INTERAKCIJA PROGRAMA:
Unesite duzinu permutacije: 3
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

```

- 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.
- Napisati rekurzivnu funkciju koja izračunava  $d_n$  kao sumu elemenata  $n$ -te hipotenuze Paskalovog trougla.

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

### Test 1

```

ULAZ:
5 3
IZLAZ:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
8

```

[Rešenje 1.33]

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

*Test 1*

```
ULAZ:
3
IZLAZ:
a a a
a a b
a b a
a b b
b a a
b a b
b b a
b b b
```

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

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

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

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

## 1.4 Rešenja

### Rešenje 1.1

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

```
4  /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
   imaginaran deo kompleksnog broja */
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 + i y)
   Ovoj funkciji se kompleksan broj prenosi po vrednosti (za ispis
   nije neophodna adresa)
   */
20 void ispisi_kompleksan_broj(KompleksanBroj z) {
   printf("(");
22     if( z.real != 0 ) {
       printf("%.2f",z.real);
24         if(z.imag > 0 )
           printf(" +");
26     }

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

34     printf(")");
}

36 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
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;
59 }
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;
69 }
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;
79 }
80 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
   argumenata funkcije */
82 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) {
    KompleksanBroj z;
84     z.real = z1.real * z2.real - z1.imag * z2.imag;
86     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 int main() {
```

```
98      /* Deklaracija 2 promenljive tipa KompleksanBroj */
KompleksanBroj z1, z2;

100     /* Ucitavanje prvog kompleksnog broja u promenljivu z1, a potom
njegovo ispisivanje na standardni izlaz */
102     ucitaj_kompleksan_broj(&z1);
ispisi_kompleksan_broj(z1);

104     /* Ispisuje se na standardni izlaz realan, imaginaran deo i moduo
kompleksnog broja z1 */
106     printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
realan_deo(z1), imaginaran_deo(z1), moduo(z1));
printf("\n");

108     /* Ucitavanje drugog kompleksnog broja u promenljivu z2, a potom
njegovo ispisivanje na standardni izlaz */
110     ucitaj_kompleksan_broj(&z2);
ispisi_kompleksan_broj(z2);

112     /* Racunanje i ispisivanje konjugovano kompleksan broj od z2 */
114     printf("\nNjegov konjugovano kompleksan broj: ");
ispisi_kompleksan_broj( konjugovan(z2) );
printf("\n");

116     /* Sabiranje kompleksnih brojeva */
118     printf("\n");
ispisi_kompleksan_broj(z1);
120     printf(" + ");
ispisi_kompleksan_broj(z2);
122     printf(" = ");
ispisi_kompleksan_broj(saberi(z1, z2));
124     printf("\n");

126     /* Oduzimanje kompleksnih brojeva */
128     printf("\n");
ispisi_kompleksan_broj(z1);
130     printf(" - ");
ispisi_kompleksan_broj(z2);
132     printf(" = ");
ispisi_kompleksan_broj(oduzmi(z1, z2));
printf("\n");

134     /* Mnozenje kompleksnih brojeva */
136     printf("\n");
ispisi_kompleksan_broj(z1);
138     printf(" * ");
ispisi_kompleksan_broj(z2);
140     printf(" = ");
ispisi_kompleksan_broj(mnozi(z1, z2));

142     /* Testiranje funkcije koja racuna argument kompleksnih brojeva
*/
```

```

144     printf("\n");
        ispisi_kompleksan_broj(z2);
146     printf("\nArgument kompleksnog broja %f\n", argument(z2) );

148     return 0;
}

```

## Rešenje 1.2

```

/* Uključuje se zaglavlje neophodno za rad sa kompleksnim brojevima.
2   Ovde je to neophodno jer nam je neophodno da bude poznata definicija
   tipa KompleksanBroj. Takođe, time
   su uključena zaglavlja standardne biblioteke koja su navedena u
   complex.h
4   */
   #include "complex.h"
6
   /* Funkcija učitava sa standardnog ulaza realan i imaginaran deo
   kompleksnog broja i smesta ih u strukturu čija adresa je argument
   funkcije */
8   void učitaj_kompleksan_broj(KompleksanBroj* z) {
        printf("Unesite realan i imaginaran deo kompleksnog broja: ");
10        scanf("%f", &z->real);
        scanf("%f", &z->imag);
12    }

14   /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
   se šalje kao argument u obliku (x + y i)
   */
16   void ispisi_kompleksan_broj(KompleksanBroj z) {
        printf("(");
18        if(z.real != 0) {
            printf("%.2f", z.real);
20
            if(z.imag > 0)
22                printf(" + %.2f i", z.imag);
            else if(z.imag < 0)
24                printf(" - %.2f i", -z.imag);
            }
26        else
            printf("%.2f i", z.imag);

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

32        printf(")");
    }
34
   /* Funkcija vraća vrednosti realnog dela kompleksnog broja */
36   float realan_deo(KompleksanBroj z) {
        return z.real;

```

```
38 }

40 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
float imaginaran_deo(KompleksanBroj z) {
42     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;
    z1.imag *= -1;
54     return z1;
56 }

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

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

64     return z;
66 }

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

70     z.real -= z2.real;
72     z.imag -= z2.imag;

74     return z;
76 }

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

80     z.real = z1.real * z2.real - z1.imag * z2.imag;
82     z.imag = z1.real * z2.imag + z1.imag * z2.real;

84     return z;
```

```

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

```

```

/*
2  Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
    deklaracije funkcija za rad sa kompleksnim brojevima.
    Zaglavlje nikada ne treba da sadrzi definicije funkcija.
4  Da bi neki program mogao da koristi ove brojeve i funkcije iz ove
    biblioteke, neophodno je da ukljuci ovo zaglavlje.
*/
6
/* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i time
    onemogućujemo da se sadržaj zaglavlja više puta ukljuci.
8  Niska posle ključne reci ifndef je proizvoljna, ali treba da se
    ponovi u narednoj pretprocesorskoj define direktivi.
*/
10 #ifndef _COMPLEX_H
    #define _COMPLEX_H
12
/* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
    koje se koriste u definicijama funkcija navedenim u complex.c */
14 #include <stdio.h>
    #include <math.h>
16
/* Struktura KompleksanBroj*/
18 typedef struct {
    float real;
20     float imag;
} KompleksanBroj;
22
/* Deklaracije funkcija za rad sa kompleksnim brojevima.
    Sve one su definisane u complex.c */
24 void ucitaj_kompleksan_broj(KompleksanBroj* z) ;
26 void ispisi_kompleksan_broj(KompleksanBroj z) ;
28 float realan_deo(KompleksanBroj z) ;
30 float imaginaran_deo(KompleksanBroj z);
32 float moduo(KompleksanBroj z);
34 KompleksanBroj konjugovan(KompleksanBroj z) ;
36 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) ;
38 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) ;

```

## 1 Uvodni zadaci

---

```
40 KomplexanBroj mnozi(KomplexanBroj z1, KomplexanBroj z2 ) ;
42 float argument(KomplexanBroj z) ;
44 /* Kraj zakljucanog dela */
46 #endif
```

```
2 /******
Ovaj program koristi korektno definisanu biblioteku kompleksnih
brojeva. U zaglavlju complex.h nalazi se definicija kompleksnog
4 broja i popis deklaracija podrzanih funkcija, a u complex.c se
nalaze njihove definicije.
6
Kompilacija programa se najjednostavnije postize naredbom
8 gcc -Wall -lm -o izvrsni complex.c main.c
10
Kompilacija se moze uraditi i na sledeci nacin:
gcc -Wall -c -o complex.o complex.c
12 gcc -Wall -c -o main.o main.c
gcc -lm -o complex complex.o main.o
14 *****/
16
#include <stdio.h>
18 /* Ukljucuje aw zaglavlje neophodno za rad sa kompleksnim
brojevima */
20 #include "complex.h"
22 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje
njegov polarni oblik */
24 int main() {
KomplexanBroj z;
26
/* Ucitavamo kompleksan broj */
28 ucitaj_komplexan_broj(&z);
30 printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f\n",
moduo(z), argument(z));
32 return 0;
}
```

### Rešenje 1.3

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

```

6  /* Funkcija koja ispisuje polinom na standardan izlaz u citljivom
   obliku.
   Kako bi uštedeli kopiranje cele strukture, polinom prenosimo po
   adresi */
8  void ispisi(const Polinom * p)
   {
10     int i;
12     for (i = p->stepen; i >= 0; i--) {
14         if (p->koef[i]) {
16             if (p->koef[i] >= 0 && i != p->stepen)
18                 putchar('+');
20             if (i > 1)
22                 printf("%.2fx^%d", p->koef[i], i);
24             else if (i == 1)
26                 printf("%.2fx", p->koef[i]);
28             else
30                 printf("%.2f", p->koef[i]);
32             }
34         }
36         putchar('\n');
38     }
40 }

42 /* Funkcija koja ucitava polinom sa tastature */
44 Polinom ucitaj()
46 {
48     int i;
50     Polinom p;

52     /* Ucitavamo stepen polinoma */
54     scanf("%d", &p.stepen);

56     /* Ponavljamo ucitavanje stepena sve dok ne unesemo stepen iz
58     dozvoljenog opsega */
60     while (p.stepen > MAX_STEPEN || p.stepen < 0) {
62         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
64         scanf("%d", &p.stepen);
66     }

68     /* Unosimo koeficijente polinoma */
70     for (i = p.stepen; i >= 0; i--)
72         scanf("%lf", &p.koef[i]);
74     return p;
76 }

78 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
79 algoritmom */
81 /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
82 double izracunaj(const Polinom * p, double x)
84 {
86     double rezultat = 0;
88     int i = p->stepen;
90     for (; i >= 0; i--)

```

```
54     rezultat = rezultat * x + p->koef[i];
55     return rezultat;
56 }

58 /* Funkcija koja sabira dva polinoma */
59 Polinom saberi(const Polinom * p, const Polinom * q)
60 {
61     Polinom rez;
62     int i;

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

66     for (i = 0; i <= rez.stepen; i++)
67         rez.koef[i] =
68             (i > p->stepen ? 0 : p->koef[i]) + (i >
69                 q->stepen ? 0 : q->
70                 koef[i]);
71     return rez;
72 }

74 /* Funkcija mnozi dva polinoma p i q */
75 Polinom pomnozi(const Polinom * p, const Polinom * q)
76 {
77     int i, j;
78     Polinom r;

80     r.stepen = p->stepen + q->stepen;
81     if (r.stepen > MAX_STEPEN) {
82         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
83         exit(EXIT_FAILURE);
84     }

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

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

94     return r;
95 }

96 /* Funkcija racuna izvod polinoma p */
97 Polinom izvod(const Polinom * p)
98 {
99     int i;
100    Polinom r;

102    if (p->stepen > 0) {
104        r.stepen = p->stepen - 1;
```



```

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

```

```

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

```

## 1 Uvodni zadaci

---

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

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

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

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

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

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

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

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

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

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

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

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

```

24      /* Ispis polinoma */
25      ispisi(&p);

26      printf("Unesite tacku u kojoj racunate vrednost polinoma\n");
27      scanf("%lf", &x);

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

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

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

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

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

49      /* Uspesno završavamo program */
50      return 0;
51 }

```

## Rešenje 1.5

```

1  #include <stdio.h>

2
3  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
4     celog broja u memoriji. Bitove koji predstavljaju binarnu
5     reprezentaciju broja treba ispisati sa leva na desno, tj. od
6     bita najveće težine ka bitu najmanje težine. */
7  void print_bits(unsigned x)
8  {

9
10     /* Broj bitova celog broja */
11     unsigned velicina = sizeof(unsigned) * 8;
12     /* Maska koja se koristi za "ocitavanje" bitova */

```

```
14 unsigned maska;

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

28     putchar('\n');
   }

30

32 int main()
   {
34     int broj;
       scanf("%x", &broj);
36     print_bits(broj);

38     return 0;
   }
```

### Rešenje 1.6

```
#include <stdio.h>

2
/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji */
4 void print_bits(int x)
   {
6     unsigned velicina = sizeof(int) * 8;
       unsigned maska;

8
10     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
       putchar(x & maska ? '1' : '0');

12     putchar('\n');
14 }

16 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
   x kreiranjem odgovarajuce maske i njenim pomeranjem*/
18 int count_bits1(int x)
```

```

20 {
21     int br = 0;
22     unsigned wl = sizeof(unsigned) * 8 - 1;
23
24     /* Formiranje se maska cija binarna reprezentacija izgleda
25        100000...00000000, koja služi za očitavanje
26        bita najveće težine. U svakoj iteraciji maska se pomera u
27        desno za 1 mesto, i očitavamo sledeći bit. Petlja se
28        završava kada više nema jedinica tj. kada maska postane
29        nula. */
30     unsigned maska = 1 << wl;
31     for (; maska != 0; maska >>= 1)
32         x & maska ? br++ : 1;
33
34     return br;
35 }
36
37 /* Funkcija vraća broj jedinica u binarnoj reprezentaciji broja
38    x formiranjem odgovarajuće maske i pomeranjem promenljive x
39    */
40 int count_bits2(int x)
41 {
42     int br = 0;
43     unsigned wl = sizeof(int) * 8 - 1;
44
45     /* Kako je argument funkcije označen ceo broj x naredba x>>=1
46        vrsila
47        bi aritmetičko pomeranje u desno, tj. popunjavanje bita najveće
48        težine bitom znaka. U tom slučaju nikad ne bi bio ispunjen
49        uslov x!=0 i program bi bio zarobljen u beskonačnoj petlji.
50        Zbog toga se koristi pomeranje broja x ulevo i maska koja
51        očitava bit najveće težine. */
52
53     unsigned maska = 1 << wl;
54     for (; x != 0; x <<= 1)
55         x & maska ? br++ : 1;
56
57     return br;
58 }
59
60 int main()
61 {
62     int x;
63     scanf("%x", &x);
64     printf("Broj jedinica u zapisu je\n");
65     printf("funkcija count_bits1: %d\n", count_bits1(x));
66     printf("funkcija count_bits2: %d\n", count_bits2(x));
67     return 0;
68 }

```

### Rešenje 1.7

```
1 #include <stdio.h>

3 /* Funkcija vraca najveći neoznačeni broj sastavljen od istih
   bitova koji se nalaze u binarnoj reprezentaciji vrednosti
   promenljive x */
5 unsigned najveći(unsigned x)
7 {
   unsigned velicina = sizeof(unsigned) * 8;

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

13  /* Rezultat se inicijalizuje vrednošću 0 */
   unsigned rezultat = 0;

15
17  /* Promenljiva x se pomera u levo sve dok postoje jedinice
   u njoj binarnoj reprezentaciji (tj. sve dok je promenljiva
   x različita od nule). */
19  for (; x != 0; x <<= 1) {
       /* Za svaku jedinicu koja se korišćenjem maske detektuje na
       poziciji najveće težine u binarnoj reprezentaciji
       promenljive x, potiskuje se jedna nova jedinicu sa
       leva u rezultat */
23     if (x & maska) {
25         rezultat >>= 1;
           rezultat |= maska;
27     }
   }

29   return rezultat;
31 }

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

41
43   /* Promenljiva x se pomera u desno sve dok postoje jedinice
   u njoj binarnoj reprezentaciji (tj. sve dok je promenljiva
   x različita od nule). */
45   for (; x != 0; x >>= 1) {
       /* Za svaku jedinicu koja se korišćenjem vrednosti 1 za masku
       detektuje na
       poziciji najmanje težine u binarnoj reprezentaciji
       promenljive x, potiskuje se jedna nova jedinicu sa
       desna u rezultat */
49     if (x & 1) {
```

```

51     rezultat <<= 1;
      rezultat |= 1;
53 }
    }
55
    return rezultat;
57 }

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

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

69     putchar('\n');
    }

71
73 int main()
{
75     int broj;
      scanf("%x", &broj);

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

79
      printf("Najmanji:\n");
      print_bits(najmanji(broj));

81
83     return 0;
    }

```

### Rešenje 1.8

```

#include <stdio.h>

2

4 /******
      Funkcija postavlja na nulu n bitova pocev od pozicije p.
      Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
      se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
      1010 1110 0111 1010 1011 1100 1101 1110 1000 0010 0111 */
      unsigned reset(unsigned x, unsigned n, unsigned p)
10 {
12 /******
      Cilj je anulirati samo zeljene bitove, a da ostali
      ostanu nepromenjeni. Maska koja ce se koristiti je ona cija
      binarna reprezentacija ima n bitova
14

```

## 1 Uvodni zadaci

```
16     postavljениh na 0 pocev od pozicije p, dok su svi ostali
    postavljени na 1.

18     Na primer, za n=5 i p=10 cilj je maska oblika
    1111 1111 1111 1111 1111 1000 0011 1111
20     To se postize na sledeci nacin:
    ~0                1111 1111 1111 1111 1111 1111 1111 1111
22 (~0 << n)          1111 1111 1111 1111 1111 1111 1110 0000
~(~0 << n)            0000 0000 0000 0000 0000 0000 0001 1111
24 ~(~0 << n) << (p-n+1)) 0000 0000 0000 0000 0000 0111 1100 0000
~(~0 << n) << (p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111
26 *****/
    unsigned maska = ~(~0 << n) << (p - n + 1));
28
    return x & maska;
30 }

32
34 /******
    Funkcija postavlja na 1 n bitova pocev od pozicije p.
    Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
36 se broji od nule . Npr, za n=5, p=10
    1010 1011 1100 1101 1110 1010 1110 0111
38 1010 1011 1100 1101 1110 1111 1110 0111
    *****/
40 unsigned set(unsigned x, unsigned n, unsigned p)
{
42
44     /******
        Cilj je samo odredjenih n bitova postaviti na 1, dok
        ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
46 formira se maska oblika
        0000 0000 0000 0000 0000 0111 1100 0000
48 prateci vrlo slican postupak kao za prethodnu funkciju
    *****/
50     unsigned maska = ~(~0 << n) << (p - n + 1);
52
    return x | maska;
54 }

56 /******
    Funkcija vraća celobrojno polje bitova, desno poravnato, koje
58 predstavlja n bitova pocev od pozicije p u binarnoj
    reprezentaciji broja x, pri cemu se pozicija broji sa desna
    ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
60 i broj cija je binarna reprezentacija:
    1010 1011 1100 1101 1110 1010 1110 0111
62 trazi se
    0000 0000 0000 0000 0000 0000 0000 1011
64 *****/
66 unsigned get_bits(unsigned x, unsigned n, unsigned p)
```



```

68 {
69 /******
70     Kreira se maska kod koje su poslednjih n bitova 1, a
        ostali su 0. Na primer za n=5
72     0000 0000 0000 0000 0000 0000 0001 1111
        *****/
74     unsigned maska = ~(~0 << n);

76     /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
        polje bude uz
        desni kraj. Zatim se maskiraju ostali bitovi, sem zeljenih n i
78     funkcija vraca tako dobijenu vrednost */
        return maska & (x >> (p - n + 1));
80 }

82
84 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
        postavljeni na vrednosti n bitova najnize tezine binarne
        reprezentacije broja y */
86 unsigned set_n_bits(unsigned x, unsigned n, unsigned p,
        unsigned y)
88 {
89 /******
90     Kreira se maska kod koje su poslednjih n bitova 1, a
        ostali su 0. Na primer za n=5
92     0000 0000 0000 0000 0000 0000 0001 1111
        *****/
94     unsigned last_n_1 = ~(~0 << n);
96     /******
97     Kao sto je i u funkciji reset, i ovde se kreira masku koja ima n
        bitova postavljenih na 0 pocevsi od pozicije p, dok su
98     ostali bitovi 1. Na primer za n=5 i p =10
        1111 1111 1111 1111 1111 1000 0011 1111
100    *****/
102     unsigned middle_n_0 = ~(~0 << n) << (p - n + 1);

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

108     /* U promenljivu y_shift_middle se smesta vrednost dobijena od
        binarne reprezentacije vrednosti promenljive y cijih je n bitova
        najnize tezine pomera tako da stoje
        pocev od pozicije p. Ostali bitovi su nule. (y & last_n_1)
        Resetuju se svi bitovi osim najnizih n */
110     unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);

112     return x_reset ^ y_shift_middle;
}

```

```

114  /* Funkcija invertuje bitove u zapisu broja x pocevsi od
116     pozicije p njih n */
118  unsigned invert(unsigned x, unsigned n, unsigned p)
119  {
120      /******
121         Formira se maska sa n jedinica pocev od pozicije p.
122         Na primer za n=5 i p=10
123         0000 0000 0000 0000 0000 0111 1100 0000
124         *****/
125         unsigned maska = ~(~0 << n) << (p - n + 1);

126         /* Operator ekskluzivno ili invertuje sve bitove gde je
127            odgovarajuci bit maske 1. Ostali bitovi ostaju
128            nepromenjeni. */
129         return maska ^ x;
130     }

132     /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
133        celog broja u memoriji */
134     void print_bits(int x)
135     {
136         unsigned velicina = sizeof(int) * 8;
137         unsigned maska;

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

140         putchar('\n');
141     }

142
143
144
145
146
147
148     int main()
149     {
150         unsigned broj, p, n, y;
151         scanf("%u%u%u%u", &broj, &n, &p, &y);
152         printf("Broj %5u %25s= ", broj, "");
153         print_bits(broj);

154
155         printf("reset(%5u,%5u,%5u)%11s = ", broj, n, p, "");
156         print_bits(reset(broj, n, p));

157         printf("set(%5u,%5u,%5u)%13s = ", broj, n, p, "");
158         print_bits(set(broj, n, p));

159         printf("get_bits(%5u,%5u,%5u)%8s = ", broj, n, p, "");
160         print_bits(get_bits(broj, n, p));

161
162         printf("y = %31u = ", y);

```

```

166 print_bits(y);
    printf("set_n_bits(%5u,%5u,%5u,%5u) = ", broj, n, p, y);
168 print_bits(set_n_bits(broj, n, p, y));

170 printf("invert(%5u,%5u,%5u)%10s = ", broj, n, p, "");
    print_bits(invert(broj, n, p));
172
    return 0;
174 }

```

### Rešenje 1.9

```

1  #include <stdio.h>

3  /*****
   Funkcija binarnu reprezentaciju svog argumenta x rotira u
5  levo za n mesta i vraća odgovarajući neoznačen ceo broj čija
   je binarna reprezentacija dobijena nakon rotacije.
7  Na primer za n =5 i x čija je interna reprezentacija
   1010 1011 1100 1101 1110 0001 0010 0011
9  funkcija vraća neoznačen ceo broj čija je binarna
   reprezentacija:
11  0111 1001 1011 1100 0010 0100 0111 0101
   *****/
13 unsigned rotate_left(int x, unsigned n)
   {
15     unsigned first_bit;
       /* Maska koja ima samo najvisi bit postavljen na 1 neophodna
17     da bi pre siftovanja u levo za 1 najvisi bit bio sacuvan.*/
       unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
19     int i;

21     /* n puta se vrši rotaciju za jedan bit u levo. U svakoj
       iteraciji se odredi prvi bit, a potom se pomera binarna
       reprezentacija trenutne vrednosti promenljive
23     x u levo za 1. Nakon toga, potom najnizi bit se postavlja na
       vrednost
       koju je imao prvi bit koji je istisnut siftovanjem */
25     for (i = 0; i < n; i++) {
         first_bit = x & first_bit_mask;
27         x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
       }
29     return x;
   }

31

33 /*****
   Funkcija neoznačen broj x rotira u desno za n.
   Na primer za n=5 i x čija je binarna reprezentacija
35  1010 1011 1100 1101 1110 0001 0010 0011
   funkcija vraća neoznačen ceo broj čija je binarna
37  reprezentacija:

```

```

0001 1101 0101 1110 0110 1111 0000 1001
39  *****/
unsigned rotate_right(unsigned x, unsigned n)
41 {
    unsigned last_bit;
43     int i;

45     /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
        iteraciji se odredjuje bit najmanje tezine broja x, zatm
47     tako odredjeni bit se siftuje u levo tako da najnizi bit
        dode do pozicije najviseg bita. Zatim, nakon siftovanja binarne
        reprezentacije trenutne vrednosti promenljive x za 1 u
49     desno, najvisi bit se postavlja na vrednost vec zapamcenog bita
        koji je bio na poziciji najmanje tezine. */
    for (i = 0; i < n; i++) {
51         last_bit = x & 1;
        x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
53     }

55     return x;
57 }

/* Verzija funkcije koja broj x rotira u desno za n mesta, gde
59 je argument funkcije x oznaceni ceo broj */
int rotate_right_signed(int x, unsigned n)
61 {
    unsigned last_bit;
63     int i;

65     /* U svakoj iteraciji se odredjuje bit najmanje tezine i smesta u
        promenljivu
67     last_bit. Kako je x oznacen ceo broj, tada se prilikom
        siftovanja u desno vrši aritmeticki sift i cuva se znak
69     broja. Dakle, razlikuju se dva slucaja u zavisnosti od
        znaka od x. Nije dovoljno da se ova provera izvrši pre
71     petlje, s obzirom da rotiranjem u desno na mesto najviseg bita
        moze
        doci i 0 i 1, nezavisno od pocetnog znaka broja smestenog u
        promenljivu x. */
73     for (i = 0; i < n; i++) {
        last_bit = x & 1;

75         if (x < 0)
77     /******
        Siftovanjem u desno broja koji je negativan dobija se 1
79     kao bit na najvisoj poziciji. Na primer ako je x
        1010 1011 1100 1101 1110 0001 0010 001b
        (sa b je oznacen ili 1 ili 0 na najnizoj poziciji)
        Onda je sadrzaj promenljive last_bit:
        0000 0000 0000 0000 0000 0000 0000 000b
83     Nakon siftovanja sadrzaja promenljive x za 1 u desno

```

```

85         1101 0101 1110 0110 1111 0000 1001 0001
      Kako bi umesto 1 na najvisoj poziciji u trenutnoj
87         binarnoj reprezentaciji x bilo postavljeno b nije
      dovoljno da se siftuje na najvisu poziciju jer bi se
89         time dobile 0, a u ovom slucaju su potrebne jedinice
      zbog bitovskog & zato se prvo vrsi komplementiranje, a
91         zatim siftovanje
      ~last_bit << (sizeof(int)*8 -1)
93         B000 0000 0000 0000 0000 0000 0000 0000
      gde B oznacava ~b.
95         Potom se ponovo vrsi komplementiranje kako bi se b
      nalazilo na najvisoj poziciji i sve jedinice na ostalim
97         pozicijama
      ~(~last_bit << (sizeof(int)*8 -1))
99         b111 1111 1111 1111 1111 1111 1111 1111
      *****/
101         x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
      else
103         x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
      }

105     return x;
107 }

109 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
111     celog broja u memoriji */
void print_bits(int x)
113 {
    unsigned velicina = sizeof(int) * 8;
115     unsigned maska;
    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
117         putchar(x & maska ? '1' : '0');

119     putchar('\n');
121 }

123 int main()
{
    unsigned x, k;
125     scanf("%x%x", &x, &k);
    printf("x %36s = ", "");
127     print_bits(x);
    printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
129     print_bits(rotate_left(x, k));

131     printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
    print_bits(rotate_right(x, k));
133

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

```

## 1 Uvodni zadaci

---

```
137 | return 0;
    | }
```

### Rešenje 1.10

```
1 | #include <stdio.h>
3 | /*****
5 | Funkcija vraća vrednost čija je binarna reprezentacija slika
7 | u ogledalu binarne reprezentacije broja x koji se prosledjuje
9 | kao argument funkcije. Na primer za x
11 | čija binarna reprezentacija izgleda ovako
13 | 101010111100110111100100100100011
15 | funkcija treba da vrati broj čija binarna reprezentacija
17 | izgleda:
19 | 11000100100001111011001111010101
21 | *****/
23 | unsigned mirror(unsigned x)
25 | {
27 |     unsigned najnizi_bit;
29 |     unsigned rezultat = 0;
31 |
33 |     int i;
35 |     /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
37 |        vrednosti broja x se određuje i pamti u promenljivoj najnizi_bit
39 |        , nakon čega se na promenljivu x primeni siftovanje u desno.*/
41 |     for (i = 0; i < sizeof(x) * 8; i++) {
43 |         najnizi_bit = x & 1;
45 |         x >>= 1;
47 |         /* Potiskivanjem trenutnog rezultata ka levom kraju svi
49 |            prethodno postavljeni bitovi dobijaju veću poziciju. Novi
51 |            bit se postavlja na najnižu poziciju */
53 |         rezultat <<= 1;
55 |         rezultat |= najnizi_bit;
57 |     }
59 |     return rezultat;
61 | }
63 |
65 | /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
67 |    celog broja u memoriji */
69 | void print_bits(int x)
71 | {
73 |     unsigned velicina = sizeof(int) * 8;
75 |     unsigned maska;
77 |     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
79 |         putchar(x & maska ? '1' : '0');
81 |
83 |     putchar('\n');
```

```
45 int main()
46 {
47     int broj;
48     scanf("%x", &broj);
49
50     /* Ispisuje se binarna reprezentaciju unetog broja */
51     print_bits(broj);
52
53     /* Ispisuje se binarna reprezentaciju broja dobijenog pozivom
54        funkcije mirror */
55     print_bits(mirror(broj));
56
57     return 0;
58 }
```

### Rešenje 1.11

```
1  #include <stdio.h>
2
3  /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n
4     broj jedinica veci od broja nula. U suprotnom funkcija vraca
5     0 */
6  int Broj01(unsigned int n)
7  {
8
9     int broj_nula, broj_jedinica;
10    unsigned int maska;
11
12    broj_nula = 0;
13    broj_jedinica = 0;
14
15    /* Maska je inicijalizovana tako da moze da analizira bit
16       najvece tezine */
17    maska = 1 << (sizeof(unsigned int) * 4 - 1);
18
19    /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
20       iteraciji pomera u desno pa ce jedini bit koji je postavljen na 1
21       biti na svim pozicijama u binarnoj reprezentaciji maske */
22    while (maska != 0) {
23
24        /* Provera da li se na poziciji koju odredjuje maska
25           nalazi 0 ili 1 i uveca se odgovarajuci brojac */
26        if (n & maska) {
27            broj_jedinica++;
28        } else {
29            broj_nula++;
30        }
31
32        /* Pomera se maska u desnu stranu */
33        maska = maska >> 1;
34    }
```

## 1 Uvodni zadaci

---

```
34  /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
    suprotnom vraca 0 */
36  return (broj_jedinica > broj_nula) ? 1 : 0;
38  }
40  int main()
41  {
42      unsigned int n;
44      scanf("%u", &n);
46      printf("%d\n", Broj01(n));
48      return 0;
49  }
```

### Rešenje 1.12

```
#include <stdio.h>
2
int broj_parova(unsigned int x)
4 {
6     int broj_parova;
    unsigned int maska;
8
    /* Vrednost promenljive koja predstavlja broj parova se
       inicijalizuje na 0 */
10    broj_parova = 0;
12
    /* Postavlja se maska tako da moze da procitamo da li su dva
       najmanja bita u zapisu broja x 11 */
14    /* Binarna reprezentacija broja 3 je 000...00011 */
    maska = 3;
16
    while (x != 0) {
18
        /* Provera da li se na najmanjim pozicijama broj x
           nalazi 11 par */
20        if ((x & maska) == maska) {
22            broj_parova++;
        }
24
        /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
           proveravao sledeci par bitova. Pomeranjem u desno
           bit najvece tezine se popunjava nulom jer je x
           neoznaceni broj. */
26        x = x >> 1;
28    }
}
```



```

30     return broj_parova;
32 }
34
35 int main()
36 {
37     unsigned int x;
38
39     scanf("%u", &x);
40
41     printf("%d\n", broj_parova(x));
42
43     return 0;
44 }

```

### Rešenje 1.13

### Rešenje 1.14

```

1  #include <stdio.h>
2
3  /*
4   * Niska koja se formiramo je duzine (sizeof(unsigned int)*8)/4 +1
5   * jer su za svaku heksadekadnu cifru potrebne 4 binarne cifre i
6   * jedna dodatna pozicija za terminirajucu nulu.
7
8   * Prethodni izraz je identican sa sizeof(unsigned int)*2+1.
9   */
10
11 #define MAX_DUZINA sizeof(unsigned int)*2 +1
12
13 void prevod(unsigned int x, char s[])
14 {
15
16     int i;
17     unsigned int maska;
18     int vrednost;
19
20     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
21      * maska za citanje 4 uzastopne cifre */
22     maska = 15;
23
24     /******
25      * Broj se posmatra od pozicije najmanje tezine ka poziciji
26      * najvece tezine. Na primer za broj
27      * 00000000001101000100001111010101
28      * u prvom koraku se citaju bitovi izdvojeni sa <...>:
29      * 0000000000110100010000111101<0101>

```

## 1 Uvodni zadaci

---

```
30      u drugom koraku:
31      000000000011010001000011<1101>0101
32      u trecem koraku:
33      00000000001101000100<0011>11010101 i tako redom
34
35      Indeks i oznacava poziciju na koju se smesta vrednost.
36
37      */
38      for (i = MAX_DUZINA - 2; i >= 0; i--) {
39          /* Vrednost izdvojene cifre */
40          vrednost = x & maska;
41
42          /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter
43             se dobija dodavanjem ASCII koda '0'. Ako je vrednost iz
44             opsega od 10 do 15 odgovarajuci karakter se dobija tako
45             sto se prvo oduzme 10 (time se dobiju vrednosti od 0 do 5) pa
46             se na tako dobijenu vrednost doda ASCII kod 'A' (time se
47             dobija odgovarajuce slovo 'A', 'B', ...
48             'F') */
49          if (vrednost < 10) {
50              s[i] = vrednost + '0';
51          } else {
52              s[i] = vrednost - 10 + 'A';
53          }
54
55          /* Primenljiva x se pomera za 4 bita u desnu stranu i time ce u
56             narednoj iteraciji biti posmatrane sledece 4 cifre */
57          x = x >> 4;
58      }
59
60      s[MAX_DUZINA - 1] = '\0';
61  }
62
63  int main()
64  {
65
66      unsigned int x;
67      char s[MAX_DUZINA];
68
69      scanf("%u", &x);
70
71      prevod(x, s);
72
73      printf("%s\n", s);
74
75      return 0;
76  }
```

### Rešenje 1.17

---



```
printf("%d\n", stepen(x, k));
54 // printf("\n-----\n");
// printf("%d\n", stepen2(2,10));
56 return 0;
}
```

### Rešenje 1.18

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

4 #define MAX 100

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

8
10 /* Deklaracija funkcije neparan mora da bude navedena jer se ta
   funkcija koristi u telu funkcije paran, tj. koristi se pre
   svoje definicije. Funkcija je mogla biti deklarirana i u telu
   funkcije paran. */
12
14 unsigned neparan(unsigned n);

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

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

36 /* Glavna funkcija za testiranje */
int main()
38 {
    int n;
40     scanf("%d", &n);
    printf("Uneti broj ima %s paran broj cifara\n",
42         (paran(n) == 1 ? "" : "ne"));
    return 0;
}
```

44 }

## Rešenje 1.19

```

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

14 /* U sledece dve funkcije je prikazan postupak oslobadjanja od
   repne rekurzije koja postoji u funkciji faktorijelRepna,
16    koristeći algoritam sa predavanja.

17
18    Najpre, funkcija se transformise tako sto rekurzivni poziv
   zemeni sa naredbama kojima se vrednost argumenta funkcije
21    postavlja na vrednost koja bi se prosledjivala rekurzivnom
   pozivu i navodjenjem goto naredbe za vraćanje na pocetak
23    tela funkcije. */

24 int faktorijelRepna_v1(int n, int result)
   {
25 pocetak:
       if (n == 0)
27         return result;
29
30     result = n * result;
       n = n - 1;
32     goto pocetak;
33 }

34
35 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra
   programerska praksa i prethodna funkcija se koristi samo kao
37    medjukorak. Sledi iterativno resenje bez bezuslovnih skokova: */
38 int faktorijelRepna_v2(int n, int result)
   {
40     while (n != 0) {
41         result = n * result;
42         n = n - 1;
43     }

44     return result;
45 }
46
47 }
```

```
49  /* Prilikom poziva prethodnih funkcija pored prvog argumenta
51     celog broja n, mora da se salje i 1 za vrednost drugog argumenta
    u kome ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde
    radi udobnosti korisnika, jer je sasvim prirodno da za faktorijel
    zahteva
53     samo 1 parametar. Funkcija faktorijel izracunava n!, tako sto
    odgovarajucoj gore navedenoj funkciji koja zaista racuna
55     faktorijel, salje ispravne argumente i vraca rezultat koju
    joj ta funkcija vrati. Za testiranje, zameniti u telu
57     funkcije faktorijel poziv faktorijelRepna sa pozivom
    faktorijelRepna_v1, a zatim sa pozivom funkcije
59     faktorijelRepna_v2. */
int faktorijel(int n)
61 {
    return faktorijelRepna(n, 1);
63 }

65 /* Test program */
int main()
67 {
    int n;
69
    printf("Unesite n (<= 12): ");
71     scanf("%d", &n);
    printf("%d! = %d\n", n, faktorijel(n));
73
    return 0;
75 }
```

### Rešenje 1.20

### Rešenje 1.21

```
#include <stdio.h>
2
int zbir_cifara(unsigned int x)
4 {
    /* Izlazak iz rekurzije: ako je broj jednocifren */
    if (x < 10)
        return x;
8
    /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
10     poslednje cifre + poslednja cifra tog broja */
    return zbir_cifara(x / 10) + x % 10;
12 }

14 int main()
{
16     unsigned int x;
```

```

18  /* Ucitava se ceo broj sa ulaza */
    scanf("%u", &x);
20
    /* Ispisuje se zbir cifara ucitanog broja */
22  printf("%d\n", zbir_cifara(x));
24
    return 0;
}

```

### Rešenje 1.22

```

#include <stdio.h>
#define MAX_DIM 1000

/*
   Ako je n==0, onda je suma(a,0) = 0 Ako je n>0, onda je
   suma(a,n) = a[n-1] + suma(a,n-1) Suma celog niza je jednaka
   sumi prvih n-1 elementa uvecenoj za poslednji element celog
   niza. */
int sumaNiza(int *a, int n)
{
    /* Nije postavljena stroga jednakost n==0, za slucaj da korisnik
       prilikom prvog poziva, posalje negativan broj za velicinu
       niza. */
    if (n <= 0)
        return 0;

    return a[n - 1] + sumaNiza(a, n - 1);
}

/*
   Funkcija napisana na drugi nacin:
   n==0, suma(a,0) = 0 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. */
int sumaNiza2(int *a, int n)
{
    if (n <= 0)
        return 0;

    return a[0] + sumaNiza2(a + 1, n - 1);
}

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

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

```

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

44      printf("Suma elemenata je %d\n", sumaNiza(a, n));
46      // printf("Suma elemenata je %d\n", sumaNiza2(a, n));

48      return 0;
}
```

### Rešenje 1.23

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

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

12
14    /* Resavanje problema manje dimenzije */
    int max = maksimum_niza(niz, n - 1);

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

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

26     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
       Promenljiva
       i predstavlja indeks tekuceg broja. */
28     int i = 0;
    while (scanf("%d", &brojevi[i]) != EOF) {
30         i++;
    }

32     n = i;

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



## Rešenje 1.24

```

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

4  int skalarno(int a[], int b[], int n)
5  {
6      /* Izlazak iz rekurzije */
7      if (n == 0)
8          return 0;

10     /* Na osnovu resenja problema dimenzije n-1, resava se problem
        dimenzije n */
12     else
13         return a[n - 1] * b[n - 1] + skalarno(a, b, n - 1);
14 }

16 int main()
17 {
18     int i, a[MAX_DIM], b[MAX_DIM], n;

20     /* Unosi se dimenzija nizova, */
21     scanf("%d", &n);

22     /* A zatim i elementi nizova. */
23     for (i = 0; i < n; i++)
24         scanf("%d", &a[i]);

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

28     /* Ispisuje se rezultat skalarnog proizvoda dva učitana niza. */
29     printf("%d\n", skalarno(a, b, n));

32     return 0;
34 }

```

## Rešenje 1.25

```

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

4  int br_pojave(int x, int a[], int n)
5  {
6      /* Izlazak iz rekurzije */
7      if (n == 1)
8          return a[0] == x ? 1 : 0;

```

## 1 Uvodni zadaci

---

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

20     /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz;
    Promenljiva i predstavlja indeks tekuceg broja */
    i = 0;
22     while (scanf("%d", &a[i]) != EOF) {
        i++;
24     }
    n = i;
26     printf("%d\n", br_pojave(x, a, n));
30     return 0;
}
```

### Rešenje 1.26

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

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

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

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

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

24     int i = 0;
```

```

26 while (scanf("%d", &a[i]) != EOF) {
    i++;
28 }
    n = i;

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

36 return 0;
}

```

### Rešenje 1.27

```

#include <stdio.h>

2
/*****
4 Funkcija koja broji bitove svog argumenta

6 ako je x ==0, onda je count(x) = 0
   inace count(x) = najvisi_bit +count(x<<1)

8
   Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
10 siftovanjem sa desne strane uvek dopisuju 0, argument x ce u
   nekom rekurzivnom pozivu biti bas 0 i izacicemo iz rekurzije.
12 *****/
int count(int x)
14 {
    /* Izlaz iz rekurzije */
16 if (x == 0)
    return 0;

18
    /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja
20 x je postavljen na 1. Koriscenjem odgovarajuce maske proverava
   se vrednost najviseg bita. Rezultat koliko ima jedinica u
   ostatku
22 binarnog zapisa broja x se uvecava za 1. Najvisi bit je 0. Stoga
   je broj jedinica u zapisu x isti
   kao broj jedinica u zapisu broja x<<1, jer se siftovanjem
24 u levo sa desne stane dopisuju 0.
   Za rekurzivni poziv se salje vrednost koja se dobija kada se
26 x siftuje u levo.
   Napomena: argument funkcije x je oznacen ceo broj, usled cega
28 se ne koristi siftovanje udesno, jer funkciji moze biti
   prosleden i negativan broj. Iz tog razloga, odlucujemo se
30 da proveramo najvisi, umesto najnizeg bita */
if (x & (1 << (sizeof(x) * 8 - 1)))
32     return 1 + count(x << 1);
else
34     return count(x << 1);

```

## 1 Uvodni zadaci

---

```
36 }
37 /*****
38      Telo prethodne funkcije je moglo biti zapisano i krace:
39      jednolinijska return naredba sa proverom i rekurzivnim pozivom
40      return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1);
41 *****/
42
43
44 int main()
45 {
46     int x;
47     scanf("%x", &x);
48     printf("%d\n", count(x));
49
50     return 0;
51 }
```

### Rešenje 1.29

```
1  #include<stdio.h>
2
3  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne
4     cifre u broju */
5  int max_oktalna_cifra(unsigned x)
6  {
7      /* Izlazak iz rekurzije */
8      if (x == 0)
9          return 0;
10     /* Odredjivanje poslednje heksadekadne cifre u broju */
11     int poslednja_cifra = x & 7;
12     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz
13        njega izbrise poslednja oktalna cifra */
14     int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
15     return poslednja_cifra >
16         max_bez_poslednje_cifre ? poslednja_cifra :
17         max_bez_poslednje_cifre;
18 }
19
20 int main()
21 {
22     unsigned x;
23     scanf("%u", &x);
24     printf("%d\n", max_oktalna_cifra(x));
25     return 0;
26 }
```

### Rešenje 1.30

```

#include<stdio.h>

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

20
int main()
22{
    unsigned x;
24    scanf("%u", &x);
    printf("%d\n", max_heksadekadna_cifra(x));
26    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

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

12
int main()
14{
    char s[MAX_DIM];
16    int n;

18    scanf("%s", s);

20    /* Odredjuje se dužina niske */

```

## 1 Uvodni zadaci

---

```
    n = strlen(s);
22
    /* Ispisuje se poruka da li je niska palindrom ili
24       nije */
    if (palindrom(s, n))
26         printf("da\n");
    else
28         printf("ne\n");
30
    return 0;
}
```

### Rešenje 1.32

```
#include <stdio.h>
2 #include <stdlib.h>
#define MAX_DUZINA_NIZA 50
4
void ispisiNiz(int a[], int n)
6 {
    int i;
8
    for (i = 1; i <= n; i++)
10         printf("%d ", a[i]);
    printf("\n");
12 }

14 /* Funkcija proverava da li se x vec nalazi u permutaciji na
    prethodnih 1...n mesta */
16 int koriscen(int a[], int n, int x)
{
18     int i;
    for (i = 1; i <= n; i++)
20         if (a[i] == x)
            return 1;
22
    return 0;
24 }

26 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[]
    je niz u koji smesta permutacije m - oznacava da se na m-tu
28 poziciju u permutaciji smesta jedan od preostalih celih
    brojeva n- je velicina skupa koji se permutuje Funkciju
30 se poziva sa argumentom m=1 jer formiranje
    permutacije pocinje od 1. pozicije. Stoga, nece se koristiti a[0].
    */
32 void permutacija(int a[], int m, int n)
{
34     int i;

36     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti
```

```
    broj premasila velicinu skupa, onda se svi brojevi vec
38     nalaze u permutaciji i ispisuje se permutacija. */
    if (m > n) {
40         ispisiNiz(a, n);
        return;
42     }

44     /* Ideja: pronalazi se prvi broj koji moze da se postavi na
    m-to mesto u nizu (broj koji se do sada nije pojavio u
46     permutaciji). Zatim, rekursivno se pronalaze one permutacije
    koje odgovaraju ovako postavljenom pocetku permutacije.
48     Kada se to završi, vrši se provera da li postoji jos neki broj
    koji moze da se stavi na m-to mesto u nizu (to se radi u
50     petlji). Ako ne postoji, funkcija završava sa radom.
    Ukoliko takav broj postoji, onda se ponovo poziva rekursivno
52     pronalazenje odgovarajucih permutacija, ali sada sa
    drugacije postavljenim prefiksom. */

54

56     for (i = 1; i <= n; i++) {
        /* Ako se broj i nije do sada pojavio u permutaciji od 1 do
58         m-1 pozicije, onda se on postavlja na poziciju m i poziva se
        funkcija da napravi permutaciju za jedan vece duzine, tj.
60         m+1. Inace, nastavlja se dalje, trazeci broj koji se nije
        pojavio do sada u permutaciji. */
62         if (!koriscen(a, m - 1, i)) {
            a[m] = i;
64             /* Poziva se ponovo funkcija da dopuni ostatak permutacije
            posle upisivanja i na poziciju m. */
            permutacija(a, m + 1, n);
66         }
        }
68     }
}

70
72 int main(void)
73 {
    int n;
74     int a[MAX_DUZINA_NIZA];

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

84     permutacija(a, 1, n);

86     exit(EXIT_SUCCESS);
```

88 }

### Rešenje 1.33

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Rekurzivna funkcija za racunanje binomnog koeficijenta. */
5 /* ako je k=0 ili k=n, onda je binomni koeficijent 0 ako je k
   izmedju 0 i n, onda je bk(n,k) = bk(n-1,k-1) + bk(n-1,k) */
7 int binomniKoeficijent(int n, int k)
8 {
9     return (0 < k
10         && k < n) ? binomniKoeficijent(n - 1,
11                                     k - 1) +
12         binomniKoeficijent(n - 1, k) : 1;
13 }
14
15 /* Iterativno izracunavanje datog binomnog koeficijenta.
16
17     int binomniKoeficijent (int n, int k) { int i, j, b; for
18     (b=i=1, j=n; i<=k; b=b*j--/i++); return b; }
19
20 */
21
22 /* Prostim opaZanjem se uocava da se svaki element n-te
23 hipotenuze (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1
24 hipotenuze. Uz pomenute dve nove ivicne jedinice lako se
25 zakljucuje da ce suma elementa n-te hipotenuze biti tacno 2
26 puta veca. */
27 int sumaElemenataHipotenuze(int n)
28 {
29     return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
30 }
31
32 int main()
33 {
34     int n, k, i, d, r;
35
36     scanf("%d %d", &d, &r);
37
38     /* Ispisivanje Paskalovog trougla */
39     putchar('\n');
40     for (n = 0; n <= d; n++) {
41         for (i = 0; i < d - n; i++)
42             printf(" ");
43         for (k = 0; k <= n; k++)
44             printf("%4d", binomniKoeficijent(n, k));
45         putchar('\n');
46     }
47 }
```



```
49  if (r < 0) {  
    fprintf(stderr,  
51      "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"  
    );  
    exit(EXIT_FAILURE);  
53 }  
    printf("%d\n", sumaElemenataHipotenuze(r));  
55     exit(EXIT_SUCCESS);  
57 }
```



## Glava 2

# Pokazivači

### 2.1 Pokazivačka aritmetika

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

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

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

#### *Primer 1*

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

#### *Primer 2*

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 0
Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.1]

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

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

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

### Primer 1

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

### Primer 2

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

[Rešenje 2.2]

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

### Primer 1

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

### Primer 2

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

### Primer 3

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 0
Greska: neodgovarajuca dimenzija niza.
```

### Primer 4

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 101
Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.3]

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

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

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

### Primer 1

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

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

### Primer 2

```
POZIV: ./a.out

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

[Rešenje 2.4]

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

### Primer 1

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

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

### Primer 2

```
POZIV: ./a.out

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

[Rešenje 2.5]

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

### Primer 1

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

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

INTERAKCIJA PROGRAMA:
Broj reci ciji je broj karaktera 1 je 3.
```

### Primer 2

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

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

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

### Primer 3

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

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

[Rešenje 2.6]

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

### Primer 1

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

ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj ima reci
koje se zavrstavaju na ke

INTERAKCIJA PROGRAMA:
Broj reci koje se zavrstavaju na ke je 2.
```

### Primer 2

```
Poziv: ./a.out ulaz.txt sa -p

ULAZ.TXT
Ovo je sadrzaj datoteke i u njoj ima reci
koje pocinju sa sa

INTERAKCIJA PROGRAMA:
Broj reci koje pocinju na sa je 3.
```

*Primer 3*

```

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

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
Greska: Neuspesno otvaranje
datoteke ulaz.txt.

```

*Primer 4*

```

Poziv: ./a.out ulaz.txt

ULAZ.TXT
Ovo je sadrzaj ulaza.

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

```

[Rešenje 2.7]

## 2.2 Višedimenzioni nizovi

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

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

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

*Primer 1*

```

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

```

*Primer 2*

```

INTERAKCIJA PROGRAMA:
Unesite dimenziju matrice: 0
Greska: neodgovarajuca dimenzija matrice.

```

[Rešenje 2.8]

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

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

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

### Primer 1

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

[Rešenje 2.9]

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

- (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.

### Primer 1

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

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

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

[Rešenje 2.10]

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

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

- (b) Napisati funkciju koja određuje indeks kolone koja sadrži najmanji element matrice.
- (c) Napisati funkciju koja određuje indeks vrste koja sadrži najveći element matrice.
- (d) Napisati funkciju koja određuje broj negativnih elemenata matrice.

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

### Primer 1

```
Poziv: ./a.out 3

INTERAKCIJA PROGRAMA:
Unesite elemente matrice dimenzije 3:
1 2 3
-4 -5 -6
7 8 9
Najveci element matrice na sporednoj dijagonali je 7.
Indeks kolone koja sadrzi najmanji element matrice 2.
Indeks vrste koja sadrzi najveći element matrice 2.
Broj negativnih elemenata matrice je 3.
```

### Primer 2

```
Poziv: ./a.out 4

INTERAKCIJA PROGRAMA:
Unesite elemente matrice dimenzije 4:
-1 -2 -3 -4
-5 -6 -7 -8
-9 -10 -11 -12
-13 -14 -15 -16
Najveci element matrice na sporednoj dijagonali je -4.
Indeks kolone koja sadrzi najmanji element matrice 3.
Indeks vrste koja sadrzi najveći element matrice 0.
Broj negativnih elemenata matrice je 16.
```

[Rešenje 2.11]

**Zadatak 2.12** Napisati funkciju kojom se proverava da li je zadata kvadratna matrica dimenzije  $n$  ortonormirana. Matrica je ortonormirana ako je skalarni proizvod svakog para različitih vrsta jednak nuli, a skalarni proizvod vrste sa samom sobom jednak jedinici. Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju celobrojne kvadratne matrice  $n$  ( $0 < n \leq 32$ ), a zatim i njene elemente. Na standardni izlaz ispisati rezultat primene napisane funkcije na učitanoj matrici.

### Primer 1

```

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

```

### Primer 2

```

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

```

[Rešenje 2.12]

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

- (a) Napsiati funkciju koja učitava elemente matrice sa standardnog ulaza
- (b) Napsiati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice.

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

### Primer 1

```

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

```

### Primer 2

```

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

```

[Rešenje 2.13]

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

### Primer 1

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

## 2.3 Dinamička alokacija memorije

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

### Primer 1

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

### Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: -1
malloc(): neuspela alokacija memorije.
```

[Rešenje 2.15]

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

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

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite brojeve, nulu za kraj:
1 -2 3 -4 0
Unesite elemente niza:
1 -2 3 -4 0
Niz u obrnutom poretku je: -4 3 -2 1
```

[Rešenje 2.16]

**Zadatak 2.17** Napisati funkciju koja kao rezultat vraća nisku koja se dobija nadovezivanjem dve niske, bez promene njihovog sadržaja. Napisati program koji testira rad napisane funkcije. Sa standardnog ulaza učitati dve niske karaktera (pretpostaviti da niske nisu duže od 1000 karaktera i da ne sadrže praznine). Na standardni izlaz ispisati nisku koja se dobija njihovim nadovezivanjem. Za rezultujuću nisku dinamički alocirati memoriju.

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite dve niske karaktera:
Jedan Dva
Nadovezane niske: JedanDva
```

[Rešenje 2.17]

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

### Primer 1

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

[Rešenje 2.18]

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

- (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.

### Primer 1

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

[Rešenje 2.19]

**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. Ukoliko ima više takvih, ispisati prvu.

### Primer 1

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

### Primer 2

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

**Zadatak 2.21** Data je realna kvadratna matrica dimenzije  $n$ .

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

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

### Primer 1

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

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

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

[Rešenje 2.21]

**Zadatak 2.22** Napisati program koji na osnovu dve realne matrice dimenzija  $m \times n$  formira matricu dimenzije  $2 \cdot m \times n$  tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matrice su zapisane u datoteci „matrice.txt“. U prvom redu se nalaze dimenzije matrica  $m$  i  $n$ , u narednih  $m$  redova se nalaze vrste prve matrice, a u narednih  $m$  redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz.

### Primer 1

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

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

INTERAKCIJA PROGRAMA:
Trazena matrica je:
1.1 -2.2 3.3
-1.1 2.2 -3.3
-4.4 5.5 -6.6
4.4 -5.5 6.6
7.7 -8.8 9.9
-7.7 8.8 -9.9
```

**Zadatak 2.23** 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 elementa niza za jednu poziciju ulevo. Napisati program koji testira ovu funkciju. Rezultujuću matricu ispisati na standardni izlaz.

### Primer 1

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

**Zadatak 2.24** Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci „slicice.txt“ se nalaze informacije o sličicama koje mu nedostaju u formatu: **redni\_broj\_sličice ime\_reprezentacije\_kojoj\_sličica\_pripada**. Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronade ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. NAPOMENA: Za realokaciju memorije koristiti *realloc()* funkciju.

### Primer 1

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

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

**\*\* Zadatak 2.25** U datoteci „temena.txt“ se nalaze tačke koje predstavljaju temena nekog  $n$ -tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom  $n$ -touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan.



*Primer 1*

```

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

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

```

## 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`).

*Primer 1*

```

Poziv: ./a.out sin

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

```

### Primer 2

```

Poziv: ./a.out cos

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

```

[Rešenje 2.26]

**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):

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

### Primer 1

```

INTERAKCIJA PROGRAMA:
Unesite ime funkcije, n i a:
tan 1.570795 10000
Limes funkcije tan je -10134.5.

```

**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.

## Primer 1

```

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

```

**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 i  $n$ , a na standardni izlaz ispisuje vrednost odgovarajućeg integrala.

## Primer 1

```

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

```

## 2.5 Rešenja

## Rešenje 2.1

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 100
5
   /* Funkcija obrće elemente niza koriscenjem indekse sintakse */
7  void obrni_niz_v1(int a[], int n)
   {
9     int i, j;

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

```

```
17 }
18
19 /* Funkcija obrne elemente niza koriscenjem pokazivacke sintakse */
20 void obrni_niz_v2(int *a, int n)
21 {
22     /* Pokazivaci na elemente niza */
23     int *prvi, *poslednji;
24
25     /* Vrsi se obrtanje niza */
26     for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
27         int t = *prvi;
28
29         /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
30            vrednost koja se nalazi na adresi na koju pokazuje
31            pokazivac "poslednji". Nakon toga se pokazivac "prvi"
32            uvecava za jedan sto za posledicu ima da "prvi" pokazuje
33            na sledeci element u nizu */
34         *prvi++ = *poslednji;
35
36         /* Vrednost promenljive "t" se postavlja na adresu na koju
37            pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
38            umanjuje za jedan, sto za posledicu ima da pokazivac
39            "poslednji" sada pokazuje na element koji mu prethodi u
40            nizu */
41         *poslednji-- = t;
42     }
43
44     /* Drugi nacin za obrtanje niza */
45     /*
46     for (prvi = a, poslednji = a + n - 1;
47          prvi < poslednji; prvi++, poslednji--) {
48         int t = *prvi;
49         *prvi = *poslednji;
50         *poslednji = t;
51     }
52     */
53 }
54
55 int main()
56 {
57     /* Deklarise se niz od najvise MAX elemenata */
58     int a[MAX];
59
60     /* Broj elemenata niza a */
61     int n;
62
63     /* Pokazivac na elemente niza */
64     int *p;
65
66     printf("Unesite dimenziju niza: ");
67     scanf("%d", &n);
```

```

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

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

79  obrni_niz_v1(a, n);

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

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

87  obrni_niz_v2(a, n);

89  printf("Nakon ponovnog obrtanja elemenata, niz je:\n");

91  for (p = a; p - a < n; p++)
    printf("%d ", *p);
93  printf("\n");

95  return 0;
}

```

## Rešenje 2.2

```

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

4  #define MAX 100

6  /* Funkcija izracunava zbir elemenata niza */
double zbir(double *a, int n)
8  {
    double s = 0;
    int i;
10
    for (i = 0; i < n; s += a[i++]);
12
    return s;
14 }

16 /* Funkcija izracunava proizvod elemenata niza */
double proizvod(double a[], int n)
18 {

```

```
20     double p = 1;

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

24     return p;
26 }

28 /* Funkcija izracunava minimalni element niza */
double min(double *a, int n)
30 {
    /* Na pocetku, minimalni element je prvi element */
32     double min = a[0];
    int i;

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

40     return min;
42 }

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

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

56     return max;
58 }

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

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

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

```

72     exit(EXIT_FAILURE);
73 }
74
75 printf("Unesite elemente niza:\n");
76 for (i = 0; i < n; i++)
77     scanf("%lf", a + i);
78
79 /* Vrsi se testiranje definisanih funkcija */
80 printf("Zbir elemenata niza je %5.3f.\n", zbir(a, n));
81 printf("Proizvod elemenata niza je %5.3f.\n", proizvod(a, n));
82 printf("Minimalni element niza je %5.3f.\n", min(a, n));
83 printf("Maksimalni element niza je %5.3f.\n", max(a, n));
84
85 return 0;
86 }

```

### Rešenje 2.3

```

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

```

```
        (*poslednji--)--;
35    }
    }
37
int main()
39 {
    int a[MAX];
41    int n;
    int *p;
43
    printf("Unesite dimenziju niza: ");
45    scanf("%d", &n);

47    /* Proverava se da li je doslo do prekoračenja ograničenja
        dimenzije */
49    if (n <= 0 || n > MAX) {
        fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
51        exit(EXIT_FAILURE);
    }
53
    printf("Unesite elemente niza:\n");
55    for (p = a; p - a < n; p++)
        scanf("%d", p);
57
    povecaj_smanji(a, n);
59
    printf("Transformisan niz je:\n");
61    for (p = a; p - a < n; p++)
        printf("%d ", *p);
63    printf("\n");

65    return 0;
}
```

### Rešenje 2.4

```
#include <stdio.h>
2
int main(int argc, char *argv[])
4 {
    int i;
    char tip_ispisa;
6
    printf("Broj prihvacenih argumenata komandne linije je %d.\n", argc
    );
8
    printf("Kako zelite da ispisete argumente, ");
    printf("koriscenjem indeksne ili pokazivacke sintakse (I ili P)? ")
    ;
12    scanf("%c", &tip_ispisa);
}
```



```

14 printf("Argumenti komandne linije su:\n");
15 if(tip_ispisa=='I') {
16     /* Ispisuju se argumenti komandne linije koriscenjem indeksne
17        sintakse */
18     for (i = 0; i < argc; i++)
19         printf("%d %s\n", i, argv[i]);
20 } else if(tip_ispisa=='P'){
21     /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
22        sintakse */
23     i = argc;
24     for (; argc > 0; argc--)
25         printf("%d %s\n", i - argc, *argv++);
26
27     /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje
28        na polje u memoriji koje se nalazi iza poslednjeg argumenta
29        komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
30        broja argumenta komandne linije to sada moze ponovo da se
31        postavi "argv" da pokazuje na nulti argument komandne linije */
32     argv = argv - i;
33     argc = i;
34 }

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

47
48 return 0;
49 }

```

## Rešenje 2.5

```

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

4
5  /* Funkcija ispituje da li je niska palindrom */
6  int palindrom(char *niska)
7  {
8      int i, j;
9      for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
10         if (*(niska + i) != *(niska + j))
11             return 0;

```

```
    return 1;
13 }

15 int main(int argc, char **argv)
16 {
17     int i, n = 0;

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

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

### Rešenje 2.6

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

4 #define MAX_KARAKTERA 100

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

14 int main(int argc, char **argv)
15 {
16     char rec[MAX_KARAKTERA];
    int br = 0, n;
18     FILE *in;

20     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska */
22     if (argc < 3) {
        printf("Greska: ");
24         printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_dat br_karaktera.\n",
26             argv[0]);
        exit(EXIT_FAILURE);
28     }

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

```

34     if (in == NULL) {
        fprintf(stderr, "Greska: ");
        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
36             argv[1]);
        exit(EXIT_FAILURE);
38     }

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

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

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

50     /* Zatvara se datoteka */
    fclose(in);
52     return 0;
}

```

## Rešenje 2.7

```

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

4  #define MAX_KARAKTERA 100

6  /* Implementacija funkcije strcpy() iz standardne biblioteke */
void kopiranje_niske(char *dest, char *src)
8  {
    int i;
10     for (i = 0; *(src + i); i++)
        *(dest + i) = *(src + i);
12 }

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

24 /* Implementacija funkcije strlen() iz standardne biblioteke */
int duzina_niske(char *s)
26 {
    int i;

```

```
28     for (i = 0; *(s + i); i++);
29     return i;
30 }
31
32 /* Funkcija ispituje da li je niska zadata drugim argumentom
33    funkcije sufiks niske zadate prvi argumentom funkcije */
34 int sufiks_niske(char *niska, char *sufiks)
35 {
36     if (duzina_niske(sufiks) <= duzina_niske(niska) &&
37         poredjenje_niski(niska + duzina_niske(niska) -
38                         duzina_niske(sufiks), sufiks) == 0)
39         return 1;
40     return 0;
41 }
42
43 /* Funkcija ispituje da li je niska zadata drugim argumentom
44    funkcije prefiks niske zadate prvi argumentom funkcije */
45 int prefiks_niske(char *niska, char *prefiks)
46 {
47     int i;
48     if (duzina_niske(prefiks) <= duzina_niske(niska)) {
49         for (i = 0; i < duzina_niske(prefiks); i++)
50             if (*(prefiks + i) != *(niska + i))
51                 return 0;
52         return 1;
53     } else
54         return 0;
55 }
56
57 int main(int argc, char **argv)
58 {
59     /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
60        greska */
61     if (argc < 4) {
62         printf("Greska: ");
63         printf("Nedovoljan broj argumenata komandne linije.\n");
64         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
65               argv[0]);
66         exit(EXIT_FAILURE);
67     }
68
69     FILE *in;
70     int br = 0;
71     char rec[MAX_KARAKTERA];
72
73     in = fopen(*(argv + 1), "r");
74     if (in == NULL) {
75         fprintf(stderr, "Greska: ");
76         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
77               argv[1]);
78         exit(EXIT_FAILURE);
79     }
80 }
```

```

80  /* Provera se opcija kojom je pozvan program a zatim se
82  učitavaju reci iz datoteke i broji se koliko njih zadovoljava
      traženi uslov */
84  if (!(poredjenje_niski(*(argv + 3), "-s"))) {
      while (fscanf(in, "%s", rec) != EOF)
86      br += sufiks_niske(rec, *(argv + 2));
      printf("Broj reci koje se završavaju na %s je %d.\n", *(argv + 2),
      br);
88  } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
      while (fscanf(in, "%s", rec) != EOF)
90      br += prefiks_niske(rec, *(argv + 2));
      printf("Broj reci koje počinju na %s je %d.\n", *(argv + 2), br);
92  }

94  fclose(in);
      return 0;
96  }

```

## Rešenje 2.8

```

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

4
#define MAX 100

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

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

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

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

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

```

```

32  /* Ispis elemenata matrice koriscenjem indeksne sintakse.
    Ispis vrsimo vrstu po vrstu */
34  for (i = 0; i < n; i++) {
    /* Ispisujemo elemente i-te vrste */
36      for (j = 0; j < n; j++)
          printf("%d ", A[i][j]);
38      printf("\n");
    }

40
42  /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
    Kod ovako definisane matrice, elementi su uzastopno
    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++) { for ( j=0 ; j<n; j++) printf("%d ",
50      *(A+i+j)); printf("\n"); } */

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

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

58
60  return 0;
    }

62  /* Definisemo funkcije koju smo ranije deklarisisali */

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

72
74  /* Funkcija izracunava euklidsku normu matrice */
    double euklidska_norma(int M[][MAX], int n)
76  {
        double norma = 0.0;
        int i, j;

78
80        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                norma += M[i][j] * M[i][j];
82

```

```

    return sqrt(norma);
84 }

86 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
87 int gornja_vandijagonalna_norma(int M[][MAX], int n)
88 {
89     int norma = 0;
90     int i, j;

92     for (i = 0; i < n; i++) {
93         for (j = i + 1; j < n; j++)
94             norma += abs(M[i][j]);
95     }

96     return norma;
97 }
98 }

```

### Rešenje 2.9

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 100
5
6  /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
7   standardnog ulaza */
8  void ucitaj_matricu(int m[][MAX], int n)
9  {
10     int i, j;

11
12     for (i = 0; i < n; i++)
13         for (j = 0; j < n; j++)
14             scanf("%d", &m[i][j]);
15 }

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

22
23     for (i = 0; i < n; i++) {
24         for (j = 0; j < n; j++)
25             printf("%d ", m[i][j]);
26         printf("\n");
27     }
28 }

29
30 /* Funkcija proverava da li su zadate kvadratne matrice a i b
31 dimenzije n jednake */
32 int jednake_matrice(int a[][MAX], int b[][MAX], int n)

```

```
33 {
34     int i, j;
35
36     for (i = 0; i < n; i++)
37         for (j = 0; j < n; j++)
38             /* Nasli smo elemente na istim pozicijama u matricama koji
39              se razlikuju */
40             if (a[i][j] != b[i][j])
41                 return 0;
42
43     /* Prošla je provera jednakosti za sve parove elemenata koji
44        su na istim pozicijama sto znaci da su matrice jednake */
45     return 1;
46 }
47
48 /* Funkcija izracunava zbir dve kvadratne matrice */
49 void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
50 {
51     int i, j;
52
53     for (i = 0; i < n; i++)
54         for (j = 0; j < n; j++)
55             c[i][j] = a[i][j] + b[i][j];
56 }
57
58 /* Funkcija izracunava proizvod dve kvadratne matrice */
59 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
60 {
61     int i, j, k;
62
63     for (i = 0; i < n; i++)
64         for (j = 0; j < n; j++) {
65             /* Mnozimo i-tu vrstu prve sa j-tom kolonom druge matrice */
66             c[i][j] = 0;
67             for (k = 0; k < n; k++)
68                 c[i][j] += a[i][k] * b[k][j];
69         }
70 }
71
72 int main()
73 {
74     /* Matrice ciji se elementi zadaju sa ulaza */
75     int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
76
77     /* Matrice zbira i proizvoda */
78     int zbir[MAX][MAX], proizvod[MAX][MAX];
79
80     /* Dimenzija matrica */
81     int n;
82     int i, j;
83
84     /* Ucitavamo dimenziju kvadratnih matrica i proveravamo njenu
```



```

85     korektnost */
scanf("%d", &n);

87

/* Proveravamo da li je prekoraceno ogranicenje */
89 if (n > MAX || n <= 0) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
91     fprintf(stderr, "matrica.\n");
    exit(EXIT_FAILURE);
93 }

95 /* Ucitavamo matrice */
ucitaj_matricu(a, n);
97 ucitaj_matricu(b, n);

99 /* Izracunavamo zbir i proizvod matrica */
saberu(a, b, zbir, n);
101 pomnozi(a, b, proizvod, n);

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

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

111 printf("Proizvod matrica je:\n");
ispisi_matricu(proizvod, n);
113

115 return 0;
}

```

## Rešenje 2.10

```

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

4  #define MAX 64

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

14     /* Obilazimo glavnu dijagonalu matrice. Za elemente na glavnoj
       dijagonali vazi da je indeks vrste jednak indeksu kolone */
16     for (i = 0; i < n; i++) {

```

```
        if (m[i][i] != 1)
18         return 0;
    }

20     return 1;
22 }

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

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

36     /* Postavljamo na glavnoj dijagonali jedinice */
38     for (i = 0; i < n; i++)
        zatvorenje[i][i] = 1;
40 }

42 /* Funkcija proverava da li je relacija simetricna. Relacija je
    simetricna ako za svaki par elemenata vazi: ako je element
44 "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)
    {
48         int i, j;

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

56         return 1;
58     }

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

68         /* Prepisujemo vrednosti elemenata matrice m */
```

```

70     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            zatvorenje[i][j] = m[i][j];
72
73     /* Odredjujemo simetricno zatvorenje matrice */
74     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
76         if (zatvorenje[i][j] == 1)
            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
82 "i" 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             /* Pokusavamo da pronadjemo element koji narusava *
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 zadate
101 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++)
114         for (i = 0; i < n; i++)
115             for (j = 0; j < n; j++)
116                 if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
117                     && (zatvorenje[i][j] == 0))
118                     zatvorenje[i][j] = 1;
119
120 }

```

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

```

174         fscanf(ulaz, "%d", &m[i][j]);
175
176     /* Ispisujemo trazene vrednosti */
177     printf("Refleksivnost: %s\n",
178           refleksivnost(m, n) == 1 ? "da" : "ne");
179
180     printf("Simetricnost: %s\n",
181           simetricnost(m, n) == 1 ? "da" : "ne");
182
183     printf("Tranzitivnost: %s\n",
184           tranzitivnost(m, n) == 1 ? "da" : "ne");
185
186     printf("Refleksivno zatvorenje:\n");
187     ref_zatvorenje(m, n, pomocna);
188     pisi_matricu(pomocna, n);
189
190     printf("Simetricno zatvorenje:\n");
191     sim_zatvorenje(m, n, pomocna);
192     pisi_matricu(pomocna, n);
193
194     printf("Refleksivno-tranzitivno zatvorenje:\n");
195     tran_zatvorenje(m, n, pomocna);
196     pisi_matricu(pomocna, n);
197
198     /* Zatvaramo datoteku */
199     fclose(ulaz);
200
201     return 0;
202 }

```

### Rešenje 2.11

```

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

#define MAX 32

int max_sporedna_dijagonala(int m[][MAX], int n)
{
    int i, j;
    /* Trazimo najveći element na sporednoj dijagonali. Za
       elemente sporedne dijagonale vazi da je zbir indeksa vrste
       i indeksa kolone jednak n-1. Za pocetnu vrednost maksimuma
       uzimamo element u gornjem desnom uglu */
    int max_na_sporednoj_dijagonali = m[0][n - 1];
    for (i = 1; i < n; i++)
        if (m[i][n - 1 - i] > max_na_sporednoj_dijagonali)
            max_na_sporednoj_dijagonali = m[i][n - 1 - i];

    return max_na_sporednoj_dijagonali;
}

```

```
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 {
44     int i, j;
45     /* Za maksimalni element uzimamo gornji levi ugao */
46     int max = m[0][0], indeks_vrste = 0;
47
48     for (i = 0; i < n; i++)
49         for (j = 0; j < n; j++)
50             /* Ako je tekuci element manji od minimalnog */
51             if (m[i][j] > max) {
52                 /* cuvamo njegovu vrednost */
53                 max = m[i][j];
54                 /* i cuvamo indeks vrste u kojoj se nalazi */
55                 indeks_vrste = i;
56             }
57     return indeks_vrste;
58 }
59
60 /* Funkcija izracunava broj negativnih elemenata matrice */
61 int broj_negativnih(int m[][MAX], int n)
62 {
63     int i, j;
64
65     int broj_negativnih = 0;
66
67     for (i = 0; i < n; i++)
68         for (j = 0; j < n; j++)
69             if (m[i][j] < 0)
70                 broj_negativnih++;
71     return broj_negativnih;
72 }
```

```

72 }

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

80     /* Proveravamo broj argumenata komandne linije */
81     if (argc < 2) {
82         printf("Greska: ");
83         printf("Nedovoljan broj argumenata komandne linije.\n");
84         printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
85         exit(EXIT_FAILURE);
86     }

88     /* Ucitavamo vrednost dimenzije i proveravamo njenu korektnost
89     */
90     n = atoi(argv[1]);

92     if (n > MAX || n <= 0) {
93         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
94         fprintf(stderr, "matrice.\n");
95         exit(EXIT_FAILURE);
96     }

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

102     int max_sd = max_sporedna_dijagonala(m, n);
103     int i_min = indeks_min(m, n);
104     int i_max = indeks_max(m, n);
105     int bn = broj_negativnih(m, n);

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

110     /* Prekidamo izvršavanje programa */
112     return 0;
}

```

## 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
17    izlaz */
18 void ispisi_matricu(int m[][MAX], int n)
19 {
20     int i, j;

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

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

34     /* Proveravamo uslov normiranosti, odnosno da li je proizvod
35        svake vrste matrice sa samom sobom jednak jedinici */
36     for (i = 0; i < n; i++) {
37
38         /* Izracunavamo skalarni proizvod vrste sa samom sobom */
39         proizvod = 0;

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

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

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

51             /* Izracunavamo skalarni proizvod */
52             proizvod = 0;
```



```

60     for (k = 0; k < n; k++)
        proizvod += m[i][k] * m[j][k];
62
        /* Ako proizvod dve bilo koje razlicite vrste nije jednak
64         nuli, odmah zakljucujemo da matrica nije ortogonalna */
        if (proizvod != 0)
66             return 0;
        }
68     }

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

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

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

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

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

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

98    return 0;
}

```

### Rešenje 2.13

```

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

4 #define MAX_V 10
#define MAX_K 10

6
/* Funkcija proverava da li su ispisani svi elementi iz matrice,

```

```
8      odnosno da li se narušio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
10 {
11     return !(top <= bottom && left <= right);
12 }
13
14 /* Funkcija spiralno ispisuje elemente matrice */
15 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16 {
17     int i, j, top, bottom, left, right;
18
19     top = left = 0;
20     bottom = n - 1;
21     right = m - 1;
22
23     while (!krajIspisa(top, bottom, left, right)) {
24         /* Ispisuje se prvi red */
25         for (j = left; j <= right; j++)
26             printf("%d ", a[top][j]);
27
28         /* Spustamo prvi red */
29         top++;
30
31         if (krajIspisa(top, bottom, left, right))
32             break;
33
34         for (i = top; i <= bottom; i++)
35             printf("%d ", a[i][right]);
36
37         /* Pomeramo desnu kolonu za naredni krug ispisa blize levom
38            kraju */
39         right--;
40
41         if (krajIspisa(top, bottom, left, right))
42             break;
43
44         /* Ispisujemo donju vrstu */
45         for (j = right; j >= left; j--)
46             printf("%d ", a[bottom][j]);
47
48         /* Podizemo donju vrstu za naredni krug ispisa */
49         bottom--;
50
51         if (krajIspisa(top, bottom, left, right))
52             break;
53
54         /* Ispisujemo prvu kolonu */
55         for (i = bottom; i >= top; i--)
56             printf("%d ", a[i][left]);
57
58         /* Pripremamo levu kolonu za naredni krug ispisa */
59         left++;
60     }
```

```

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

64 void ucitaj_matricu(int a[][MAX_K], int n, int m)
65 {
66     int i, j;

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

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

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

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

86     ucitaj_matricu(a, n, m);
87     ispisi_matricu_spiralno(a, n, m);

89     return 0;
90 }

```

### Rešenje 2.14

### Rešenje 2.15

```

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

4 /* NAPOMENA: Primer demonstrira dinamičku alokaciju niza od n
5    elemenata. Dovoljno je alocirati n * sizeof(T) bajtova, gde
6    je T tip elemenata niza. Povratnu adresu malloc()-a treba
7    pretvoriti iz void * u T *, kako bismo dobili pokazivac koji
8    pokazuje na prvi element niza tipa T. Na dalje se elementima
9    može pristupiti na isti način kao da nam je dato ime niza
10   (koje se tako i ponasa - kao pokazivac na element tipa T koji
11   je prvi u nizu) */
12 int main()

```

```
{
14  int *p = NULL;
16  int i, n;

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

22

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

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

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

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

42

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

46    return 0;
}
```

### Rešenje 2.16

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

5  int main(void)
    {
7      /* Adresa prvog alociranog bajta */
        int *a = NULL;

9

        /* Velicina alocirane memorije */
11       int alocirano;

13

        /* Broj elemenata niza */
```

```
15  int n;
16
17  /* Broj koji se učitava sa ulaza */
18  int x;
19  int i;
20  int *b = NULL;
21
22  /* Inicijalizacija */
23  alocirano = n = 0;
24
25  /* Unosimo brojeve sa ulaza */
26  scanf("%d", &x);
27
28  /* Sve dok je procitani broj razlicit od nule... */
29  while (x != 0) {
30
31      /* Ako broj ucitanih elemenata niza odgovara broju
32       alociranih mesta, za smestanje novog elementa treba
33       obezbediti dodatni prostor. Da se ne bi za svaki sledeci
34       element pojedinačno alocirala memorija, prilikom
35       alokacije se vrsi rezervacija za jos KORAK dodatnih mesta
36       za buduće elemente */
37      if (n == alocirano) {
38          /* Povecava se broj alociranih mesta */
39          alocirano = alocirano + KORAK;
40
41          /* Vrsi se realokacija memorije sa novom velicinom */
42          /******
43          /* Resenje sa funkcijom malloc() */
44          /******
45          /* Vrsi se alokacija memorije sa novom velicinom, a adresa
46             pocetka novog memorijskog bloka se cuva u promenljivoj
47             b */
48          b = (int *) malloc(alocirano * sizeof(int));
49
50          /* Ako prilikom alokacije dodje do neke greske */
51          if (b == NULL) {
52              /* poruku ispisujemo na izlaz za greske */
53              fprintf(stderr, "malloc(): ");
54              fprintf(stderr, "greska pri alokaciji memorije.\n");
55
56              /* Pre kraja programa moramo svu dinamicki alociranu
57                 memoriju da oslobodimo. U ovom slucaju samo memoriju
58                 na adresi a */
59              free(a);
60
61              /* Završavamo program */
62              exit(EXIT_FAILURE);
63          }
64
65          /* Svih n elemenata koji pocinju na adresi a prepisujemo
66             na novu adresu b */
```

```
67     for (i = 0; i < n; i++)
        b[i] = a[i];

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

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

77     /******
79     /* Resenje sa funkcijom realloc() */
    /******
81     /* Zbog funkcije realloc je neophodno da i u prvoj
        iteraciji "a" bude inicijalizovano na NULL */
83     /*
        a = (int*) realloc(a, alocirano * sizeof(int));

85
87     if(a == NULL) { fprintf(stderr, "realloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE); } */
89 }

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

93
95     /* i učitavamo sledeći element */
    scanf("%d", &x);
97 }

99     /* Ispisujemo brojeve u obrnutom poretaku */
    for (n--; n >= 0; n--)
        printf("%d ", a[n]);
101 printf("\n");

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

105
107 /* Program se završava */
    exit(EXIT_SUCCESS);
}
```

### Rešenje 2.17

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

5 #define MAX 1000
```

```
7  /* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
9  Ovako nesto se moze improvizovati tako sto se u funkciji
   dinamički kreira niz potrebne velicine, popuni se potrebnim
11  informacijama, a zatim se vrati njegova adresa. Imajuci u
   vidu cinjenicu da dinamički kreiran objekat ne nestaje kada
13  se izadje iz funkcije koja ga je kreirala, vraceni pokazivac
   se kasnije u pozivajucoj funkciji moze koristiti za pristup
15  "vracenom" nizu. Medjutim, pozivajuca funkcija ima
   odgovornost i da se brine o dealokaciji istog prostora */

17  /* Funkcija dinamički kreira niz karaktera u koji smesta
   rezultat nadovezivanja niski. Adresa niza se vraca kao
19  povratna vrednost. */
   char *nadovezi(char *s, char *t)
21  {
   /* Dinamički kreiramo prostor dovoljne velicine */
23   char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
                               * sizeof(char));

25   /* Proveravamo uspeh alokacije */
27   if (p == NULL) {
       fprintf(stderr, "malloc(): ");
29   fprintf(stderr, "greska pri alokaciji memorije.\n");
       exit(EXIT_FAILURE);
31   }

33   /* Kopiramo i nadovezujemo stringove */

35   /* Resenje bez koriscenja biblioteckih funkcija */
   /*
37   int i,j; for(i=j=0; s[j]!='\0'; i++, j++) p[i]=s[j];

39   for(j=0; t[j]!='\0'; i++, j++) p[i]=t[j];

41   p[i]='\0'; */

43   /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
       string.h */
45   strcpy(p, s);
       strcat(p, t);

47   /* Vracamo pokazivac p */
49   return p;
   }

51  int main()
53  {
       char *s = NULL;
55   char s1[MAX], s2[MAX];

57   /* Ucitavamo dve niske koje cemo da nadovezemo */
```

## 2 Pokazivači

```
scanf("%s", s1);
59 scanf("%s", s2);

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

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

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

69 return 0;
71 }
```

### Rešenje 2.18

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

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

9     /* Pokazivac na dinamički alociran niz pokazivaca na vrste
matrice */
11     double **A = NULL;

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

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

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

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

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

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



```

35     if (A[i] == NULL) {
36         /* Alokacija je neuspesna. Pre zavrsetka programa moramo
37            da oslobodimo svih i-1 prethodno alociranih vrsta, i
38            alociran niz pokazivaca */
39         for (j = 0; j < i; j++)
40             free(A[j]);
41         free(A);
42
43         exit(EXIT_FAILURE);
44     }
45 }
46
47 /* Unosimo sa standardnog ulaza brojeve u matricu. Popunjavamo
48 vrstu po vrstu */
49 for (i = 0; i < n; i++)
50     for (j = 0; j < m; j++)
51         scanf("%lf", &A[i][j]);
52
53 /* Racunamo trag matrice, odnosno sumu elemenata na glavnoj
54 dijagonali */
55 trag = 0.0;
56
57 for (i = 0; i < n; i++)
58     trag += A[i][i];
59
60 printf("%.2f\n", trag);
61
62 /* Oslobadjamo prostor rezervisan za svaku vrstu */
63 for (j = 0; j < n; j++)
64     free(A[j]);
65
66 /* Oslobadjamo memoriju za niz pokazivaca na vrste */
67 free(A);
68
69 return 0;
70 }

```

### Rešenje 2.19

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  void ucitaj_matricu(int **M, int n, int m)
6  {
7      int i, j;
8
9      /* Popunjavamo matricu vrstu po vrstu */
10     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;
31     int **matrica = NULL;

33     /* Unosimo dimenzije matrice */
    scanf("%d %d", &n, &m);
35

    /* Alociramo prostor za niz pokazivaca na vrste matrice */
37     matrica = (int **) malloc(n * sizeof(int *));
    if (matrica == NULL) {
39         fprintf(stderr, "malloc(): Neuspela alokacija\n");
        exit(EXIT_FAILURE);
41     }

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

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

55
    ucitaj_matricu(matrica, n, m);

57
    ispisi_elemente_ispod_dijagonale(matrica, n, m);

59

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

```

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

69  return 0;
}

```

### Rešenje 2.20

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

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

```

```
40     return zbir;
41 }
42
43 /* Funkcija učitava elemente kvadratne matrice dimenzije n iz
44    zadate datoteke */
45 void ucitaj_matricu(FILE * ulaz, float **m, int n)
46 {
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 na
55    standardni izlaz */
56 void ispisi_matricu(float **m, int n)
57 {
58     int i, j;
59
60     for (i = 0; i < n; i++) {
61         for (j = 0; j < n; j++)
62             printf("%.2f ", m[i][j]);
63         printf("\n");
64     }
65 }
66
67 /* Funkcija alokira memoriju za kvadratnu matricu dimenzije n */
68 float **alociraj_memoriju(int n)
69 {
70     int i, j;
71     float **m;
72
73     m = (float **) malloc(n * sizeof(float *));
74     if (m == NULL) {
75         fprintf(stderr, "malloc(): Neuspela alokacija\n");
76         exit(EXIT_FAILURE);
77     }
78
79     /* Za svaku vrstu matrice */
80     for (i = 0; i < n; i++) {
81         /* Alociramo memoriju */
82         m[i] = (float *) malloc(n * sizeof(float));
83
84         /* Proveravamo da li je doslo do greske pri alokaciji */
85         if (m[i] == NULL) {
86             /* Ako jeste, ispisujemo poruku */
87             printf("malloc(): neuspela alokacija memorije!\n");
88
89             /* Oslobadjamo memoriju zauzetu do ovog koraka */
90             for (j = 0; j < i; j++)
```

```

    free(m[i]);
92     free(m);
    exit(EXIT_FAILURE);
94 }
}
96 return m;
}
98
/* Funkcija oslobadja memoriju zauzetu kvadratnom matricom
100 dimenzije n */
void oslobodi_memoriju(float **m, int n)
102 {
    int i;
104
    for (i = 0; i < n; i++)
106         free(m[i]);
    free(m);
108 }

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

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

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

134     /* citamo dimenziju matrice */
    fscanf(ulaz, "%d", &n);
136
    /* Alociramo memoriju */
138     a = alociraj_memoriju(n);

140     /* Ucitavamo elemente matrice */
    ucitaj_matricu(ulaz, a, n);
142
```

```
144     float zbir = zbir_ispod_sporedne_dijagonale(a, n);
146     /* Pozivamo funkciju za modifikovanje elemenata */
148     izmeni(a, n);
150     /* Ispisujemo rezultat */
152     printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
154     printf("je %.2f.\n", zbir);
156     printf("Transformisana matrica je:\n");
158     ispisi_matricu(a, n);
160     /* Oslobadjamo memoriju */
162     oslobodi_memoriju(a, n);
164     /* Zatvaramo datoteku */
166     fclose(ulaz);
168     /* i prekidamo sa izvršavanjem programa */
170     return 0;
172 }
```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

Rešenje 2.26

```
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <math.h>
5 #include <string.h>
6
7 /* NAPOMENA: Zaglavlje math.h sadrzi deklaracije raznih
8  matematičkih funkcija. dIzmeu ostalog, to su čsledee
9  funkcije: double sin(double x); double cos(double x); double
10  tan(double x); double asin(double x); double acos(double x);
11  double atan(double x); double atan2(double y, double x);
12  double sinh(double x); double cosh(double x); double
13  tanh(double x); double exp(double x); double log(double x);
14  double log10(double x); double pow(double x, double y);
15  double sqrt(double x); double ceil(double x); double
```

```

16     floor(double x); double fabs(double x); */

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

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

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

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

54     /* Ako korisnik nije uneo žtraene argumente, prijavljujemo
        šgreku */
56     if (argc < 2) {
        printf("Greska: ");
58         printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
60             argv[0]);
        exit(EXIT_FAILURE);
62     }

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

```

```
68  /* Inicijalizujemo čpokaziva na funkciju koja treba da se
    tabelira */
70  if (strcmp(ime_fje, "sin") == 0)
    fp = &sin;
72  else if (strcmp(ime_fje, "cos") == 0)
    fp = &cos;
74  else if (strcmp(ime_fje, "tan") == 0)
    fp = &tan;
76  else if (strcmp(ime_fje, "atan") == 0)
    fp = &atan;
78  else if (strcmp(ime_fje, "acos") == 0)
    fp = &acos;
80  else if (strcmp(ime_fje, "asin") == 0)
    fp = &asin;
82  else if (strcmp(ime_fje, "exp") == 0)
    fp = &exp;
84  else if (strcmp(ime_fje, "log") == 0)
    fp = &log;
86  else if (strcmp(ime_fje, "log10") == 0)
    fp = &log10;
88  else if (strcmp(ime_fje, "sqrt") == 0)
    fp = &sqrt;
90  else if (strcmp(ime_fje, "floor") == 0)
    fp = &floor;
92  else if (strcmp(ime_fje, "ceil") == 0)
    fp = &ceil;
94  else if (strcmp(ime_fje, "sqr") == 0)
    fp = &sqr;
96  else {
    printf("Program jos uvek ne podrzava trazenu funkciju!\n");
    exit(EXIT_SUCCESS);
98  }

100
    printf("Unesite krajeve intervala:\n");
102    scanf("%lf %lf", &a, &b);

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

108    /* Mreza mora da čukljuuje bar krajeve intervala, tako da se
        mora uneti broj veci od 2 */
110    if (n < 2) {
        fprintf(stderr, "Broj čtaaka žmree mora biti bar 2!\n");
        exit(EXIT_FAILURE);
112    }

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

118    /* dProsleujemo funkciji tabela() funkciju zadatu kao
        argument komandne linije */
```



```
120     tabela(a, b, n, fp);  
122     exit(EXIT_SUCCESS);  
}
```

Rešenje [2.27](#)

Rešenje [2.28](#)

Rešenje [2.29](#)



## Glava 3

# Algoritmi pretrage i sortiranja

### 3.1 Pretraživanje

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

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

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

#### Test 1

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

```
IZLAZ:
```

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

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

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

#### Test Test 2

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

```
IZLAZ:
```

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

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

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

[Rešenje 3.1]

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

#### Primer 1

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

#### Primer 2

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

[Rešenje 3.2]

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks studenta čije informacije se potom prikazuju na ekranu. Zatim, korisnik unosi prezime studenta i prikazuju mu se informacije o prvom studentu sa unetim prezimenom. U slučaju neuspješnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

*Primer 1*

```

Poziv: ./a.out datoteka.txt

DATOTEKA.TXT
20140003 Marina Petrovic
20140012 Stefan Mitrovic
20140032 Dejan Popovic
20140049 Mirko Brankovic
20140076 Sonja Stevanovic
20140104 Ivan Popovic
20140187 Vlada Stankovic
20140234 Darko Brankovic

INTERAKCIJA PROGRAMA:
Unesite indeks studenta cije informacije zelite: 20140076
Indeks: 20140076, Ime i prezime: Sonja Stevanovic
Unesite prezime studenta cije informacije zelite: Popovic
Indeks: 20140032, Ime i prezime: Dejan Popovic

```

[Rešenje 3.3]

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

[Rešenje 3.4]

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

*Test 1*

```

Poziv: ./a.out dat.txt -x

DAT.TXT
12 53
2.342 34.1
-0.3 23
-1 23.1
123.5 756.12

IZLAZ:
-0.3 23

```

*Test 2*

```

Poziv: ./a.out dat.txt

DAT.TXT
12 53
2.342 34.1
-0.3 23
-1 2.1
123.5 756.12

IZLAZ:
-1 2.1

```

*Test 3*

```

Poziv: ./a.out dat.txt -y

DAT.TXT
12 53
2.342 34.1
-0.3 0.23
-1 2.1
123.5 756.12

IZLAZ:
-0.3 0.23

```

[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

### 3 Algoritmi pretrage i sortiranja

---

vrednost kosinusne funkcije razlikuje za najviše 0.001 od nule. UPUTSTVO: *Koristiti algoritam analogan algoritmu binarne pretrage.*

*Test 1*

```
|| IZLAZ:
|| 1.57031
```

[Rešenje 3.6]

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

*Test 1*

```
|| 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 indeks prvog elementa manjeg od nule. Ukoliko nema elemenata manjih od nule, funkcija kao rezultat vraća -1. Napisati program koji testira ovu funkciju za opadajući niz celih brojeva koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

*Test 1*

```
|| ULAZ:
|| 151 44 5 -12 -13 -15
||
|| IZLAZ:
|| 3
```

*Test 2*

```
|| ULAZ:
|| 100 55 15 0 -15 -124
|| -155 -258 -315 -516
||
|| IZLAZ:
|| 4
```

*Test 3*

```
|| ULAZ:
|| 100 15 11 8 7 5 4 3 2
||
|| IZLAZ:
|| -1
```

[Rešenje 3.8]

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

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

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ:	ULAZ:	ULAZ:
4	17	1031
IZLAZ:	IZLAZ:	IZLAZ:
2 2	4 4	10 10

[Rešenje 3.9]

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

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 3</i>
INTERAKCIJA PROGRAMA:	INTERAKCIJA PROGRAMA:	INTERAKCIJA PROGRAMA:
Unesi broj tacaka: 2	Unesi broj tacaka: 2	Unesi broj tacaka: 3
Unesi koordinate tacaka:	Unesi koordinate tacaka:	Unesi koordinate tacaka:
2 0 2 1	1 0 1 1	1 0 1 1
1 2 2 2	0 1 1 1	2 0 2 1
26.57	45	1 2 2 2
		26.57

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

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

Test 1	Test 2	Test 3
ULAZ:    R	ULAZ:    E	ULAZ:    A
IZLAZ:    Dusko Radovic	IZLAZ:    Dobrica Eric	IZLAZ:    Mika Antic

## 3.2 Sortiranje

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

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

[Rešenje 3.12]

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

Primer 1	Primer 2	Primer 3
INTERAKCIJA PROGRAMA:    Unesite prvu nisku anagram    Unesite drugu nisku rangana    jesu	INTERAKCIJA PROGRAMA:    Unesite prvu nisku anagram    Unesite drugu nisku anagram    nisu	INTERAKCIJA PROGRAMA:    Unesite prvu nisku test    Unesite drugu nisku tset    jesu



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

[Rešenje 3.14]

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

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

[Rešenje 3.15]

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

#### Test 1

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

#### Test 2

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

[Rešenje 3.16]

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

#### Primer 1

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

#### Primer 2

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

[Rešenje 3.17]

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

*Test 1*

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

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

[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

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

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

#### Test 2

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

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

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

[Rešenje 3.19]

**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

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

IZLAZ:
3
```

#### Test 2

```
BIRACKI-SPISAK.TXT
Milan Milicevic

IZLAZ:
1
```

#### Test 3

```
BIRACKI-SPISAK.TXT
[datoteka ne postoji]

IZLAZ:
Problem pri otvaranju
datoteke.
```

[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 decu istog godišta sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci čije se ime zadaje kao prvi argument komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

*Test 1*

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

```
IN.OUT
```

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

```
OUT.TXT
```

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

*Test 2*

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

```
IN.OUT
```

```
Milijana Maric 2009
```

```
OUT.TXT
```

```
Milijana Maric 2009
```

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

*Test 1*

```
NISKE.TXT
```

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

```
IZLAZ:
```

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

[Rešenje 3.22]

**Zadatak 3.23** Napisati program koji simulira rad kase u prodavnici. Kupci prilaze kasi, a prodavac unošenjem bar-koda kupljenog proizvoda dodaje njegovu

### 3 Algoritmi pretrage i sortiranja

---

cenu na ukupan račun. Na kraju, program ispisuje ukupnu vrednost svih proizvoda. Sve artikle, zajedno sa bar-kodovima, proizvođačima i cenama učitati iz datoteke `artikli.txt`. Pretraživanje niza artikala vršiti binarnom pretragom.

#### Primer 1

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

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

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

UKUPNO: 270.00 dinara.

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

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

UKUPNO: 230.00 dinara.

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

Kraj rada kase!
```

[Rešenje 3.23]

**Zadatak 3.24** Napisati program koji iz datoteke `aktivnost.txt` čita podatke o aktivnostima studenata na praktikumima i u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt` upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se

po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po prezimenu opadajuće. U datoteci se nalazi ime, prezime studenta, broj časova na kojima je prisustvovao, kao i ukupan broj urađenih zadataka. Pretpostaviti da studenata neće biti više od 500 i da je za ime studenta dovoljno 20, a za prezime 25 karaktera.

### Test 1

```

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

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

```

```

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

```

```

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

```

[Rešenje 3.24]

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

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

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

### 3 Algoritmi pretrage i sortiranja

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

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

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

[Rešenje 3.25]

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

#### Primer 1

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

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



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

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

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

Test 1

ULAZ:

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

IZLAZ:

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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

#### Primer 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
```

#### Primer 2

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

[Rešenje 3.28]

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

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

#### Primer 1

```
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 standardnog ulaza, unosi se prezime traženog studenta i prikazuje se osoba sa tim prezimenom ili poruka da se nijedan student tako ne preziva. Za pretraživanje koristiti odgovarajuće bibliotečke funkcije. Podaci o studentima čitaju se iz datoteke čije se ime zadaje preko argumenata komandne linije. Za svakog studenta u datoteci postoje ime, prezime i bodovi osvojeni na kolokvijumu. Pretpostaviti da neće biti više od 500 studenata i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

#### Primer 1

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

ULAZNA DATOTEKA (KOLOKVIJUM.TXT):
Aleksandra Cvetic 15
Bojan Golubovic 30
Dragan Markovic 25
Filip Dukic 20
Ivana Stankovic 25
Marija Stankovic 15
Ognjen Peric 20
Uros Milic 10
Andrija Petrovic 0
Anja Ilic 5
Ivana Markovic 5
Lazar Micic 20
Nenad Brankovic 15
```

```
INTERAKCIJA 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  $n$  niski. Maksimalna dužina svake niske je 31 karakter. Sortirati niz  $S$  bibliotečkom funkcijom `qsort` i proveriti da li u njemu ima identičnih niski.

#### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite broj niski: 4
Unesite niske:
prog search sort search
ima
```

#### Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite broj niski: 3
Unesite niske:
test kol ispit
nema
```

#### Primer 3

```
INTERAKCIJA PROGRAMA:
Unesite broj niski: 5
Unesite niske:
a ab abc abcd abcde
nema
```

[Rešenje 3.34]

**Zadatak 3.35** Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr97125`, `mm09001`), ime, prezime i broj poena. Ni ime, ni prezime neće biti duže od 20 karaktera. Napisati

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

#### Test 1

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

STUDENTI.TXT
mr14123 Marko Antic 20
mm13321 Marija Radic 12
ml13011 Ivana Mitrovic 19
ml13066 Pera Simic 15
mv14003 Jovan Jovanovic 17

IZLAZ.TXT
ml13011 Ivana Mitrovic 19
ml13066 Pera Simic 15
mm13321 Marija Radic 12
mr14123 Marko Antic 20
mv14003 Jovan Jovanovic 17

IZLAZ:
mm13321 Marija Radic 12
```

#### Test 2

```
Poziv: ./a.out -p

STUDENTI.TXT
mr14123 Marko Antic 20
mm13321 Marija Radic 12
ml13011 Ivana Mitrovic 19
ml13066 Pera Simic 15
mv14003 Jovan Jovanovic 17

IZLAZ.TXT
mr14123 Marko Antic 20
ml13011 Ivana Mitrovic 19
mv14003 Jovan Jovanovic 17
ml13066 Pera Simic 15
mm13321 Marija Radic 12
```

[Rešenje 3.35]

**Zadatak 3.36** Definisana je struktura:

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

Napisati funkciju koja poredi dva datuma hronološki. Potom, napisati i program koji učitava datume iz datoteke koja se zadaje kao prvi argument komandne linije (ne više od 128 datuma), sortira ih pozivajući funkciju `qsort` iz standardne biblioteke i pozivanjem funkcije `bsearch` iz standardne biblioteke proverava da li datumi učitani sa standardnog ulaza postoje među prethodno unetim datumima. Datumi se učitavaju sve do kraja ulaza.

### 3 Algoritmi pretrage i sortiranja

---

#### Primer 1

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

DATOTEKA.TXT
1.1.2013.
13.12.2016.
11.11.2011.
3.5.2015.
5.2.2009.
```

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

**Zadatak 3.37** Za zadatau celobrojnu matricu dimenzije  $n \times m$  napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrsti. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu.

#### 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
```

#### Test 2

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

[Rešenje 3.37]

**Zadatak 3.38** Za zadatau kvadratnu matricu dimenzije  $n$  napisati funkciju koja sortira kolone matrice opadajuće na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu.

*Primer 1*

```

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

```

*Primer 2*

```

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

```

## 3.4 Rešenja

## Rešenje 3.1

```

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
/* Dokle god je indeks levi levo od indeksa desni */
32 while (levi <= desni) {
    /* Srednji indeks je njihova aritmeticka sredina */
34   srednji = (levi + desni) / 2;
    /* Ako je element sa sredisnjim indeksom veci od x, tada se x
36   mora nalaziti u levoj polovini niza */
    if (x < a[srednji])
38       desni = srednji - 1;
    /* Ako je element sa sredisnjim indeksom manji od x, tada se x
40   mora nalaziti u desnoj polovini niza */
    else if (x > a[srednji])
42       levi = srednji + 1;
    else
44       /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
        x pronadjen na poziciji srednji */
46       return srednji;
}
48 /* Ako element x nije pronadjen, vraca se -1 */
return -1;
50 }

52 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
    pozicije nadjenog elementa ili -1, ako element nije pronadjen */
54 int interpolaciona_pretraga(int a[], int n, int x)
{
56   int levi = 0;
58   int desni = n - 1;
    int srednji;
    /* Dokle god je indeks levi levo od indeksa desni... */
60   while (levi <= desni) {
        /* Ako je trazeni element manji od pocetnog ili veci od
62   poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
        nije u tom delu niza. Ova provera je neophodna, da se ne bi
64   dogodilo da se prilikom izracunavanja indeksa srednji izadje
        izvan opsega indeksa [levi,desni] */
66   if (x < a[levi] || x > a[desni])
        return -1;
68   /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
        a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
70   jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
        desni). Ova provera je neophodna, jer bi se u suprotnom
72   prilikom izracunavanja indeksa srednji pojavilo deljenje
        nulom. */
74   else if (a[levi] == a[desni])
        return levi;
76   /* Racunanje srednjeg indeksa */
    srednji =
78       levi +
        ((double) (x - a[levi]) / (a[desni] - a[levi])) *
80       (desni - levi);
    /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
82   verovatno biti blize trazenoj vrednosti nego da je prosto uvek
```



```

84     uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
    porediti sa pretragom recnika: ako neko trazi rec na slovo 'B
    ',
    sigurno nece da otvori recnik na polovini, vec verovatno negde
86     blize pocetku. */
    /* Ako je element sa indeksom srednji veci od trazenog, tada se
88     trazeni element mora nalaziti u levoj polovini niza */
    if (x < a[srednji])
90         desni = srednji - 1;
    /* Ako je element sa indeksom srednji manji od trazenog, tada se
92     trazeni element mora nalaziti u desnoj polovini niza */
    else if (x > a[srednji])
94         levi = srednji + 1;
    else
96         /* Ako je element sa indeksom srednji jednak trazenom, onda se
            pretraga završava na poziciji srednji */
98         return srednji;
    }
100 /* U slucaju neuspesne pretrage vraca se -1 */
    return -1;
102 }

104 int main(int argc, char **argv)
    {
106     int a[MAX];
    int n, i, x;
108     struct timespec time1, time2, time3, time4, time5, time6;
    FILE *f;

110     /* Provera argumenata komandne linije */
112     if (argc != 3) {
        fprintf(stderr,
114             "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
        ;
        exit(EXIT_FAILURE);
116     }

118     /* Dimenzija niza */
    n = atoi(argv[1]);
120     if (n > MAX || n <= 0) {
        fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
122         exit(EXIT_FAILURE);
    }

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

128     /* Elementi niza se generisu slucajno, tako da je svaki sledeci
        veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
130     pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
        izgledao, seed se postavlja na tekuće vreme u sekundama od Nove
132     godine 1970. random()%100 daje brojeve izmedju 0 i 99 */

```

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

```

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

190     /* Zatvaranje datoteke */
192     fclose(f);

194     return 0;
}

```

### Rešenje 3.2

```

#include <stdio.h>
2
#define MAX 1024
4
int lin_pretraga_rek_sufiks(int a[], int n, int x)
6
{
    int tmp;
    /* Izlaz iz rekurzije */
    if (n <= 0)
10         return -1;
    /* Ako je prvi element trazeni */
12     if (a[0] == x)
        return 0;
14     /* Pretraga ostatka niza */
    tmp = lin_pretraga_rek_sufiks(a + 1, n - 1, x);
16     return tmp < 0 ? tmp : tmp + 1;
}

18
int lin_pretraga_rek_prefiks(int a[], int n, int x)
20
{
    /* Izlaz iz rekurzije */
22     if (n <= 0)
        return -1;
24     /* Ako je poslednji element trazeni */
    if (a[n - 1] == x)
26         return n - 1;
    /* Pretraga ostatka niza */
28     return lin_pretraga_rek_prefiks(a, n - 1, x);
}

30
int bin_pretraga_rek(int a[], int l, int d, int x)
32
{
    int srednji;
34     if (l > d)
        return -1;
36     /* Sredisnja pozicija na kojoj se trazi vrednost x */
    srednji = (l + d) / 2;
38     /* Ako je element na sredisnjoj poziciji trazeni */
}

```

### 3 Algoritmi pretrage i sortiranja

---

```
40     if (a[srednji] == x)
41         return srednji;
42     /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
43        pretrazuje se desna polovina niza */
44     if (a[srednji] < x)
45         return bin_pretraga_rek(a, srednji + 1, d, x);
46     /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
47        pretrazuje se leva polovina niza */
48     else
49         return bin_pretraga_rek(a, l, srednji - 1, x);
50 }

52 int interp_pretraga_rek(int a[], int l, int d, int x)
53 {
54     int p;
55     if (x < a[l] || x > a[d])
56         return -1;
57     if (a[d] == a[l])
58         return l;
59     /* Pozicija na kojoj se trazi vrednost x */
60     p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
61     if (a[p] == x)
62         return p;
63     if (a[p] < x)
64         return interp_pretraga_rek(a, p + 1, d, x);
65     else
66         return interp_pretraga_rek(a, l, p - 1, x);
67 }

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

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

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

86     /* Linearna pretraga */
87     printf("Linearna pretraga\n");
88     indeks = lin_pretraga_rek_sufiks(a, i, x);
89     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_pretraga_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_pretraga_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
/* 0 svakom studentu postoje 3 informacije i one su objedinjene u
9  strukturi kojom se predstavlja svaki student. */
typedef struct {
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

/* Ucitani niz studenata ce biti sortiran rastuce prema indeksu, jer
19  su studenti u datoteci vec sortirani. Iz tog razloga pretraga po
    indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
21  jer nema garancije da postoji uredjenje po prezimenu. */

23  /* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenata
    sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
25  ako element nije pronadjen. */
int binarna_pretraga(Student a[], int n, long x)

```

### 3 Algoritmi pretrage i sortiranja

---

```
27 {
28     int levi = 0;
29     int desni = n - 1;
30     int srednji;
31     /* Dokle god je indeks levi levo od indeksa desni */
32     while (levi <= desni) {
33         /* Racuna se srednja pozicija */
34         srednji = (levi + desni) / 2;
35         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
36            se trazeni indeks mora nalaziti u levoj polovini niza */
37         if (x < a[srednji].indeks)
38             desni = srednji - 1;
39         /* Ako je pak manji od trazenog, tada se on mora nalaziti u
40            desnoj polovini niza */
41         else if (x > a[srednji].indeks)
42             levi = srednji + 1;
43         else
44             /* Ako je jednak trazenom indeksu x, tada je pronadjen student
45                sa trazenom indeksom na poziciji srednji */
46             return srednji;
47     }
48     /* Ako nije pronadjen, vraca se -1 */
49     return -1;
50 }
51
52 /* Linearnom pretragom niza studenata trazi se prezime x */
53 int linearna_pretraga(Student a[], int n, char x[])
54 {
55     int i;
56     for (i = 0; i < n; i++)
57         /* Poredjenje prezimena i-tog studenta i poslatog x */
58         if (strcmp(a[i].prezime, x) == 0)
59             return i;
60     return -1;
61 }
62
63 /* Main funkcija mora imati argumente jer se ime datoteke prosledjuje
64    kao argument komandne linije */
65 int main(int argc, char *argv[])
66 {
67     Student dosije[MAX_STUDENATA];
68     FILE *fin = NULL;
69     int i;
70     int br_studenata = 0;
71     long trazeni_indeks = 0;
72     char trazeno_prezime[MAX_DUZINA];
73
74     /* Provera da li je korisnik prilikom poziva programa prosledio ime
75        datoteke sa informacijama o studentima */
76     if (argc != 2) {
77         fprintf(stderr,
78             "Greska: Program se poziva sa %s ime_datoteke\n",
```

```
79         argv[0]);
80     exit(EXIT_FAILURE);
81 }

82 /* Otvaranje datoteke */
83 fin = fopen(argv[1], "r");
84 if (fin == NULL) {
85     fprintf(stderr,
86         "Neuspješno otvaranje datoteke %s za citanje\n", argv[1]);
87     exit(EXIT_FAILURE);
88 }

89 /* Citanje se vrši sve dok postoji red sa informacijama o studentu
90 */
91 i = 0;
92 while (1) {
93     if (i == MAX_STUDENATA)
94         break;
95     if (fscanf
96         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
97          dosije[i].prezime) != 3)
98         break;
99     i++;
100 }
101 br_studenata = i;

102 /* Nakon citanja, datoteka više nije neophodna i zatvara se. */
103 fclose(fin);

104 /* Unos indeksa koji se binarno traži u nizu */
105 printf("Unesite indeks studenta čije informacije želite: ");
106 scanf("%ld", &trazen_indeks);
107 i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
108 /* Rezultat binarne pretrage */
109 if (i == -1)
110     printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
111 else
112     printf("Indeks: %ld, Ime i prezime: %s %s\n",
113         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

114 /* Unos prezimena koje se linearno traži u nizu */
115 printf("Unesite prezime studenta čije informacije želite: ");
116 scanf("%s", trazeno_prezime);
117 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
118 /* Rezultat linearne pretrage */
119 if (i == -1)
120     printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
121 else
122     printf("Indeks: %ld, Ime i prezime: %s %s\n",
123         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

124 return 0;
```

```
}
```

#### Rešenje 3.4

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

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

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

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



```

48     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
       return i >= 0 ? 1 + i : -1;
50 }

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

64 /* Main funkcija mora imate argumente jer se ime datoteke prosledjuje
   kao argument komandne linije */
66 int main(int argc, char *argv[])
{
68     Student dosije[MAX_STUDENATA];
       FILE *fin = NULL;
70     int i;
       int br_studenata = 0;
72     long trazen_indeks = 0;
       char trazeno_prezime[MAX_DUZINA];

74     /* Provera da li je korisnik prilikom poziva prosledio ime datoteke
       sa informacijama o studentima */
76     if (argc != 2) {
78         fprintf(stderr,
           "Greska: Program se poziva sa %s ime_datoteke\n",
80             argv[0]);
       exit(EXIT_FAILURE);
82     }

84     /* Otvaranje datoteke */
       fin = fopen(argv[1], "r");
86     if (fin == NULL) {
88         fprintf(stderr,
           "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
       exit(EXIT_FAILURE);
90     }

92     /* Citanje se vrši sve dok postoji sledeci red sa informacijama o
       studentu */
94     i = 0;
       while (1) {
96         if (i == MAX_STUDENATA)
           break;
           if (fscanf
98             (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,

```

### 3 Algoritmi pretrage i sortiranja

```
100         dosije[i].prezime) != 3)
101             break;
102         i++;
103     }
104     br_studenata = i;
105
106     /* Nakon citanja datoteka vise nije neophodna i zatvara se.*/
107     fclose(fin);
108
109     /* Unos indeksa koji se binarno trazi u nizu */
110     printf("Unesite indeks studenta cijje informacije zelite: ");
111     scanf("%ld", &trazen_indeks);
112     i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
113                                     trazen_indeks);
114
115     if (i == -1)
116         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
117     else
118         printf("Indeks: %ld, Ime i prezime: %s %s\n",
119               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
120
121     /* Unos prezimena koje se linearno trazi u nizu */
122     printf("Unesite prezime studenta cijje informacije zelite: ");
123     scanf("%s", trazeno_prezime);
124     i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
125                                         trazeno_prezime);
126
127     if (i == -1)
128         printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
129     else
130         printf
131             ("Prvi takav student:\nIndeks: %ld, Ime i prezime: %s %s\n",
132              dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
133
134     return 0;
135 }
```

#### Rešenje 3.5

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 /* Struktura koja opisuje tacku u ravni */
7 typedef struct Tacka {
8     float x;
9     float y;
10 } Tacka;
11
12 /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13    pocetka (0,0) */
14 float rastojanje(Tacka A)
```

```
15 {
16     return sqrt(A.x * A.x + A.y * A.y);
17 }

19 /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u nizu
   zadatih tacaka t dimenzije n */
21 Tacka najbliza_koordinatnom(Tacka t[], int n)
22 {
23     Tacka najbliza;
24     int i;
25     najbliza = t[0];
26     for (i = 1; i < n; i++) {
27         if (rastojanje(t[i]) < rastojanje(najbliza)) {
28             najbliza = t[i];
29         }
30     }
31     return najbliza;
32 }

33 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih tacaka
   t dimenzije n */
35 Tacka najbliza_x_osi(Tacka t[], int n)
36 {
37     Tacka najbliza;
38     int i;
39     najbliza = t[0];
40     for (i = 1; i < n; i++) {
41         if (fabs(t[i].x) < fabs(najbliza.x)) {
42             najbliza = t[i];
43         }
44     }
45     return najbliza;
46 }

47 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
   t dimenzije n */
49 Tacka najbliza_y_osi(Tacka t[], int n)
50 {
51     Tacka najbliza;
52     int i;
53     najbliza = t[0];
54     for (i = 1; i < n; i++) {
55         if (fabs(t[i].y) < fabs(najbliza.y)) {
56             najbliza = t[i];
57         }
58     }
59     return najbliza;
60 }

61 #define MAX 1024
```

### 3 Algoritmi pretrage i sortiranja

---

```
67 int main(int argc, char *argv[])
68 {
69     FILE *ulaz;
70     Tacka tacke[MAX];
71     Tacka najbliza;
72     int i, n;
73
74     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75        ime datoteke sa tackama */
76     if (argc < 2) {
77         fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
79         return EXIT_FAILURE;
80     }
81
82     /* Otvaranje datoteke za citanje */
83     ulaz = fopen(argv[1], "r");
84     if (ulaz == NULL) {
85         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
87         return EXIT_FAILURE;
88     }
89
90     /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
91        tackama; i predstavlja indeks tekuce tacke */
92     i = 0;
93     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94         i++;
95     }
96     n = i;
97
98     /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
99        dodatnih argumenata */
100    if (argc == 2)
101        /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
102        najbliza = najbliza_koordinatnom(tacke, n);
103    /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
104       pitanju opcija -x */
105    else if (strcmp(argv[2], "-x") == 0)
106        /* Racuna se rastojanje u odnosu na x osu */
107        najbliza = najbliza_x_osi(tacke, n);
108    /* Ako je u pitanju opcija -y */
109    else if (strcmp(argv[2], "-y") == 0)
110        /* Racuna se rastojanje u odnosu na y osu */
111        najbliza = najbliza_y_osi(tacke, n);
112    else {
113        /* Ako nije zadata opcija -x ili -y, ispisuje se obavestjenje za
114           korisnika i prekida se izvršavanje programa */
115        fprintf(stderr, "Pogresna opcija\n");
116        return EXIT_FAILURE;
117    }
```

```
119  /* Stampanje koordinata trazene tacke */
    printf("%g %g\n", najbliza.x, najbliza.y);
121
    /* Zatvaranje datoteke */
123  fclose(ulaz);
125
    return 0;
}
```

### Rešenje 3.6

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

4  /* Tacnost */
   #define EPS 0.001

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

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

18     /* Sve dok se ne pronadje trazena vrednost argumenta */
19     while (1) {
20         /* Polovi se interval */
21         s = (l + d) / 2;
22         /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
           tacnosti, prekida se pretraga */
23         if (fabs(cos(s)) < EPS) {
24             break;
25         }
26         /* Ako je nula u levom delu intervala, nastavlja se pretraga na
           [l, s] */
27         if (cos(l) * cos(s) < 0)
28             d = s;
29         else
30             /* Inace, na intervalu [s, d] */
31             l = s;
32     }

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

36     return 0;
}
```

#### Rešenje 3.7

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

#### Rešenje 3.8

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

```

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

```

### Rešenje 3.9

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  unsigned int logaritam_a(unsigned int x)
5  {
6      /* Izlaz iz rekurzije */
7      if (x == 1)

```

### 3 Algoritmi pretrage i sortiranja

---

```

    return 0;
9  /* Rekurzivni korak */
    return 1 + logaritam_a(x >> 1);
11 }

13 unsigned int logaritam_b(unsigned int x)
{
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
        najvece vaznosti, tj. najlevlja. Pretragu se vrsi od pozicije 0
17     do 31 */
    int d = 0, l = sizeof(unsigned int) * 8 - 1;
19     int s;
    /* Sve dok je desna granica pretrage desnije od leve */
21     while (d <= l) {
        /* Racuna se sredisnja pozicija */
        s = (l + d) / 2;
23         /* Proverava se da li je na toj poziciji trazena jedinica */
        if ((1 << s) <= x && (1 << (s + 1)) > x)
25             return s;
        /* Pretraga desne polovine binarnog zapisa */
        if ((1 << s) > x)
27             l = s - 1;
        /* Pretraga leve polovine binarnog zapisa */
        else
31             d = s + 1;
    }
33     return s;
35 }

37 int main()
{
39     unsigned int x;

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

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

49     /* Ispis povratnih vrednosti funkcija */
    printf("%u %u\n", logaritam_a(x), logaritam_b(x));
51
53     return 0;
}
```

#### Rešenje 3.12

---



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

### 3 Algoritmi pretrage i sortiranja

---

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

55     /* Sortiranje */
    selectionSort(a, i);

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

61     return 0;
}
```

#### Rešenje 3.13

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

4  #define MAX_DIM 128

6  /* Funkcija za sortiranje niza karaktera */
void selectionSort(char s[])
8  {
    int i, j, min;
10     char pom;
    for (i = 0; s[i] != '\0'; i++) {
12         min = i;
        for (j = i + 1; s[j] != '\0'; j++)
14             if (s[j] < s[min])
                min = j;
16         if (min != i) {
            pom = s[i];
18             s[i] = s[min];
            s[min] = pom;
20         }
    }
22 }

24 /* Funkcija vraća 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
26 {
    int i;

28     /* Ako dve niske imaju razlicit broj karaktera onda one nisu
30     anagrami */
    if (strlen(s) != strlen(t))
32         return 0;

34     /* Sortiramo niske */
    selectionSort(s);
36     selectionSort(t);
}
```

```

38  /* Dve sortirane niske su anagrami ako i samo ako su jednake */
    for (i = 0; s[i] != '\0'; i++)
40      if (s[i] != t[i])
          return 0;
42      return 1;
    }

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

48
    /* Ucitavanje niski sa ulaza */
50    printf("Unesite prvu nisku: ");
    scanf("%s", s);
52    printf("Unesite drugu nisku: ");
    scanf("%s", t);

54
    /* Poziv funkcije */
56    if (anagrami(s, t))
        printf("jesu\n");
58    else
        printf("nisu\n");

60
    return 0;
62 }

```

### Rešenje 3.14

```

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

3
    /* Funkcija za sortiranje niza */
5  void selectionSort(int s[], int n)
    {
7      int i, j, min;
      char pom;
9      for (i = 0; i < n; i++) {
          min = i;
11         for (j = i + 1; j < n; j++)
             if (s[j] < s[min])
13                 min = j;
             if (min != i) {
15                 pom = s[i];
                 s[i] = s[min];
17                 s[min] = pom;
             }
19     }
    }

21
    /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
23     najviše puta pojavio u tom nizu */

```

```
int najvise_puta(int a[], int n)
{
    int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
    /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
    for (i = 0; i < n; i = j) {
        br_pojava = 1;
        for (j = i + 1; j < n && a[i] == a[j]; j++)
            br_pojava++;
        /* Ispitivanje da li se do tog trenutka i-ti element pojavio
           najvise puta u nizu */
        if (br_pojava > max_br_pojava) {
            max_br_pojava = br_pojava;
            i_max_pojava = i;
        }
    }
    /* Vraca se element koji se najvise puta pojavio u nizu */
    return a[i_max_pojava];
}

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

    /* Ucitavanje elemenata niza sve do kraja ulaza */
    i = 0;
    while (scanf("%d", &a[i]) != EOF)
        i++;

    /* Niz se sortira */
    selectionSort(a, i);

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

    return 0;
}
```

#### Rešenje 3.15

```
1 #include<stdio.h>
2 #define MAX_DIM 256
3
4 /* Funkcija za sortiranje niza */
5 void selectionSort(int a[], int n)
6 {
7     int i, j, min, pom;
8     for (i = 0; i < n - 1; i++) {
9         min = i;
10        for (j = i + 1; j < n; j++)
11            if (a[j] < a[min])
12                min = j;
```

```
13     if (min != i) {
14         pom = a[i];
15         a[i] = a[min];
16         a[min] = pom;
17     }
18 }
19 }

21 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
22    u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
23    poretku */
24 int binarna_pretraga(int a[], int n, int x)
25 {
26     int levi = 0, desni = n - 1, srednji;
27
28     while (levi <= desni) {
29         srednji = (levi + desni) / 2;
30         if (a[srednji] == x)
31             return 1;
32         else if (a[srednji] > x)
33             desni = srednji - 1;
34         else if (a[srednji] < x)
35             levi = srednji + 1;
36     }
37     return 0;
38 }
39
40 int main()
41 {
42     int a[MAX_DIM], n = 0, zbir, i;
43
44     /* Ucitava se trazeni zbir */
45     printf("Unesite trazeni zbir: ");
46     scanf("%d", &zbir);
47
48     /* Ucitavaju se elementi niza sve do kraja ulaza */
49     i = 0;
50     printf("Unesite elemente niza: ");
51     while (scanf("%d", &a[i]) != EOF)
52         i++;
53     n = i;
54
55     /* Sortira se niz */
56     selectionSort(a, n);
57
58     for (i = 0; i < n; i++)
59         /* Za i-ti element niza binarno se pretrazuje da li se u ostatku
60            niza nalazi element koji sabran sa njim ima ucitanu vrednost
61            zbira */
62         if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
63             printf("da\n");
64             return 0;
65         }
```

### 3 Algoritmi pretrage i sortiranja

---

```
65     }
66     printf("ne\n");
67
68     return 0;
69 }
```

#### Rešenje 3.16

```
/* 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 insertion sort(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
```

```
/* Datoteka sort.c */
2
3  #include "sort.h"
4
5  #define MAX 1000000
6
7  /* Funkcija sortira niz celih brojeva metodom sortiranja izborom.
8   Ideja algoritma je sledeca: U svakoj iteraciji pronalazi se
9   najmanji element i premesta se na pocetak niza. Dakle, u prvoj
10  iteraciji, pronalazi se najmanji element, i dovodi na nultu mesto
11  u nizu. U i-toj iteraciji najmanjih i elemenata su vec na svojim
12  pozicijama, pa se od i+1 do n-1 elementa trazi najmanji, koji se
13  dovodi na i+1 poziciju. */
14 void selectionsort(int a[], int n)
15 {
16     int i, j;
17     int min;
18     int pom;
19
20     /* U svakoj iteraciji ove petlje pronalazi se najmanji element
21      medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
22      poziciju i, dok se element na poziciji i premesta na poziciju
```

```

    min, na kojoj se nalazio najmanji od gore navedenih elemenata.
    */
24 for (i = 0; i < n - 1; i++) {
    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
26    najmanji od elemenata a[i],...,a[n-1]. */
    min = i;
28    for (j = i + 1; j < n; j++)
        if (a[j] < a[min])
30        min = j;

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 /* Funkcija sortira niz celih brojeva metodom sortiranja umetanjem.
    Ideja algoritma je sledeca: neka je na pocetku i-te iteracije niz
44 prvih i elemenata (a[0],a[1],...,a[i-1]) sortirano. U i-toj
    iteraciji treba element a[i] umetnuti na pravu poziciju medju
46 prvih i elemenata tako da se dobije niz duzine i+1 koji je
    sortiran. Ovo se radi tako sto se i-ti element najpre uporedi sa
48 njegovim prvim levim susedom (a[i-1]). Ako je a[i] vece, tada je
    on vec na pravom mestu, i niz a[0],a[1],...,a[i] je sortiran, pa
50 se moze preci na sledecu iteraciju. Ako je a[i-1] vece, tada se
    zamenjuju a[i] i a[i-1], a zatim se proverava da li je potrebno
52 dalje potiskivanje elementa u levo, poredeci ga sa njegovim novim
    levim susedom. Ovim uzastopnim premestanjem se a[i] umece na pravo
54 mesto u nizu. */
void insertionsort(int a[], int n)
56 {
    int i, j;

58    /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
    sortiran */
60    for (i = 1; i < n; i++) {

62        /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
        potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...a[i]
64        bude sortiran. Indeks j je trenutna pozicija na kojoj se
        element koji se umece nalazi. Petlja se zavrшава ili kada
66        element dodje do levog kraja (j==0) ili kada se naidje na
        element a[j-1] koji je manji od a[j]. */
68        for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
90            int temp = a[j];
            a[j] = a[j - 1];
92            a[j - 1] = temp;
        }
    }
}

```

### 3 Algoritmi pretrage i sortiranja

---

```
74     }
75 }
76
77 /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
78 algoritma je sledeca: prolazi se kroz niz redom poredeci susedne
79 elemente, i pri tom ih zamenjujuci ako su u pogresnom poretku.
80 Ovim se najveći element poput mehurica istiskuje na "povrsinu",
81 tj. na krajnju desnu poziciju. Nakon toga je potrebno ovaj
82 postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih n-1
83 elemenata niza bez poslednjeg koji je postavljen na pravu
84 poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
85 kracim prefiksima niza, cime se jedan po jedan istiskuju
86 elementi na svoje prave pozicije. */
87 void bubblesort(int a[], int n)
88 {
89     int i, j;
90     int ind;
91
92     for (i = n, ind = 1; i > 1 && ind; i--)
93     {
94         /* Poput "mehurica" potiskuje se najveći element medju elementima
95         od a[0] do a[i-1] na poziciju i-1 upoređujući susedne
96         elemente niza i potiskujući veci u desno */
97         for (j = 0, ind = 0; j < i - 1; j++)
98             if (a[j] > a[j + 1]) {
99                 int temp = a[j];
100                 a[j] = a[j + 1];
101                 a[j + 1] = temp;
102
103                 /* Promenljiva ind registruje da je bilo premestanja. Samo u
104                 tom slucaju ima smisla ici na sledecu iteraciju, jer ako
105                 nije bilo premestanja, znaci da su svi elementi vec u
106                 dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
107                 niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
108                 ispravno, ali manje efikasano, jer bi se cesto nepotrebno
109                 vrsila mnoga upoređivanja, kada je vec jasno da je
110                 sortiranje zavrшено. */
111                 ind = 1;
112             }
113     }
114
115     /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
116     dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
117     sastoji u tome da se kroz algoritam umetanja prolazi vise puta; u
118     prvom prolazu, umesto koraka 1 uzima se neki korak h koji je manji
119     od n (sto omogucuje razmenu udaljenih elemenata) i tako se dobija
120     h-sortiran niz, tj. niz u kome su elementi na rastojanju h
121     sortirani, mada susedni elementi to ne moraju biti. U drugom
122     prolazu kroz isti algoritam sprovodi se isti postupak ali za manji
123     korak h. Sa prolazima se nastavlja sve do koraka h = 1, u kome se
124     dobija potpuno sortirani niz. Izbor pocetne vrednosti za h, i
125     nacina njegovog smanjivanja menja u nekim slucajevima brzinu
```



```

126     algoritma, ali bilo koja vrednost ce rezultovati ispravnim
128     sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
        vrednost 1. */
void shellsort(int a[], int n)
130 {
    int h = n / 2, i, j;
132     while (h > 0) {
        /* Insertion sort sa korakom h */
134         for (i = h; i < n; i++) {
            int temp = a[i];
136             j = i;
            while (j >= h && a[j - h] > temp) {
138                 a[j] = a[j - h];
                j -= h;
140             }
            a[j] = temp;
142         }
        h = h / 2;
144     }
}

146 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem. Sortiranje
148     se vrši od elementa na poziciji l do onog na poziciji d. Na
        pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a d je
150     jednako poslednjem validnom indeksu u nizu. Funkcija niz podeli na
        dve polovine, levu i desnu, koje zatim rekurzivno sortira. Od ova
152     dva sortirana podniza, sortiran niz se dobija ucesljavanjem, tj.
        istovremenim prolaskom kroz oba niza i izborom trenutnog manjeg
154     elementa koji se smesta u pomocni niz. Na kraju algoritma,
        sortirani elementi su u pomocnom nizu, koji se kopira u originalni
156     niz. */
void mergesort(int a[], int l, int d)
158 {
    int s;
160     static int b[MAX];          /* Pomocni niz */
    int i, j, k;

162     /* Izlaz iz rekurzije */
164     if (l >= d)
        return;

166     /* Odredjivanje sredisnjeg indeksa */
168     s = (l + d) / 2;

170     /* Rekurzivni pozivi */
    mergesort(a, l, s);
172     mergesort(a, s + 1, d);

174     /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
        niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
176     prolazi kroz pomocni niz b[] */
    i = l;

```

### 3 Algoritmi pretrage i sortiranja

---

```
178     j = s + 1;
179     k = 0;
180
181     /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
182     while (i <= s && j <= d) {
183         if (a[i] < a[j])
184             b[k++] = a[i++];
185         else
186             b[k++] = a[j++];
187     }
188
189     /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive j
190        iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
191        polovine niza */
192     while (i <= s)
193         b[k++] = a[i++];
194
195     /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive i
196        iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
197        polovine niza */
198     while (j <= d)
199         b[k++] = a[j++];
200
201     /* Prepisuje se "ucesljani" niz u originalni niz */
202     for (k = 0, i = 1; i <= d; i++, k++)
203         a[i] = b[k];
204 }
205
206 /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
207 void swap(int a[], int i, int j)
208 {
209     int tmp = a[i];
210     a[i] = a[j];
211     a[j] = tmp;
212 }
213
214 /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r. Njena
215    ideja sortiranja je izbor jednog elementa niza, koji se naziva
216    pivot, i koji se dovodi na svoje mesto. Posle ovog koraka, svi
217    elementi levo od njega bice manji, a svi desno bice veci od njega.
218    Kako je pivot doveden na svoje mesto, da bi niz bio kompletno
219    sortirao, potrebno je sortirati elemente levo (manje) od njega, i
220    elemente desno (vece). Kako su dimenzije ova dva podniza manje od
221    dimenzije pocetnog niza koji je trebalo sortirati, ovaj deo moze
222    se uraditi rekurzivno. */
223 void quicksort(int a[], int l, int r)
224 {
225     int i, pivot_position;
226
227     /* Izlaz iz rekurziije -- prazan niz */
228     if (l >= r)
```

```

230     return;

232

233     /* Particionisanje niza. Svi elementi na pozicijama levo od
234        pivot_position (izuzev same pozicije l) su strogo manji od
235        pivota. Kada se pronadje neki element manji od pivota, uvecava
236        se promenljiva pivot_position i na tu poziciju se premesta
237        nadjeni element. Na kraju ce pivot_position zaista biti pozicija
238        na koju treba smestiti pivot, jer ce svi elementi levo od te
239        pozicije biti manji a desno biti veci ili jednaki od pivota. */
240     pivot_position = l;
241     for (i = l + 1; i <= r; i++)
242         if (a[i] < a[l])
243             swap(a, ++pivot_position, i);

244     /* Postavljanje pivota na svoje mesto */
245     swap(a, l, pivot_position);

246     /* Rekurzivno sortiranje elementa manjih od pivota */
247     quicksort(a, l, pivot_position - 1);
248     /* Rekurzivno sortiranje elementa vecih od pivota */
249     quicksort(a, pivot_position + 1, r);
250 }
251

```

```

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

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

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

26     /* Dimenzija niza koji se sortira */
27     int dimenzija;
28

```

```
int i;
30 int niz[MAX];

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

38
/* Ocitavanje opcija i argumenata prilikom poziva programa */
40 for (i = 1; i < argc; i++) {
    /* Ako je u pitanju opcija... */
42     if (argv[i][0] == '-') {
        switch (argv[i][1]) {
44             case 'i':
                tip_sortiranja = 1;
46                 break;
                case 'b':
48                 tip_sortiranja = 2;
                    break;
50                 case 's':
                    tip_sortiranja = 3;
52                     break;
                    case 'm':
54                     tip_sortiranja = 4;
                        break;
56                     case 'q':
                        tip_sortiranja = 5;
58                         break;
                        case 'r':
60                         tip_niza = 1;
                            break;
62                             case 'o':
                                tip_niza = 2;
64                                 break;
                                default:
66                                 printf("Pogresna opcija -%c\n", argv[i][1]);
                                    return 1;
68                                     break;
                                    }
70         }
        /* Ako je u pitanju argument, onda je to duzina niza koji treba
72         da se sortira */
        else {
74             dimenzija = atoi(argv[i]);
            if (dimenzija <= 0 || dimenzija > MAX) {
76                 fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
                    exit(EXIT_FAILURE);
78             }
        }
80     }
```

```

82  /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
83     niza dobijenom iz komandne linije. srandom() funkcija
84     obezbedjuje novi seed za pozivanje random funkcije, i kako
85     generisani niz ne bi uvek bio isti seed je postavljen na tekuce
86     vreme u sekundama od Nove godine 1970. random()%100 daje brojeve
87     izmedju 0 i 99 */
88  srandom(time(NULL));
89  if (tip_niza == 0)
90      for (i = 0; i < dimenzija; i++)
91          niz[i] = random();
92  else if (tip_niza == 1)
93      for (i = 0; i < dimenzija; i++)
94          niz[i] = i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
95  else
96      for (i = 0; i < dimenzija; i++)
97          niz[i] = i == 0 ? random() % 100 : niz[i - 1] - random() % 100;
98
99  /* Ispisivanje elemenata niza */
100  /*****
101     Ovaj deo je iskomentaran jer sledeci ispis ne treba da se nadje
102     na standardnom izlazu. Njegova svrha je samo bila provera da li
103     je niz generisan u skladu sa opcijama komandne linije.
104
105     printf("Niz koji sortiramo je:\n");
106     for (i = 0; i < dimenzija; i++)
107         printf("%d\n", niz[i]);
108  *****/
109
110  /* Sortiranje niza na odgovarajuci nacin */
111  if (tip_sortiranja == 0)
112      selectionsort(niz, dimenzija);
113  else if (tip_sortiranja == 1)
114      insertion sort(niz, dimenzija);
115  else if (tip_sortiranja == 2)
116      bubblesort(niz, dimenzija);
117  else if (tip_sortiranja == 3)
118      shellsort(niz, dimenzija);
119  else if (tip_sortiranja == 4)
120      mergesort(niz, 0, dimenzija - 1);
121  else
122      quicksort(niz, 0, dimenzija - 1);
123
124  /* Ispis elemenata niza */
125  /*****
126     Ovaj deo je iskomentaran jer vreme potrebno za njegovo
127     izvršavanje ne bi trebalo da bude ukljuceno u vreme izmereno
128     programom time. Takodje, kako je svrha ovog programa da prikaze
129     vremena razlicitih algoritama sortiranja, dimenzije nizova ce
130     biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
131     od toliko elemenata. Ovaj deo je koriscen u razvoju programa
132  *****/

```

### 3 Algoritmi pretrage i sortiranja

```
        zarad testiranja korektnosti.

134     printf("Sortiran niz je:\n");
136     for (i = 0; i < dimenzija; i++)
        printf("%d\n", niz[i]);
138     *****/
140     return 0;
}
```

#### Rešenje 3.17

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

4 int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
           int dim3)
6 {
    int i = 0, j = 0, k = 0;
    8 /* U slucaju da je dimenzija treceg niza manja od neophodne,
        funkcija vraca -1 */
    10 if (dim3 < dim1 + dim2)
        return -1;
    12
    /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog
        od njih */
    14 while (i < dim1 && j < dim2) {
        16 if (niz1[i] < niz2[j])
            niz3[k++] = niz1[i++];
        18 else
            niz3[k++] = niz2[j++];
    20 }
    /* Ostatak prvog niza prepisujemo u treci */
    22 while (i < dim1)
        niz3[k++] = niz1[i++];
    24
    /* Ostatak drugog niza prepisujemo u treci */
    26 while (j < dim2)
        niz3[k++] = niz2[j++];
    28 return dim1 + dim2;
}

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

    36 /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
        Pretpostavka je da na ulazu nece biti vise od MAX_DIM elemenata
        */
    38 printf("Unesite elemente prvog niza: ");
```

```

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");
61
62     return 0;
63 }

```

### Rešenje 3.18

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main(int argc, char *argv[])
6  {
7      FILE *fin1 = NULL, *fin2 = NULL;
8      FILE *fout = NULL;
9      char ime1[11], ime2[11];
10     char prezime1[16], prezime2[16];
11     int kraj1 = 0, kraj2 = 0;
12
13     /* Ako nema dovoljno argumenata komandne linije */
14     if (argc < 3) {
15         fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0])
16         ;
17         exit(EXIT_FAILURE);
18     }
19
20     /* Otvaranje datoteke zadate prvim argumentom komandne linije */
21     fin1 = fopen(argv[1], "r");
22     if (fin1 == NULL) {
23         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
24         exit(EXIT_FAILURE);
25     }

```

```
24  }

26  /* Otvaranje datoteke zadate drugim argumentom komandne linije */
   fin2 = fopen(argv[2], "r");
28  if (fin2 == NULL) {
   fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
30  exit(EXIT_FAILURE);
   }

32

34  /* Otvaranje datoteke za upis rezultata */
   fout = fopen("ceo-tok.txt", "w");
36  if (fout == NULL) {
   fprintf(stderr,
38       "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
   exit(EXIT_FAILURE);
   }

40

42  /* Citanje narednog studenta iz prve datoteke */
   if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
       kraj1 = 1;

44

46  /* Citanje narednog studenta iz druge datoteke */
   if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
       kraj2 = 1;

48

50  /* Sve dok nije dostignut kraj neke datoteke */
   while (!kraj1 && !kraj2) {
       if (strcmp(ime1, ime2) < 0) {
52           /* Ime i prezime iz prve datoteke je leksikografski ranije, i
              biva upisano u izlaznu datoteku */
           fprintf(fout, "%s %s\n", ime1, prezime1);
           /* Citanje narednog studenta iz prve datoteke */
54           if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
               kraj1 = 1;
56       } else {
           /* Ime i prezime iz druge datoteke je leksikografski ranije, i
              biva upisano u izlaznu datoteku */
           fprintf(fout, "%s %s\n", ime2, prezime2);
           /* Citanje narednog studenta iz druge datoteke */
58           if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
               kraj2 = 1;
60       }
   }

62

64  /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
   druge datoteke, onda ima jos studenata u prvoj datoteci, koje
   treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
   */
66

68  while (!kraj1) {
       fprintf(fout, "%s %s\n", ime1, prezime1);
       if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
70           kraj1 = 1;
72
74 }
```



```

76     }

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

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

92     return 0;
   }

```

### Rešenje 3.19

```

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

5

   #define MAX_BR_TACAKA 128

7  /* Struktura koja reprezentuje koordinate tacke */
9  typedef struct Tacka {
       int x;
11      int y;
   } Tacka;

13

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

21  /* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
       pocetka */
23  void sortiraj_po_rastojanju(Tacka t[], int n)
       {
25         int min, i, j;
           Tacka tmp;

27         for (i = 0; i < n - 1; i++) {
29             min = i;
               for (j = i + 1; j < n; j++) {

```

### 3 Algoritmi pretrage i sortiranja

---

```
31         if (rastojanje(t[j]) < rastojanje(t[min])) {
32             min = j;
33         }
34     }
35     if (min != i) {
36         tmp = t[i];
37         t[i] = t[min];
38         t[min] = tmp;
39     }
40 }
41 }

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     for (i = 0; i < n - 1; i++) {
49         min = i;
50         for (j = i + 1; j < n; j++) {
51             if (abs(t[j].x) < abs(t[min].x)) {
52                 min = j;
53             }
54         }
55         if (min != i) {
56             tmp = t[i];
57             t[i] = t[min];
58             t[min] = tmp;
59         }
60     }
61 }

63 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
64 void sortiraj_po_y(Tacka t[], int n)
65 {
66     int min, i, j;
67     Tacka tmp;

68     for (i = 0; i < n - 1; i++) {
69         min = i;
70         for (j = i + 1; j < n; j++) {
71             if (abs(t[j].y) < abs(t[min].y)) {
72                 min = j;
73             }
74         }
75         if (min != i) {
76             tmp = t[i];
77             t[i] = t[min];
78             t[min] = tmp;
79         }
80     }
81 }
```

```
83 }
85 int main(int argc, char *argv[])
86 {
87     FILE *ulaz;
88     FILE *izlaz;
89     Tacka tacke[MAX_BR_TACAKA];
90     int i, n;
91
92     /* Proveravanje broja argumenata komandne linije: ocekuje se ime
93        izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
94        datoteke, tj. 4 argumenta */
95     if (argc != 4) {
96         fprintf(stderr,
97             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
98         return 0;
99     }
100
101     /* Otvaranje datoteke u kojoj su zadate tacke */
102     ulaz = fopen(argv[2], "r");
103     if (ulaz == NULL) {
104         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
105             argv[2]);
106         return 0;
107     }
108
109     /* Otvaranje datoteke u koju treba upisati rezultat */
110     izlaz = fopen(argv[3], "w");
111     if (izlaz == NULL) {
112         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
113             argv[3]);
114         return 0;
115     }
116
117     /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
118        koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
119        brojacem i. */
120     i = 0;
121     while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
122         i++;
123     }
124
125     /* Ukupan broj procitanih tacaka */
126     n = i;
127
128     /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
129        "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
130        (karakter -), a karakter argv[1][1] odredjuje kriterijum
131        sortiranja */
132     switch (argv[1][1]) {
133     case 'x':
134         /* Sortiranje po vrednosti x koordinate */
```

### 3 Algoritmi pretrage i sortiranja

```
135     sortiraj_po_x(tacke, n);
136     break;
137 case 'y':
138     /* Sortiranje po vrednosti y koordinate */
139     sortiraj_po_y(tacke, n);
140     break;
141 case 'o':
142     /* Sortiranje po udaljenosti od koordinatnog pocetka */
143     sortiraj_po_rastojanju(tacke, n);
144     break;
145 }

147 /* Upisivanje dobijenog niza u izlaznu datoteku */
148 for (i = 0; i < n; i++) {
149     fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
150 }

151 /* Zatvaranje otvorenih datoteka */
152 fclose(ulaz);
153 fclose(izlaz);
154
155 return 0;
156 }
157 }
```

#### Rešenje 3.20

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

4
#define MAX 1000
6 #define MAX_DUZINA 16

8 /* Struktura koja reprezentuje jednog gradjanina */
typedef struct gr {
10     char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
12 } Gradjanin;

14 /* Funkcija sortira niz gradjana rastuce po imenima */
void sort_ime(Gradjanin a[], int n)
16 {
    int i, j;
18     int min;
    Gradjanin pom;

20     for (i = 0; i < n - 1; i++) {
22         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
            najmanji od elemenata a[i].ime,...,a[n-1].ime. */
24         min = i;
        for (j = i + 1; j < n; j++)
```

```

26     if (strcmp(a[j].ime, a[min].ime) < 0)
27         min = j;
28     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
29        su (i) i min razliciti, inace je nepotrebno. */
30     if (min != i) {
31         pom = a[i];
32         a[i] = a[min];
33         a[min] = pom;
34     }
35 }
36 }

38 /* Funkcija sortira niz gradjana rastuce po prezimenima */
39 void sort_prezime(Gradjanin a[], int n)
40 {
41     int i, j;
42     int min;
43     Gradjanin pom;
44
45     for (i = 0; i < n - 1; i++) {
46         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
47            najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
48         min = i;
49         for (j = i + 1; j < n; j++)
50             if (strcmp(a[j].prezime, a[min].prezime) < 0)
51                 min = j;
52         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
53            su (i) i min razliciti, inace je nepotrebno. */
54         if (min != i) {
55             pom = a[i];
56             a[i] = a[min];
57             a[min] = pom;
58         }
59     }
60 }

62 /* Pretraga niza Gradjana */
63 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64 {
65     int i;
66     for (i = 0; i < n; i++)
67         if (strcmp(a[i].ime, x->ime) == 0
68             && strcmp(a[i].prezime, x->prezime) == 0)
69             return i;
70     return -1;
71 }

72
74 int main()
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;

```

### 3 Algoritmi pretrage i sortiranja

---

```
78  int i, n;
    FILE *fp = NULL;

80

    /* Otvaranje datoteke */
82  if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
        fprintf(stderr,
84         "Neuspesno otvaranje datoteke biracki-spisak.txt.\n");
        exit(EXIT_FAILURE);
86  }

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

94

    /* Zatvaranje datoteke */
96  fclose(fp);

98  sort_ime(spisak1, n);

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

104  printf("Biracki spisak [uredjen prema imenima]:\n");
    for(i=0; i<n; i++)
106         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
    *****/

108  sort_prezime(spisak2, n);

110

112  /*****
    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih nizova. Koriscen je samo u fazi testiranja programa.

114  printf("Biracki spisak [uredjen prema prezimenima]:\n");
    for(i=0; i<n; i++)
116         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
    *****/

120  /* Linearno pretrazivanje nizova */
    for (i = 0; i < n; i++)
122         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
            isti_rbr++;

124

    /* Alternativno (efikasnije) resenje */
126  /*****
    for(i=0; i<n ;i++)
128         if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
                strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
```

```

130     isti_rbr++;
131     *****/
132
133     /* Ispis rezultata */
134     printf("%d\n", isti_rbr);
135
136     exit(EXIT_SUCCESS);
137 }

```

### Rešenje 3.22

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  #define MAX_BR_RECII 128
6  #define MAX_DUZINA_RECII 32
7
8  /* Funkcija koja izracunava broj suglasnika u reci */
9  int broj_suglasnika(char s[])
10 {
11     char c;
12     int i;
13     int suglasnici = 0;
14     /* Prolazi karakter po karakter kroz zadatu nisku */
15     for (i = 0; s[i]; i++) {
16         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
17            pokriven slucaj i malih i velikih suglasnika. */
18         if (isalpha(s[i])) {
19             c = toupper(s[i]);
20             /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika.
21                */
22             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
23                 suglasnici++;
24         }
25     }
26     /* Vraca se izracunata vrednost */
27     return suglasnici;
28 }
29
30 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
31    duzini reci se mora proslediti zbog pravilnog upravljanja
32    memorijom */
33 void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)
34 {
35     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
36         duzina_j, duzina_min;
37     char tmp[MAX_DUZINA_RECII];
38     for (i = 0; i < n - 1; i++) {
39         min = i;
40         for (j = i; j < n; j++) {

```

### 3 Algoritmi pretrage i sortiranja

---

```
41      /* Prvo se upoređuje broj suglasnika */
    broj_suglasnika_j = broj_suglasnika(reci[j]);
    broj_suglasnika_min = broj_suglasnika(reci[min]);
43    if (broj_suglasnika_j < broj_suglasnika_min)
        min = j;
45    else if (broj_suglasnika_j == broj_suglasnika_min) {
        /* Zatim, recima koje imaju isti broj suglasnika upoređuju
47        se duzine */
        duzina_j = strlen(reci[j]);
49        duzina_min = strlen(reci[min]);

51        if (duzina_j < duzina_min)
            min = j;
53        else
            /* Ako reci imaju i isti broj suglasnika i iste duzine,
55            upoređuju se leksikografski */
            if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
57                min = j;
    }
59 }
    if (min != i) {
61        strcpy(tmp, reci[min]);
        strcpy(reci[min], reci[i]);
63        strcpy(reci[i], tmp);
    }
65 }
}

67 int main()
69 {
    FILE *ulaz;
71    int i = 0, n;

73    /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
        a drugi maksimalnu duzinu pojedinačne reci */
75    char reci[MAX_BR_RECII][MAX_DUZINA_RECII];

77    /* Otvaranje datoteke niske.txt za citanje */
    ulaz = fopen("niske.txt", "r");
79    if (ulaz == NULL) {
        fprintf(stderr,
81            "Greska prilikom otvaranja datoteke niske.txt!\n");
        return 0;
83    }

85    /* Sve dok se moze procitati sledeca rec */
    while (fscanf(ulaz, "%s", reci[i]) != EOF) {
87        /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
            prekida se ucitavanje */
89        if (i == MAX_BR_RECII)
            break;
91        /* Priprema brojaca za narednu iteraciju */
```



```

    i++;
93 }

95 /* n je duzina niza reci i predstavlja poslednju vrednost
    koriscenog brojaca */
97 n = i;
    /* Poziv funkcije za sortiranje reci */
99 sortiraj_reci(reci, n);

101 /* Ispis sortiranog niza reci */
    for (i = 0; i < n; i++) {
103     printf("%s ", reci[i]);
    }
105 printf("\n");

107 /* Zatvaranje datoteke */
    fclose(ulaz);
109
    return 0;
111 }

```

### Rešenje 3.23

```

#include <stdio.h>
2  #include <stdlib.h>
    #include <string.h>
4
    #define MAX_ARTIKALA 100000
6
    /* Struktura koja predstavlja jedan artikal */
8  typedef struct art {
        long kod;
10     char naziv[20];
        char proizvodjac[20];
12     float cena;
    } Artikal;
14

    /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
16     trazenim bar kodom */
    int binarna_pretraga(Artikal a[], int n, long x)
18 {
        int levi = 0;
20     int desni = n - 1;

22     /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
24         /* Racuna se sredisnji indeks */
        int srednji = (levi + desni) / 2;
26         /* Ako je sredisnji element veci od trazenog, tada se trazeni
            mora nalaziti u levoj polovini niza */
28         if (x < a[srednji].kod)

```

### 3 Algoritmi pretrage i sortiranja

---

```
        desni = srednji - 1;
30    /* Ako je sredisnji element manji od trazenog, tada se trazen
        mora nalaziti u desnoj polovini niza */
32    else if (x > a[srednji].kod)
        levi = srednji + 1;
34    else
        /* Ako je sredisnji element jednak trazenom, tada je artikal sa
36        bar kodom x pronadjen na poziciji srednji */
        return srednji;
38    }
    /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
40    return -1;
}

42    /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44    void selection_sort(Artikal a[], int n)
    {
46        int i, j;
48        int min;
49        Artikal pom;

50        for (i = 0; i < n - 1; i++) {
51            min = i;
52            for (j = i + 1; j < n; j++)
53                if (a[j].kod < a[min].kod)
54                    min = j;
55            if (min != i) {
56                pom = a[i];
57                a[i] = a[min];
58                a[min] = pom;
59            }
60        }
61    }

62    int main()
63    {
64        Artikal asortiman[MAX_ARTIKALA];
65        long kod;
66        int i, n;
67        float racun;

70        FILE *fp = NULL;

72        /* Otvaranje datoteke */
73        if ((fp = fopen("artikli.txt", "r")) == NULL) {
74            fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
75            exit(EXIT_FAILURE);
76        }

78        /* Ucitavanje artikala */
79        i = 0;
80        while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
```

```

82         asortiman[i].naziv, asortiman[i].proizvodjac,
           &asortiman[i].cena) == 4)
83     i++;
84
85     /* Zatvaranje datoteke */
86     fclose(fp);
87
88     n = i;
89
90     /* Sortira se celokupan asortiman prodavnice prema kodovima jer ce
91     pri kucanju racuna prodavac unositi kod artikla. Prilikom
92     kucanja svakog racuna pretrazuje se asortiman, da bi se utvrdila
93     cena artikla. Kucanje racuna obuhvata vise pretraga asortimana i
94     cilj je da ta operacija bude sto efikasnija. Zato se koristi
95     algoritam binarne pretrage prilikom pretrazivanja po kodu
96     artikla. Iz tog razloga, potrebno je da asortiman bude sortiran
97     po kodovima i to ce biti uradjeno primenom selection sort
98     algoritma. Sortiranje se vrši samo jednom na pocetku, ali se
99     zato posle artikli mogu brzo pretrazivati prilikom kucanja
100    proizvoljno puno racuna. Vreme koje se utrosi na sortiranje na
101    pocetku izvršavanja programa, kasnije se isplati jer se za
102    brojna trazanja artikla umesto linearne moze koristiti
103    efikasnija binarna pretraga. */
104    selection_sort(asortiman, n);
105
106    /* Ispis stanja u prodavnici */
107    printf
108    ("Asortiman:\nKOD          Naziv artikla      Ime
109     proizvodjaca      Cena\n");
110    for (i = 0; i < n; i++)
111        printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
112            asortiman[i].naziv, asortiman[i].proizvodjac,
113            asortiman[i].cena);
114
115    kod = 0;
116    while (1) {
117        printf("-----\n");
118        printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
119        printf("- Za nov racun unesite kod artikla!\n\n");
120        /* Unos bar koda provog artikla sledeceg kupca */
121        if (scanf("%ld", &kod) == EOF)
122            break;
123        /* Trenutni racun novog kupca */
124        racun = 0;
125        /* Za sve artikle trenutnog kupca */
126        while (1) {
127            /* Vrši se njihov pronalazak u nizu */
128            if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
129                printf("\tGRESKA: Ne postoji proizvod sa trazanim kodom!\n");
130            } else {
131                printf("\tTrazili ste:\t%s %s %12.2f\n",
132                    asortiman[i].naziv, asortiman[i].proizvodjac,

```

### 3 Algoritmi pretrage i sortiranja

---

```
132         asortiman[i].cena);
133         /* I dodavanje na ukupan racun */
134         racun += asortiman[i].cena;
135     }
136     /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
137        nema vise artikla */
138     printf("Unesite kod artikla [ili 0 za prekid]: \t");
139     scanf("%ld", &kod);
140     if (kod == 0)
141         break;
142 }
143 /* Stampanje ukupnog racuna trenutnog kupca */
144 printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
145 }
146
147 printf("Kraj rada kase!\n");
148
149 exit(EXIT_SUCCESS);
150 }
```

#### Rešenje 3.24

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 500
6
7 /* Struktura sa svim informacijama o pojedinacnom studentu */
8 typedef struct {
9     char ime[20];
10    char prezime[25];
11    int prisustvo;
12    int zadaci;
13 } Student;
14
15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
16    rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
18 {
19     int i, j;
20     int min;
21     Student pom;
22
23     for (i = 0; i < n - 1; i++) {
24         min = i;
25         for (j = i + 1; j < n; j++)
26             if (strcmp(niz[j].ime, niz[min].ime) < 0)
27                 min = j;
28
29         if (min != i) {
```

```

31     pom = niz[min];
    niz[min] = niz[i];
    niz[i] = pom;
33 }
    }
35 }

37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
    zadataka opadajuće, a ukoliko neki studenti imaju isti broj
39 uradjenih zadataka sortiraju se po dužini imena rastuće. */
void sort_zadatke_pa_imena(Student niz[], int n)
41 {
    int i, j;
43     int max;
    Student pom;
45     for (i = 0; i < n - 1; i++) {
        max = i;
47         for (j = i + 1; j < n; j++)
            if (niz[j].zadaci > niz[max].zadaci)
49                 max = j;
            else if (niz[j].zadaci == niz[max].zadaci
51                     && strlen(niz[j].ime) < strlen(niz[max].ime))
                max = j;
53         if (max != i) {
            pom = niz[max];
55             niz[max] = niz[i];
            niz[i] = pom;
57         }
        }
59 }

61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
    su bili opadajuće. Ukoliko neki studenti imaju isti broj casova,
63 sortiraju se opadajuće po broju uradjenih zadataka, a ukoliko se
    i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65 prezimenu opadajuće. */
void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
67 {
    int i, j;
69     int max;
    Student pom;
71     for (i = 0; i < n - 1; i++) {
        max = i;
73         for (j = i + 1; j < n; j++)
            if (niz[j].prisustvo > niz[max].prisustvo)
75                 max = j;
            else if (niz[j].prisustvo == niz[max].prisustvo
77                     && niz[j].zadaci > niz[max].zadaci)
                max = j;
79         else if (niz[j].prisustvo == niz[max].prisustvo
            && niz[j].zadaci == niz[max].zadaci
81             && strcmp(niz[j].prezime, niz[max].prezime) > 0)
                max = j;
    }
}

```

```

        max = j;
83     if (max != i) {
        pom = niz[max];
85     niz[max] = niz[i];
        niz[i] = pom;
87     }
    }
89 }

91 int main(int argc, char *argv[])
{
93     Student praktikum[MAX];
    int i, br_studenata = 0;
95
    FILE *fp = NULL;
97
    /* Otvaranje datoteke za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke aktivnost.txt.\n");
101     exit(EXIT_FAILURE);
    }
103
    /* Ucitavanje sadrzaja */
105     for (i = 0;
        fscanf(fp, "%s%s%d", praktikum[i].ime,
107             praktikum[i].prezime, &praktikum[i].prisustvo,
                &praktikum[i].zadaci) != EOF; i++);
109     /* Zatvaranje datoteke */
    fclose(fp);
111     br_studenata = i;

113     /* Kreiranje prvog spiska studenata po prvom kriterijumu */
    sort_ime_leksikografski(praktikum, br_studenata);
115     /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat1.txt", "w")) == NULL) {
117         fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
        exit(EXIT_FAILURE);
119     }
    /* Upis niza u datoteku */
121     fprintf
        (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
123     for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
125             praktikum[i].prezime, praktikum[i].prisustvo,
                praktikum[i].zadaci);
127     /* Zatvaranje datoteke */
    fclose(fp);
129
    /* Kreiranje drugog spiska studenata po drugom kriterijumu */
131     sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
133     if ((fp = fopen("dat2.txt", "w")) == NULL) {
```

```

135     fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
        exit(EXIT_FAILURE);
    }
137     /* Upis niza u datoteku */
    fprintf(fp, "Studenti sortirani po broju zadataka opadajuće,\n");
139     fprintf(fp, "pa po dužini imena rastuće:\n");
    for (i = 0; i < br_studenata; i++)
141         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
                    praktikum[i].prezime, praktikum[i].prisustvo,
143                    praktikum[i].zadaci);
    /* Zatvaranje datoteke */
145     fclose(fp);

147     /* Kreiranje trećeg spiska studenata po trećem kriterijumu */
    sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum, br_studenata);
149     /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat3.txt", "w")) == NULL) {
151         fprintf(stderr, "Neupesno otvaranje datoteke dat3.txt.\n");
        exit(EXIT_FAILURE);
153     }
    /* Upis niza u datoteku */
155     fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
    fprintf(fp, "pa po broju zadataka,\n");
157     fprintf(fp, "pa po prezimenima leksikografski opadajuće:\n");
    for (i = 0; i < br_studenata; i++)
159         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
                    praktikum[i].prezime, praktikum[i].prisustvo,
161                    praktikum[i].zadaci);
    /* Zatvaranje datoteke */
163     fclose(fp);

165     return 0;
}

```

### Rešenje 3.25

```

#include <stdio.h>
2  #include <stdlib.h>
    #include <string.h>
4
    #define KORAK 10
6
    /* Struktura koja opisuje jednu pesmu */
8  typedef struct {
    char *izvodjac;
10     char *naslov;
    int broj_gledanja;
12 } Pesma;

14 /* Funkcija za upoređivanje pesama po broju gledanosti (potrebna za
    rad qsort funkcije) */

```

### 3 Algoritmi pretrage i sortiranja

---

```
16 int uporedi_gledanost(const void *pp1, const void *pp2)
17 {
18     Pesma *p1 = (Pesma *) pp1;
19     Pesma *p2 = (Pesma *) pp2;
20
21     return p2->broj_gledanja - p1->broj_gledanja;
22 }
23
24 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
25    funkcije) */
26 int uporedi_naslove(const void *pp1, const void *pp2)
27 {
28     Pesma *p1 = (Pesma *) pp1;
29     Pesma *p2 = (Pesma *) pp2;
30
31     return strcmp(p1->naslov, p2->naslov);
32 }
33
34 /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
35    qsort funkcije) */
36 int uporedi_izvodjace(const void *pp1, const void *pp2)
37 {
38     Pesma *p1 = (Pesma *) pp1;
39     Pesma *p2 = (Pesma *) pp2;
40
41     return strcmp(p1->izvodjac, p2->izvodjac);
42 }
43
44 int main(int argc, char *argv[])
45 {
46     FILE *ulaz;
47     Pesma *pesme;
48
49     /* Pokazivac na deo memorije za
50        cuvanje pesama */
51     int alocirano_za_pesme;
52     /* Broj mesta alociranih za pesme */
53     int i;
54     /* Redni broj pesme cije se
55        informacije citaju */
56     int n;
57     /* Ukupan broj pesama */
58     int j, k;
59     char c;
60     int alocirano;
61     /* Broj mesta alociranih za propratne
62        informacije o pesmama */
63     int broj_gledanja;
64
65     /* Priprema datoteke za citanje */
66     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
67     if (ulaz == NULL) {
68         printf("Greska pri otvaranju ulazne datoteke!\n");
69         return 0;
70     }
71
72     /* Citanje informacija o pesmama */
73     pesme = NULL;
```



```
68   alocirano_za_pesme = 0;
69   i = 0;
70
71   while (1) {
72
73       /* Proverava da li je dostignut kraj datoteke */
74       c = fgetc(ulaz);
75       if (c == EOF) {
76           /* Nema vise sadrzaja za citanje */
77           break;
78       } else {
79           /* Inace, vracamo procitani karakter nazad */
80           ungetc(c, ulaz);
81       }
82
83       /* Provera da li postoji dovoljno memorije za citanje nove pesme
84       */
85       if (alocirano_za_pesme == i) {
86
87           /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
88           novih KORAK mesta */
89           alocirano_za_pesme += KORAK;
90           pesme =
91               (Pesma *) realloc(pesme,
92                               alocirano_za_pesme * sizeof(Pesma));
93
94           /* Proverava da li je nova memorija uspesno realocirana */
95           if (pesme == NULL) {
96               /* Ako nije ispisuje se obavestenje */
97               printf("Problem sa alokacijom memorije!\n");
98               /* I oslobadja sva memorija zauzeta do ovog koraka */
99               for (k = 0; k < i; k++) {
100                   free(pesme[k].izvodjac);
101                   free(pesme[k].naslov);
102               }
103               free(pesme);
104               return 0;
105           }
106       }
107
108       /* Ako jeste, nastavlja se sa citanjem pesama ... */
109       /* Cita se ime izvodjaca */
110       j = 0;                                     /* Pozicija na koju treba smestiti
111                                               procitani karakter */
112       alocirano = 0;                             /* Broj alociranih mesta */
113       pesme[i].izvodjac = NULL;                 /* Memorija za smestanje procitanih
114                                               karaktera */
115
116       /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
117       izvodjaca) citaju se karakteri iz datoteke */
118       while ((c = fgetc(ulaz)) != ' ') {
119           /* Proverav da li postoji dovoljno memorije za smestanje
```

```
120     procitanog karaktera */
121     if (j == alocirano) {
122         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
123            se novih KORAK mesta */
124         alocirano += KORAK;
125         pesme[i].izvodjac =
126             (char *) realloc(pesme[i].izvodjac,
127                             alocirano * sizeof(char));
128
129         /* Provera da li je nova alokacija uspesna */
130         if (pesme[i].izvodjac == NULL) {
131             /* Ako nije oslobadja se sva memorija zauzeta do ovog
132                koraka */
133             for (k = 0; k < i; k++) {
134                 free(pesme[k].izvodjac);
135                 free(pesme[k].naslov);
136             }
137             free(pesme);
138             /* I prekida sa izvršavanjem programa */
139             return 0;
140         }
141     }
142
143     /* Ako postoji dovoljno memorije, smestamo procitani karakter
144     */
145     pesme[i].izvodjac[j] = c;
146     j++;
147     /* I nastavlja se sa citanjem */
148 }
149
150 /* Upis terminirajuće nule na kraj reci */
151 pesme[i].izvodjac[j] = '\0';
152
153 /* Preskace se karakter - */
154 fgetc(ulaz);
155
156 /* Preskace se razmak */
157 fgetc(ulaz);
158
159 /* Cita se naslov pesme */
160 j = 0;                                /* Pozicija na koju treba smestiti
161                                         procitani karakter */
162 alocirano = 0;                         /* Broj alociranih mesta */
163 pesme[i].naslov = NULL;                /* Memorija za smestanje procitanih
164                                         karaktera */
165
166 /* Sve do zarez (koji se nalazi nakon naslova pesme) citaju se
167    karakteri iz datoteke */
168 while ((c = fgetc(ulaz)) != ',') {
169     /* Provera da li postoji dovoljno memorije za smestanje
170        procitanog karaktera */

```

```

170     if (j == alocirano) {
171         /* Ako ne, ako je potrošena sva alocirana memorija, alocira
172            se novih KORAK mesta */
173         alocirano += KORAK;
174         pesme[i].naslov =
175             (char *) realloc(pesme[i].naslov,
176                             alocirano * sizeof(char));
177
178         /* Provera da li je nova alokacija uspesna */
179         if (pesme[i].naslov == NULL) {
180             /* Ako nije, oslobadja se sva memorija zauzeta do ovog
181                koraka */
182             for (k = 0; k < i; k++) {
183                 free(pesme[k].izvodjac);
184                 free(pesme[k].naslov);
185             }
186             free(pesme[i].izvodjac);
187             free(pesme);
188
189             /* I prekida izvršavanje programa */
190             return 0;
191         }
192     }
193     /* Ako postoji dovoljno memorije, smesta se procitani karakter
194     */
195     pesme[i].naslov[j] = c;
196     j++;
197     /* I nastavlja dalje sa citanjem */
198 }
199 /* Upisuje se terminirajuca nula na kraj reci */
200 pesme[i].naslov[j] = '\0';
201
202 /* Preskace se razmak */
203 fgetc(ulaz);
204
205 /* Cita se broj gledanja */
206 broj_gledanja = 0;
207
208 /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
209    datoteke */
210 while ((c = fgetc(ulaz)) != '\n') {
211     broj_gledanja = broj_gledanja * 10 + (c - '0');
212 }
213 pesme[i].broj_gledanja = broj_gledanja;
214
215 /* Prelazi se na citanje sledece pesme */
216 i++;
217 }
218
219 /* Informacija o broju procitanih pesama */
220 n = i;
221 /* Zatvaranje nepotrebne datoteke */

```

```
fclose(ulaz);

222
/* Analiza argumenta komandne linije */
224 if (argc == 1) {
    /* Nema dodatnih opcija => sortiranje po broju gledanja */
226 qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
} else {
228     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
        /* Sortiranje po naslovu */
230 qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
    } else {
232         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
            /* Sortiranje po izvodjaku */
234 qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
        } else {
236             printf("Nedozvoljeni argumenti!\n");
            free(pesme);
238             return 0;
        }
    }
240 }
}

242
/* Ispis rezultata */
244 for (i = 0; i < n; i++) {
    printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
246           pesme[i].broj_gledanja);
}

248
/* Oslobadjanje memorije */
250 for (i = 0; i < n; i++) {
    free(pesme[i].izvodjac);
252    free(pesme[i].naslov);
}
254 free(pesme);

256 return 0;
}
```

#### Rešenje 3.28

```
1 #include <stdio.h>
  #include <stdlib.h>
3 #include <math.h>
  #include <search.h>
5
  #define MAX 100
7
  /* Funkcija poredjenja dva cela broja */
9 int compare_int(const void *a, const void *b)
  {
11     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
```

```
13     se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
14
15     /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
16     oduzimanje treba izbegavati 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         return -1;
25     else
26         return 0;
27 }
28
29 int compare_int_desc(const void *a, const void *b)
30 {
31     /* Za obrnuti poredak treba samo oduzimati a od b */
32     /* return *((int *)b) - *((int *)a); */
33
34     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
35     funkcija */
36     return -compare_int(a, b);
37 }
38
39 int main()
40 {
41     size_t n;
42     int i, x;
43     int a[MAX], *p = NULL;
44
45     /* Unos dimenzije */
46     printf("Uneti dimenziju niza: ");
47     scanf("%ld", &n);
48     if (n > MAX)
49         n = MAX;
50
51     /* Unos elementa niza */
52     printf("Uneti elemente niza:\n");
53     for (i = 0; i < n; i++)
54         scanf("%d", &a[i]);
55
56     /* Sortiranje niza celih brojeva */
57     qsort(a, n, sizeof(int), &compare_int);
58
59     /* Prikaz sortiranog niz */
60     printf("Sortirani niz u rastucem poretku:\n");
61     for (i = 0; i < n; i++)
62         printf("%d ", a[i]);
63     putchar('\n');
```

### 3 Algoritmi pretrage i sortiranja

```
/* Pretrazivanje niza */
65 /* Vrednost koja ce biti trazena u nizu */
printf("Uneti element koji se trazi u nizu: ");
67 scanf("%d", &x);

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

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

85 return 0;
}
```

#### Rešenje 3.29

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

5 #define MAX 100

7 /* Funkcija racuna broj delilaca broja x */
9 int no_of_deviders(int x)
{
11     int i;
    int br;

13     /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
15     if (x < 0)
        x = -x;
17     if (x == 0)
        return 0;
19     if (x == 1)
        return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
    br = 2;
23     for (i = 2; i < sqrt(x); i++)
        if (x % i == 0)
25         /* Ako i deli x onda su delioci: i, x/i */
```

```
        br += 2;
27  /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
    promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29  jos jednog delioca */
    if (i * i == x)
31      br++;

33  return br;
}

35  /* Funkcija poredjenja dva cela broja po broju delilaca */
37  int compare_no_deviders(const void *a, const void *b)
{
39      int ak = *(int *) a;
    int bk = *(int *) b;
41      int n_d_a = no_of_deviders(ak);
    int n_d_b = no_of_deviders(bk);
43
    if (n_d_a > n_d_b)
45        return 1;
    else if (n_d_a < n_d_b)
47        return -1;
    else
49        return 0;
}

51  int main()
53  {
    size_t n;
55      int i;
    int a[MAX];

57      /* Unos dimenzije */
59      printf("Uneti dimenziju niza: ");
    scanf("%ld", &n);
61      if (n > MAX)
        n = MAX;

63      /* Unos elementa niza */
65      printf("Uneti elemente niza:\n");
    for (i = 0; i < n; i++)
67        scanf("%d", &a[i]);

69      /* Sortiranje niza celih brojeva prema broju delilaca */
    qsort(a, n, sizeof(int), &compare_no_deviders);

71      /* Prikaz sortiranog niza */
73      printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
    for (i = 0; i < n; i++)
75        printf("%d ", a[i]);
    putchar('\n');
77
```

```
    return 0;  
79 }
```

#### Rešenje 3.30

```
1  #include <stdio.h>  
2  #include <stdlib.h>  
3  #include <string.h>  
4  #include <search.h>  
5  
6  #define MAX_NISKI 1000  
7  #define MAX_DUZINA 30  
8  
9  /*****  
10   Niz nizova karaktera ovog potpisa  
11   char niske[3][4];  
12   se moze graficki predstaviti ovako:  
13   -----  
14   | a | b | c | \0 | | d | e | \0 |   | f | g | h | \0 | |  
15   -----  
16   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za  
17   svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa  
18   nalazi na adresi koja je za 4 veka od prve reci, a za 4 manja od  
19   adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]  
20   i ona je tipa char*.  
21  
22   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata  
23   koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje  
24   reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na  
25   slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp  
26   nad njima.  
27   *****/  
28  int poredi_leksikografski(const void *a, const void *b)  
29  {  
30      return strcmp((char *) a, (char *) b);  
31  }  
32  
33  /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje  
34     leksikografski, vec po duzini */  
35  int poredi_duzine(const void *a, const void *b)  
36  {  
37      return strlen((char *) a) - strlen((char *) b);  
38  }  
39  
40  int main()  
41  {  
42      int i;  
43      size_t n;  
44      FILE *fp = NULL;  
45      char niske[MAX_NISKI][MAX_DUZINA];  
46      char *p = NULL;
```



```

47  char x[MAX_DUZINA];

49  /* Otvaranje datoteke */
if ((fp = fopen("niske.txt", "r")) == NULL) {
51      fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
      exit(EXIT_FAILURE);
53  }

55  /* Citanje sadrzaja datoteke */
for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);

57  /* Zatvaranje datoteke */
fclose(fp);
n = i;

61  /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
63      prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
      niske po duzini */
65  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);

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

71  /* Unos trazene niske */
73  printf("Uneti trazenu nisku: ");
scanf("%s", x);

75  /* Binarna pretraga */
77  /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
      je niz vec sortiran leksikografski. */
79  p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
      &poredi_leksikografski);

81  if (p != NULL)
83      printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
      p, (p - (char *) niske) / MAX_DUZINA);
85  else
      printf("Niska nije pronadjena u nizu\n");

87  /* Linearna pretraga */
89  p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
      &poredi_leksikografski);

91  if (p != NULL)
93      printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
      p, (p - (char *) niske) / MAX_DUZINA);
95  else
      printf("Niska nije pronadjena u nizu\n");

97  /* Sortiranje po duzini */

```

### 3 Algoritmi pretrage i sortiranja

```
99  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
101  printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
103      printf("%s ", niske[i]);
    printf("\n");
105
    exit(EXIT_SUCCESS);
107 }
```

#### Rešenje 3.31

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <search.h>
5
   #define MAX_NISKI 1000
7  #define MAX_DUZINA 30
9
   /******
    Niz pokazivaca na karaktere ovog potpisa
11  char *niske[3];
    posle alokacije u main-u se moze graficki predstaviti ovako:
13  -----
    | X | -----> | a | b | c | \0|
15  -----
    | Y | -----> | d | e | \0|
17  -----
    | Z | -----> | f | g | h | \0|
19  -----
    Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
21  njegov element pokazivac koji pokazuje na alocirane karaktere i-te
    reci. Njegov tip je char*.
23
    Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
25  koji trebaju biti uporedjeni (recimo adresu od X i adresu od Z), i
    kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
27  moraju i kastovati. Da bi se leksikografski uporedili elementi X i
    Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
29  u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
    kastovani na odgovarajuci tip.
31  *****/
   int poredi_leksikografski(const void *a, const void *b)
33  {
    return strcmp(*(char **) a, *(char **) b);
35  }

37  /* Funkcija slicna prethodnoj, osim sto elemente ne upoređuje
    leksikografski, vec po duzini */
39  int poredi_duzine(const void *a, const void *b)
```

```
41 {
    return strlen(*(char **) a) - strlen(*(char **) b);
43 }
45 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
    element u nizu sa kojim se poredi, pa njega treba kastovati na
    char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
    ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
    main funkciji je to x, koji je tipa char*, tako da pokazivac a
    ovde samo treba kastovati i ne dereferencirati. */
49 int poredi_leksikografski_b(const void *a, const void *b)
51 {
    return strcmp((char *) a, *(char **) b);
53 }
55 int main()
57 {
    int i;
    size_t n;
    FILE *fp = NULL;
    char *niske[MAX_NISKI];
    char **p = NULL;
    char x[MAX_DUZINA];
63
    /* Otvaranje datoteke */
65 if ((fp = fopen("niske.txt", "r")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
67     exit(EXIT_FAILURE);
    }
69
    /* Citanje sadrzaja datoteke */
71 i = 0;
    while (fscanf(fp, "%s", x) != EOF) {
73     /* Alociranje dovoljne memorije za i-tu nisku */
        if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
75             fprintf(stderr, "Greska pri alociranju niske\n");
                exit(EXIT_FAILURE);
77         }
        /* Kopiranje procitane niske na svoje mesto */
79         strcpy(niske[i], x);
            i++;
81     }
83
    /* Zatvaranje datoteke */
    fclose(fp);
    n = i;
85
    /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
    prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
    niske po duzini */
87 qsort(niske, n, sizeof(char *), &poredi_leksikografski);
89
91
```

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

/* Unos trazene niske */
printf("Uneti trazenu nisku: ");
99 scanf("%s", x);

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

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

/* Sortiramo po duzini */
qsort(niske, n, sizeof(char *), &poredi_duzine);

printf("Niske sortirane po duzini:\n");
121 for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
123 printf("\n");

/* Oslobadjanje zauzete memorije */
125 for (i = 0; i < n; i++)
    free(niske[i]);
127
129 exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.32

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

6 #define MAX 500

8 /* Struktura sa svim informacijama o pojedinacnom studentu */
typedef struct {
```

```
10  char ime[21];
11  char prezime[21];
12  int bodovi;
13  } Student;

14
15  /* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
16     istim brojem bodova se dodatno sortiraju leksikografski po
17     prezimenu */
18  int poredi1(const void *a, const void *b)
19  {
20      Student *prvi = (Student *) a;
21      Student *drugi = (Student *) b;
22
23      if (prvi->bodovi > drugi->bodovi)
24          return -1;
25      else if (prvi->bodovi < drugi->bodovi)
26          return 1;
27      else
28          /* Ako su jednaki po broju bodova, treba ih uporediti po
29             prezimenu */
30          return strcmp(prvi->prezime, drugi->prezime);
31  }
32
33  /* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
34     Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
35     parametar je element niza ciji se bodovi porede. */
36  int poredi2(const void *a, const void *b)
37  {
38      int bodovi = *(int *) a;
39      Student *s = (Student *) b;
40      return s->bodovi - bodovi;
41  }
42
43  /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
44     Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
45     parametar je element niza cije se prezime poredi. */
46  int poredi3(const void *a, const void *b)
47  {
48      char *prezime = (char *) a;
49      Student *s = (Student *) b;
50      return strcmp(prezime, s->prezime);
51  }
52
53  int main(int argc, char *argv[])
54  {
55      Student kolokvijum[MAX];
56      int i;
57      size_t br_studenata = 0;
58      Student *nadjen = NULL;
59      FILE *fp = NULL;
60      int bodovi;
61      char prezime[21];
```

### 3 Algoritmi pretrage i sortiranja

---

```
62  /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64     informacija korisniku kako se program koristi i prekida se
        izvršavanje. */
66  if (argc < 2) {
67      fprintf(stderr,
68          "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
        argv[0]);
69      exit(EXIT_FAILURE);
70  }
71
72  /* Otvaranje datoteke */
73
74  if ((fp = fopen(argv[1], "r")) == NULL) {
75      fprintf(stderr, "Neuspješno otvaranje datoteke %s\n", argv[1]);
76      exit(EXIT_FAILURE);
77  }
78
79  /* Učitavanje sadržaja */
80  for (i = 0;
81      fscanf(fp, "%s%s%d", kolokvijum[i].ime,
82          kolokvijum[i].prezime,
83          &kolokvijum[i].bodovi) != EOF; i++);
84
85  /* Zatvaranje datoteke */
86  fclose(fp);
87  br_studenata = i;
88
89  /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
90     studenata sa istim brojem bodova sortiranje vrši po prezimenu */
91  qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
92
93  printf("Studenti sortirani po broju poena opadajuće, ");
94  printf("pa po prezimenu rastuće:\n");
95  for (i = 0; i < br_studenata; i++)
96      printf("%s %s %d\n", kolokvijum[i].ime,
97          kolokvijum[i].prezime, kolokvijum[i].bodovi);
98
99  /* Pretraživanje studenata po broju bodova se vrši binarnom
100     pretragom jer je niz sortiran po broju bodova. */
101  printf("Unesite broj bodova: ");
102  scanf("%d", &bodovi);
103
104  nadjen =
105      bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
106          &poredi2);
107
108  if (nadjen != NULL)
109      printf
110          ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
        nadjen->ime, nadjen->prezime, nadjen->bodovi);
111  else
112      printf("Nema studenta sa unetim brojem bodova\n");
```

```

114      /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116         sortiran po bodovima. */
118      printf("Unesite prezime: ");
119      scanf("%s", prezime);

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

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

130      return 0;
132  }

```

### Rešenje 3.33

```

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

4
#define MAX 128

6
/* Funkcija poredi dva karaktera */
8 int uporedi_char(const void *pa, const void *pb)
{
10     return *(char *) pa - *(char *) pb;
}

12
/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14 int anagrami(char s[], char t[])
{
16     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
17     if (strlen(s) != strlen(t))
18         return 0;

20     /* Sortiranje niski */
21     qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);

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

28
int main()

```

```
30 {
31     char s[MAX], t[MAX];
32
33     /* Unos niski */
34     printf("Unesite prvu nisku: ");
35     scanf("%s", s);
36     printf("Unesite drugu nisku: ");
37     scanf("%s", t);
38
39     /* Ispituje se da li su niske anagrami */
40     if (anagrami(s, t))
41         printf("jesu\n");
42     else
43         printf("nisu\n");
44
45     return 0;
46 }
```

#### Rešenje 3.34

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  #define MAX 10
6  #define MAX_DUZINA 32
7
8  /* Funkcija porenjenja */
9  int uporedi_niske(const void *pa, const void *pb)
10 {
11     return strcmp((char *) pa, (char *) pb);
12 }
13
14 int main()
15 {
16     int i, n;
17     char S[MAX][MAX_DUZINA];
18
19     /* Unos broja niski */
20     printf("Unesite broj niski:");
21     scanf("%d", &n);
22
23     /* Unos niza niski */
24     printf("Unesite niske:\n");
25     for (i = 0; i < n; i++)
26         scanf("%s", S[i]);
27
28     /* Sortiranje niza niski */
29     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
30
31     /*****
```



```

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

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

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

47     printf("nema\n");
49     return 0;
}

```

### Rešenje 3.35

```

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

5  #define MAX 21

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

15 /* Funkcija poredi studente prema broju poena, rastuce */
    int uporedi_poeni(const void *a, const void *b)
17 {
        Student s = *(Student *) a;
19     Student t = *(Student *) b;
        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 se izdvaja godina upisa, smer i

```

```
        broj indeksa */
31 int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
    int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
33 char smer1 = s.nalog[1];
    char smer2 = t.nalog[1];
35 int indeks1 =
        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
37     s.nalog[6] - '0';
    int indeks2 =
39     (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
        t.nalog[6] - '0';
41 if (godina1 != godina2)
    return godina1 - godina2;
43 else if (smer1 != smer2)
    return smer1 - smer2;
45 else
    return indeks1 - indeks2;
47 }

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

73 int main(int argc, char **argv)
{
75     Student *nadjen = NULL;
    char nalog_trazeni[8];
77     Student niz_studenata[100];
    int i = 0, br_studenata = 0;
79     FILE *in = NULL, *out = NULL;
```

```
81  /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
    korisnik nije ispravno pozvao program i prijavljuje se greska.
    */
83  if (argc != 2 && argc != 3) {
    fprintf(stderr,
85      "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
    );
    exit(EXIT_FAILURE);
87  }

89  /* Otvaranje datoteke za citanje */
in = fopen("studenti.txt", "r");
91  if (in == NULL) {
    fprintf(stderr,
93      "Greska prilikom otvarnja datoteke studenti.txt!\n");
    exit(EXIT_FAILURE);
95  }

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

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

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

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

129  /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
    studenta... */
```

### 3 Algoritmi pretrage i sortiranja

```
131  if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
      strcpy(nalog_trazeni, argv[2]);
133
      /* ... pronalazi se student sa tim nalogom... */
135  nadjen =
      (Student *) bsearch(nalog_trazeni, niz_studenata,
137                      br_studenata, sizeof(Student),
                      &uporedi_bsearch);
139
      if (nadjen == NULL)
141          printf("Nije nadjen!\n");
      else
143          printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
                      nadjen->prezime, nadjen->poeni);
145  }

147  /* Zatvaranje datoteka */
      fclose(in);
149  fclose(out);

151  return 0;
}
```

#### Rešenje 3.37

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

4 /* Funkcija koja učitava elemente matrice a dimenzije nxm sa
   standardnog ulaza */
6 void učitaj_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 određuje 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 (pokazivace na vrste) na
   osnovu zbira koriscenjem selection sort algoritma */
32 void sortiraj_vrste(int **a, int n, int m)
33 {
34     int i, j, min;

36     for (i = 0; i < n - 1; i++) {
37         min = i;
38         for (j = i + 1; j < n; j++) {
39             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
40                 min = j;
41             }
42         }
43         if (min != i) {
44             int *tmp;
45             tmp = a[i];
46             a[i] = a[min];
47             a[min] = tmp;
48         }
49     }
50 }

52 /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
   standardni izlaz */
54 void ispisi_matricu(int **a, int n, int m)
55 {
56     int i, j;

58     for (i = 0; i < n; i++) {
59         for (j = 0; j < m; j++) {
60             printf("%d ", a[i][j]);
61         }
62         printf("\n");
63     }
64 }

66 /* Funkcija koja alokira memoriju za matricu dimenzija nxm */
67 int **alociraj_memoriju(int n, int m)
68 {
69     int i, j;
70     int **a;

72     a = (int **) malloc(n * sizeof(int *));
73     if (a == NULL) {
74         fprintf(stderr, "Problem sa alokacijom memorije!\n");
75         exit(EXIT_FAILURE);
76     }
77     /* Za svaku vrstu ponaosob */
78     for (i = 0; i < n; i++) {
```

### 3 Algoritmi pretrage i sortiranja

---

```
80      /* Alocira se memorija */
      a[i] = (int *) malloc(m * sizeof(int));
82      /* Proverava se da li je doslo do greske prilikom alokacije */
      if (a[i] == NULL) {
          /* Ako jeste, ispisuje se poruka */
84          fprintf(stderr, "Problem sa alokacijom memorije!\n");
          /* I oslobadja memorija zauzeta do ovog koraka */
86          for (j = 0; j < i; j++) {
              free(a[j]);
88          }
          free(a);
90          exit(EXIT_FAILURE);
      }
92  }

94  return a;
96  }

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

106  int main(int argc, char *argv[])
{
108     int **a;
110     int n, m;

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

116     /* Alokacija memorije */
     a = alociraj_memoriju(n, m);
118

     /* Ucitavanje elementa matrice */
120     ucitaj_matricu(a, n, m);

122     /* Poziv funkcije koja sortira vrste matrice prema zbiru */
     sortiraj_vrste(a, n, m);
124

     /* Ispis rezultujuce matrice */
126     printf("Sortirana matrica je:\n");
     ispisi_matricu(a, n, m);
128

     /* Oslobadjanje memorije */
```

```
130     oslobodi_memoriju(a, n, m);  
132     return 0;  
    }
```





## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

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

- (a) Definirati strukturu `Cvor` kojom se predstavlja čvor liste. Čvor treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.
- (b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje polja novog čvora i vraća njegovu adresu.
- (c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.
- (d) Napisati funkciju `Cvor* pronadji_poslednji(Cvor* glava)` koja pronalazi poslednji čvor u listi.
- (e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste. Povratna vrednost je sa istim značenjem kao kod funkcije
- (f) Napisati funkciju `Cvor* pronadji_mesto_umetanja(Cvor* glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.

- (g) Napisati funkciju `void dodaj_iza(Cvor* tekuci, Cvor* novi)` koja uvezuje u postojeću listu čvor `novi` iza čvora `tekuci`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor** adresa_glave, int broj)` koja dodaje novi element u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor* glava)` koja ispisuje čvorove liste uokvirene zagradama `[, ]` i međusobno razdvojene zapetama.
- (j) Napisati funkciju `Cvor* pretrazi_listu(Cvor* glava, int broj)` koja proverava da li se u listi nalazi čvor čija se vrednost zadaje kao argument funkcije.
- (k) Napisati funkciju `Cvor* pretrazi_sortiranu_listu(Cvor* glava, int broj)` koja proverava da li se u listi nalazi čvor čija se vrednost zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenom listom.
- (l) Napisati funkciju `void obrisi_cvor(Cvor** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost koja se zadaje kao argument funkcije.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost koja se zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenoj listi.
- (n) Napisati funkciju `void oslobodi_listu(Cvor** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

NAPOMENA: *Sve funkcije za rad sa listom implementirati iterativno.*

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0 ukoliko je sve bilo u redu, odnosno 1 ukoliko se dogodila greška prilikom alokacije memorije za nov čvor.

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elementa koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

*Primer 1*

```

Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [3, 2]
14
Lista: [14, 3, 2]
5
Lista: [5, 14, 3, 2]
3
Lista: [3, 5, 14, 3, 2]
17
Lista: [17, 3, 5, 14, 3, 2]

Unosite broj koji se trazi: 5
Trazeni broj 5 je u listi!

```

*Primer 2*

```

Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [35, 14, 23]

Unosite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!

```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

*Primer 1*

```

Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 14, 3]
3
Lista: [2, 3, 14, 3, 3]
17
Lista: [2, 3, 14, 3, 3, 17]
3
Lista: [2, 3, 14, 3, 3, 17, 3]

Unosite broj koji se brise: 3
Lista nakon brisanja: [2, 14, 17]

```

*Primer 2*

```

Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [23, 14]
35
Lista: [23, 14, 35]

Unosite broj koji se brise: 3
Lista nakon brisanja: [23, 14, 35]

```

- (3) U glavnom programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi ceo, zatim, broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni

## 4 Dinamičke strukture podataka

sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

### Primer 1

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Unesite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 3, 14]
3
Lista: [2, 3, 3, 3, 14]
5
Lista: [2, 3, 3, 3, 5, 14]

Unesite broj koji se traži: 14
Trazeni broj 14 je u listi!

Unesite broj koji se briše: 3
Lista nakon brisanja: [2, 5, 14]
```

### Primer 2

```
Poziv: ./a.out

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

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

Unesite broj koji se briše: 3
Lista nakon brisanja: [14, 23, 35]
```

[Rešenje 4.1]

**Zadatak 4.2** Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekursivno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

- Napisati funkciju `void obrisi_tekuci(Cvor** adresa_glave, Cvor* tekuci)` koja iz liste čija se glava nalazi na adresi `adresa_glave` briše čvor na koji pokazuje pokazivač `tekuci`.
- Napisati funkciju `void ispisi_listu_unazad(Cvor* glava)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1. Ove programe dopuniti pozivom funkcije koja ispisuje listu unazad.*

[Rešenje 4.3]

**Zadatak 4.4** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

*Test 1*

```
Poziv: ./a.out

IZRAZ.TXT
{[23 + 5344] * (24 - 234)} - 23

IZLAZ:
Zagrade su ispravno uparene.
```

*Test 2*

```
Poziv: ./a.out

IZRAZ.TXT
{[23 + 5] * (9 * 2)} - {23}

IZLAZ:
Zagrade su ispravno uparene.
```

*Test 3*

```
Poziv: ./a.out

IZRAZ.TXT
{[2 + 54] / (24 * 87)} + (234 + 23)

IZLAZ:
Zagrade nisu ispravno uparene.
```

*Test 4*

```
Poziv: ./a.out

IZRAZ.TXT
{(2 - 14) / (23 + 11)} * (2 + 13)

IZLAZ:
Zagrade nisu ispravno uparene.
```

*Test 5*

```
Poziv: ./a.out

IZRAZ.TXT
Datoteka je prazna.

IZLAZ:
Zagrade su ispravno uparene.
```

*Test 6*

```
Poziv: ./a.out

IZRAZ.TXT
Datoteka ne postoji.

IZLAZ:
Greska prilikom otvaranja
datoteke izraz.txt!
```

[Rešenje 4.4]

**Zadatak 4.5** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže HTML etikete.

### Test 1

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

DATOTEKA.HTML
<html>
  <head>
    <title>Primer</title>
  </head>
  <body>
  </body>

IZLAZ:
Etikete nisu pravilno uparene
(etiketa <html> nije zatvorena)
```

### Test 2

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

DATOTEKA.HTML
<head>
  <title>Primer</title>
</head>
<body>
</body>
</html>

IZLAZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html> koja nije otvorena)
```

### Test 3

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

DATOTEKA.HTML
<html>
  <head>
    <title>Primer</title>
  </head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    Sutra ce biti jos lepsi.
    <a link='http://www.math.rs'>Link</a>
  </body>
</html>

IZLAZ:
Etikete su pravilno uparene!
```

### Test 4

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

DATOTEKA.HTML
<html>
  <head>
    <title>Primer</title>
  </head>
  <body>
  </body>
</html>

IZLAZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html>, a poslednja
otvorena je <body>)
```

### Test 5

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

DATOTEKA.HTML
Datoteka ne postoji.

IZLAZ:
Greska prilikom otvaranja
datoteke datoteka.html.
```

### Test 6

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

DATOTEKA.HTML
Datoteka je prazna.

IZLAZ:
Etikete su pravilno uparene!
```

[Rešenje 4.5]

**Zadatak 4.6** Napisati program kojim se simulira rad jednog šaltera na kojem se prvo kod službenika zakazuju termini, a potom službenik uslužuje korisnike. Službenik evidentira korisničke JMBG brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (EOF). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije, tj. službeniku

se postavlja pitanje **Da li korisnika vracate na kraj reda?** i ukoliko on da odgovor **Da**, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije **Da**, tada službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke **izvestaj.txt**. Ova datoteka, za svaki obrađen zahtev, sadrži **JMBG** i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

### Primer 1

```
Poziv: ./a.out

INTERAKCIJA PROGRAMA:
Sluzbenik evidentira korisnicke zahteve:m
Novi zahtev [CTRL+D za kraj]
JMBG: 1234567890123
Opis problema: Otvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 2345678901234
Opis problema: Podizanje novca

Novi zahtev [CTRL+D za kraj]
JMBG: 3456789012345
Opis problema: Reklamacija

Novi zahtev [CTRL+D za kraj]
JMBG:

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 2345678901234
i zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

IZVESTAJ.TXT
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 3456789012345 Zahtev: Reklamacija
JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

[Rešenje 4.6]

**Zadatak 4.7** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti



strukturu:

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

#### Test 1

Poziv: ./a.out datoteka.html

DATOTEKA.HTML

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

IZLAZ:

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

#### Test 2

Poziv: ./a.out datoteka.html

DATOTEKA.HTML

Datoteka ne postoji.

IZLAZ:

Greska prilikom otvaranja  
datoteke datoteka.html.

[Rešenje 4.7]

**Zadatak 4.8** U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan po jedan, indksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku **da** ili **ne**, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

### Primer 1

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

STUDENTI.TXT
123/2014 Marko Lukic
3/2014 Ana Sokic
43/2013 Jelena Ilic
41/2009 Marija Zaric
13/2010 Milovan Lazic

INTERAKCIJA PROGRAMA:
3/2014 da: Ana Sokic
235/2008 ne
41/2009 da: Marija Zaric
```

### Primer 2

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

STUDENTI.TXT
Datoteka je prazna.

INTERAKCIJA PROGRAMA:
3/2014 ne
235/2008 ne
41/2009 ne
```

[Rešenje 4.8]

**Zadatak 4.9** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove čvorove, već da samo prerašpodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

### 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 dat1.txt dat2.txt

DAT1.TXT
2 4 6 10 15

DAT2.TXT
Datoteka ne postoji.

IZLAZ:
Greska prilikom otvaranja datoteke
dat2.txt.
```

### Test 3

```
Poziv: ./a.out dat1.txt dat2.txt

DAT1.TXT
Datoteka je prazna.

DAT2.TXT
5 6 11 12 14 16

IZLAZ:
[5, 6, 11, 12, 14, 16]
```

### Test 4

```
Poziv: ./a.out dat1.txt

IZLAZ:
Greska! Program se poziva sa:
./a.out dat1.txt dat2.txt!
```

[Rešenje 4.9]

**Zadatak 4.10** Date su dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od ovih listi formira novu listu L koja sadrži naizmenično raspoređene čvorove listi L1 i L2: prvi čvor iz L1, prvi čvor iz L2, drugi čvor L1, drugi čvor L2, itd. Ne formirati nove čvorove, već samo postojeće čvorove rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *Iskoristiti testove 2 - 6 za zadatak 4.9.*

#### Test 1

```
POZIV: ./a.out dat1.txt dat2.txt

DAT1.TXT
 2 4 6 10 15

DAT2.TXT
 5 6 11 12 14 16

IZLAZ:
 2 5 4 6 6 11 10 12 15 14 16
```

**Zadatak 4.11** Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

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

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

#### Test 1

```
POZIV: ./a.out

BROJEVI.TXT
43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
12 15 16
```

#### Test 2

```
POZIV: ./a.out

BROJEVI.TXT
Datoteka ne postoji.

IZLAZ:
REZULTAT.TXT
Greska prilikom otvaranja
datoteke brojevi.txt.
```

#### Test 3

```
POZIV: ./a.out

BROJEVI.TXT
Datoteka je prazna.

IZLAZ:
REZULTAT.TXT
Rezultat.txt ce biti prazna.
```

**Zadatak 4.12** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje počinje od

## 4 Dinamičke strukture podataka

---

sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

Test 1	Test 2	Test 3
<pre>Poziv: ./a.out ULAZ: 5 3 IZLAZ: 3 1 5 2 4</pre>	<pre>Poziv: ./a.out ULAZ: 8 4 IZLAZ: 4 8 5 2 1 3 7 6</pre>	<pre>Poziv: ./a.out ULAZ: 3 8 IZLAZ: n mora biti uvek vece od k, a 3 &lt; 8!</pre>

**Zadatak 4.13** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ , 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 odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.* NAPOMENA: *Iskoristiti test 3 iz 4.12. zadatka.*

Test 1	Test 2
<pre>Poziv: ./a.out ULAZ: 5 3 IZLAZ: 3 5 4 2 1</pre>	<pre>Poziv: ./a.out ULAZ: 8 4 IZLAZ: 4 8 5 7 6 3 2 1</pre>

## 4.2 Stabla

**Zadatak 4.14** Napisati program za rad sa binarnim pretraživačkim stablima.

- (a) Definirati 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.
- (b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.
- (c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.
- (d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili `NULL` ukoliko takav čvor ne postoji.
- (e) Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- (f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.
- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

Korišćenjem prethodnih funkcija, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

### Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Traži se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuće stablo: 1 2 9 18 32
```

### Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Traži se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuće stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

**Zadatak 4.15** Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku **Nedostaje ime ulazne datoteke!**. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

### Test 1

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

TEST.TXT
Sunce utorak raCunar SUNCE programiranje
jabuka PROGramiranje sunCE JABUka

IZLAZ:
jabuka: 2
programiranje: 2
racunar: 1
sunce: 3
utorak: 1

Najcesca rec: sunce (pojavljuje se 3 puta)
```

### Test 2

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

SUMA.TXT
lipa zova hrast ZOVA breza LIPA

IZLAZ:
breza: 1
hrast: 1
lipa: 2
zova: 2

Najcesca rec: lipa
(pojavljuje se 2 puta)
```

### Test 3

```
Poziv: ./a.out

IZLAZ:
Nedostaje ime ulazne datoteke!
```

[Rešenje 4.15]

**Zadatak 4.16** U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. **Pera Peric**

064/123-4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči KRAJ, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

#### Primer 1

```
IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858

INTERAKCIJA PROGRAMA:
Unesite ime datoteke: imenik.txt
Unesite ime i prezime: Pera Peric
Broj je: 011/3240-987
Unesite ime i prezime: Marko Markovic
Broj nije u imeniku!
Unesite ime i prezime: KRAJ
```

#### Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite ime datoteke: imenik1.txt
Greska prilikom otvaranja datoteke
imenik1.txt!
```

[Rešenje 4.16]

**Zadatak 4.17** U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (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

```
PRIJEMNI.TXT
Marko Markovic 45.4 12.3 11
Milan Jevremovic 35.2 1.3 9
Maja Agic 60 19 20
Nadica Zec 54.2 10 15.8
Jovana Milic 23.3 2 5.6

IZLAZ:
1. Maja Agic 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9
```

### Test 2

```
PRIJEMNI.TXT
[Ova datoteka ne postoji]

IZLAZ:
Greska prilikom otvaranja datoteke!
```

[Rešenje 4.17]

\* **Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.YYYY.`

### Primer 1

```
POZIV: a.out rodjendani.txt

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.

INTERAKCIJA PROGRAMA:
Unesite datum: 23.04.
Slavljenik: Maja Agic
Unesite datum: 01.01.
Slavljenik: Nadica Zec
Unesite datum: 01.05.
Slavljenik: Jovana Milic 05.05.
Unesite datum:
```

### Primer 2

```
POZIV: a.out rodjendani1.txt

INTERAKCIJA PROGRAMA:
Greska prilikom otvaranja datoteke!
```

[Rešenje 4.18]



**Zadatak 4.19** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napisati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. *NAPOMENA: Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka `stabla.h` predstavljati popis funkcija biblioteke, a datoteka `stabla.c` njihove implementacije. Programe koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

*Primer 1*

```
INTERAKCIJA PROGRAMA:
Prvo stablo:
10 5 15 3 2 4 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla jesu identicna.
```

*Primer 2*

```
INTERAKCIJA PROGRAMA:
Prvo stablo:
10 5 15 4 3 2 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla nisu identicna.
```

[Rešenje 4.19]

\* **Zadatak 4.20** Napisati program 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.

*Primer 1*

```
INTERAKCIJA PROGRAMA:
Prvo stablo: 1 7 8 9 2 2
Drugo stablo: 3 9 6 11 1
Unija: 1 1 2 2 3 6 7 8 9 9 11
Presek: 1 9
Razlika: 2 2 7 8
```

*Primer 2*

```
INTERAKCIJA PROGRAMA:
Prvo stablo: 11 2 7 5
Drugo stablo: 4 3 3 7
Unija: 2 3 3 4 5 7 7 11
Presek: 7
Razlika: 2 5 11
```

[Rešenje 4.20]

**Zadatak 4.21** Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

### Primer 1

```
|| INTERAKCIJA PROGRAMA:
|| n: 7
|| a: 1 11 8 6 37 25 30
|| 1 6 8 11 25 30 37
```

### Primer 2

```
|| INTERAKCIJA PROGRAMA:
|| n: 55
|| Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

**Zadatak 4.22** Dato je binarno pretraživačko stablo celih brojeva.

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

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

## Test 1

```

Poziv: ./a.out 2 15

ULAZ:
  10 5 15 3 2 4 30 12 14 13

IZLAZ:
  broj cvorova: 10
  broj listova: 4
  pozitivni listovi: 2 4 13 30
  zbir cvorova: 108
  najveći element: 30
  dubina stabla: 5
  broj cvorova na 2. nivou: 3
  elementi na 2. nivou: 3 12 30
  maksimalni na 2. nivou: 30
  zbir na 2. nivou: 45
  zbir elemenata manjih ili jednakih od 15: 78

```

## Test 2

```

Poziv: ./a.out 3 31

ULAZ:
  24 53 61 9 7 55 20 16

IZLAZ:
  broj cvorova: 8
  broj listova: 3
  pozitivni listovi: 7 16 55
  zbir cvorova: 245
  najveći element: 61
  dubina stabla: 4
  broj cvorova na 3. nivou: 2
  elementi na 3. nivou: 16 55
  maksimalni na 3. nivou: 55
  zbir na 3. nivou: 71
  zbir elemenata manjih ili jednakih od 31: 76

```

[Rešenje 4.22]

**Zadatak 4.23** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

## Test 1

```

ULAZ:
  10 5 15 3 2 4 30 12 14 13

IZLAZ:
  0.nivo: 10
  1.nivo: 5 15
  2.nivo: 3 12 30
  3.nivo: 2 4 14
  4.nivo: 13

```

## Test 2

```

ULAZ:
  6 11 8 3 -2

IZLAZ:
  0.nivo: 6
  1.nivo: 3 11
  2.nivo: -2 8

```

## Test 3

```

ULAZ:
  24 53 61 9 7 55 20 16

IZLAZ:
  0.nivo: 24
  1.nivo: 9 53
  2.nivo: 7 20 61
  3.nivo: 16 55

```

[Rešenje 4.23]

\* **Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

### Primer 1

```
INTERAKCIJA PROGRAMA:
Prvo stablo: 11 20 5 3 0
Drugo stablo: 8 14 30 1 0
Stabla su slicna kao u ogledalu.
```

### Primer 2

```
INTERAKCIJA PROGRAMA:
Prvo stablo: 11 20 5 3 0
Drugo stablo: 8 20 15 0
Stabla nisu slicna kao u ogledalu.
```

**Zadatak 4.25** AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

### Test 1

```
ULAZ:
10 5 15 2 11 16 1 13

IZLAZ:
7
```

### Test 2

```
ULAZ:
16 30 40 24 10 18 45 22

IZLAZ:
6
```

[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva hip. Napisati zatim i glavni program koji kreira stablo kao na slici 4.1, poziva funkciju `heap` i ispisuje rezultat na standardnom izlazu.

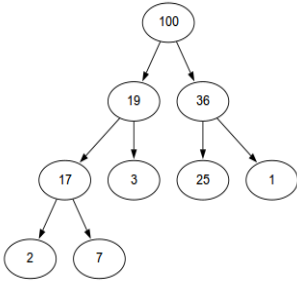
### Test 1

```
IZLAZ:
Zadato stablo je heap!
```

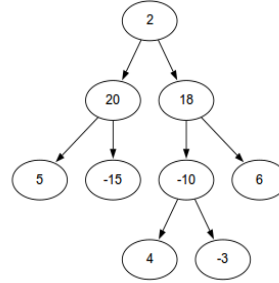
[Rešenje 4.26]

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.
- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardnom izlazu.

*Test 1*

```

|| IZLAZ:
|| Cvor sa maksimalnim desnim zbirom: 18
|| Cvor sa minimalnim levim zbirom: 18
|| 2 18 -10 4
|| 2 20 -15
  
```

## 4.3 Rešenja

### Rešenje 4.1

```

1 #ifndef _LISTA_H
2 #define _LISTA_H

4 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
6 typedef struct cvor {
   int vrednost;
8   struct cvor *sledeci;
} Cvor;
  
```

```
10 Cvor *napravi_cvor(int broj);
12 void oslobodi_listu(Cvor ** adresa_glave);
14 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
16 Cvor *pronadji_poslednji(Cvor * glava);
18 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
20 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
22 void dodaj_iza(Cvor * tekuci, Cvor * novi);
24 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
26 Cvor *pretrazi_listu(Cvor * glava, int broj);
28 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
30 void obrisi_cvor(Cvor ** adresa_glave, int broj);
32 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
34 void ispisi_listu(Cvor * glava);
36 #endif
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog cvora
6    inicijalizuje na broj, dok pokazivac na sledeci cvor u novom cvoru
7    postavlja na NULL. Funkcija vraća pokazivac na novokreirani cvor
8    ili NULL ako alokacija nije uspesno izvršena. */
9 Cvor *napravi_cvor(int broj)
10 {
11     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
12     if (novi == NULL)
13         return NULL;
14
15     novi->vrednost = broj;
16     novi->sledeci = NULL;
17     return novi;
18 }
19
20 /* Funkcija oslobadja dinamičku memoriju zauzetu za cvorove liste
21    čiji se početni cvor nalazi na adresi adresa_glave. */
22 void oslobodi_listu(Cvor ** adresa_glave)
23 {
```

```
25     Cvor *pomocni = NULL;
26
27     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
28     while (*adresa_glave != NULL) {
29         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
30            osloboditi cvor koji predstavlja glavu liste */
31         pomocni = (*adresa_glave)->sledeci;
32         free(*adresa_glave);
33         /* Sledeci cvor je nova glava liste. */
34         *adresa_glave = pomocni;
35     }
36
37     /* Funkcija dodaje broj na pocetak liste. Kreira novi cvor
38        koriscenjem funkcije napravi_cvor i uvezuje ga na pocetak.
39        Funkcija treba da vrati 0 ukoliko je sve bilo u redu, odnosno 1
40        ukoliko je doslo do greske prilikom alokacije memorije za nov
41        cvor. */
42     int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
43     {
44         /* Kreira se nov cvor i proverava se da li je bilo greske pri
45            alokaciji */
46         Cvor *novi = napravi_cvor(broj);
47         if (novi == NULL)
48             return 1;
49
50         /* Novi cvor se uvezuje na pocetak, i tako postaje nova glave
51            liste. */
52         novi->sledeci = *adresa_glave;
53         *adresa_glave = novi;
54
55         return 0;
56     }
57
58     /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
59        NULL ukoliko je lista prazna. */
60     Cvor *pronadji_poslednji(Cvor * glava)
61     {
62         /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
63         if (glava == NULL)
64             return NULL;
65
66         /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pokazivac
67            glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
68            ce pokazivati na cvor liste koji nema sledeceg, tj, poslednji je
69            cvor liste, vraca se vrednost pokazivaca glava. Pokazivac glava
70            je argument funkcije i njegove promene nece se odraziti na
71            vrednost pokazivaca glava u pozivajucoj funkciji. */
72         while (glava->sledeci != NULL)
73             glava = glava->sledeci;
74
75         return glava;
```

```

}
77
/* Funkcija dodaje broj na kraj liste. Ukoliko dodje do greske pri
79 alokaciji memorije vratice 1, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
81 {
    Cvor *novi = napravi_cvor(broj);
83     if (novi == NULL)
        return 1;
85
    /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
87    ujedno i cela lista. Azurira se vrednost na koju pokazuje
    adresa_glave i tako se azurira i pokazivacka promenljivu u
89    pozivajucoj funkciji. */
    if (*adresa_glave == NULL) {
91        *adresa_glave = novi;
        return 0;
93    }

95    /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
    se dodaje na kraj liste kao sledbenik poslednjeg. */
97    Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
99
    return 0;
101 }

103 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
    nov cvor sa vrednoscu broj. */
105 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
    {
107     /* U praznoj listi nema takvog mesta i vraca se NULL. */
    if (glava == NULL)
109         return NULL;

111     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
    pokazivala na cvor ciji je sledeci ili ne postoji ili ima
113     vrednost vecu ili jednaku vrednosti novog cvora.

115     Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
    provera da li se doslo do poslednjeg cvora liste pre nego sto se
117     proveru vrednost njegovog sledeceg cvora, jer u slucaju
    poslednjeg, sledeci ne postoji, pa ni njegova vrednost. */
119     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
        glava = glava->sledeci;

121
    /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
    poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
    vecu od broj. */
123
    return glava;
125 }
127
```



```

129 /* Funkcija uvezuje cvor novi iza cvora tekuci. */
void dodaj_iza(Cvor * tekuci, Cvor * novi)
{
131     /* Novi cvor se dodaje iza tekuceg cvora. */
    novi->sledeci = tekuci->sledeci;
133     tekuci->sledeci = novi;
}

135
/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
137 sortirana. Vraca 1 ukoliko je bilo greski pri alokaciji memorije,
    inace vraca 0. */
139 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
141     /* U slucaju prazne liste glava nove liste je novi cvor. */
    if (*adresa_glave == NULL) {
143         Cvor *novi = napravi_cvor(broj);
        if (novi == NULL)
145             return 1;
        *adresa_glave = novi;
147         return 0;
    }

149
    /* Lista nije prazna. */
151     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
        treba dodati na pocetak liste. */
153     if ((*adresa_glave)->vrednost >= broj) {
        return dodaj_na_pocetak_liste(adresa_glave, broj);
155     }

157
    /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
        tada se pronalazi cvor liste iza koga treba da se umetne nov
159 broj. */
    Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
161     Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
163         return 1;

165
    /* Uvezuje se novi cvor iza pomocnog. */
    dodaj_iza(pomocni, novi);
167     return 0;
}

169
/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
171 Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
    NULL u slucaju da takav cvor ne postoji u listi. */
173 Cvor *pretrazi_listu(Cvor * glava, int broj)
{
175     for (; glava != NULL; glava = glava->sledeci)
        if (glava->vrednost == broj)
177         return glava;

179
    /* Nema traženog broja u listi i vraca se NULL. */

```

```

    return NULL;
181 }

183 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
185 NULL u slučaju da takav cvor ne postoji u listi. Funkcija se u
    pretraži oslanja na činjenicu da je lista koja se pretražuje
187 neopadajuće sortirana. */
Cvor *pretraži_sortiranu_listu(Cvor * glava, int broj)
189 {
    /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
191 for (; glava != NULL && glava->vrednost <= broj;
        glava = glava->sledeci)
193     if (glava->vrednost == broj)
        return glava;
195
    return NULL;
197 }

199 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
    Funkcija azurira pokazivac na glavu liste, koji može biti
201 promenjen u slučaju da se obriše stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, int broj)
203 {
    Cvor *tekuci = NULL;
205 Cvor *pomocni = NULL;

207 /* Sa početka liste se brisu svi cvorovi koji su jednaki datom
    broju, i azurira se pokazivac na glavu liste. */
209 while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
        /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
211 adresi adresa_glave. */
        pomocni = (*adresa_glave)->sledeci;
213 free(*adresa_glave);
        *adresa_glave = pomocni;
215 }

217 /* Ako je nakon toga lista ostala prazna, izlazi se iz funkcije. */
    if (*adresa_glave == NULL)
219         return;

221 /* Od ovog trenutka, u svakoj iteraciji petlje tekuci pokazuje na
    cvor cija vrednost je razlicita od traženog broja. Isto vazi i
223 za sve cvorove levo od tekuceg. Poredi se vrednost sledeceg
    cvora (ako postoji) sa traženim brojem. Cvor se briše ako je
225 jednak, ili, ako je razlicit, prelazi se na sledeci cvor. Ovaj
    postupak se ponavlja dok se ne dodje do poslednjeg cvora. */
227 tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL)
229     if (tekuci->sledeci->vrednost == broj) {
        /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa

```

```

231     prvo cuva u pomocni. */
    pomocni = tekuci->sledeci;
233     /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
        njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
235     sledeci od ovog cvor koji se brise. */
    tekuci->sledeci = pomocni->sledeci;
237     /* Sada treba osloboditi cvor sa vrednoscu broj. */
    free(pomocni);
239 } else {
    /* Ne treba brisati sledeceg od tekuceg i pokazivac se pomera
241     dalje. */
    tekuci = tekuci->sledeci;
243 }
    return;
245 }

247 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
249    neopadajuće. Funkcija azurira pokazivac na glavu liste, koji moze
    biti promenjen ukoliko se obrise stara glava liste. */
251 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
253     Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;
255
    /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
257     broju, i azurira se pokazivac na glavu liste. */
    while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
259         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
            adresi adresa_glave. */
261         pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
263         *adresa_glave = pomocni;
    }
265
    /* Ako je nakon toga lista ostala prazna, funkcija se prekida.
267     Isto se radi i ukoliko glava liste sadrzi vrednost koja je veca
    od broja, jer kako je lista sortirana rastuce nema potrebe broj
269     traziti u repu liste. */
    if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
271         return;

273     /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
        na cvor cija vrednost je manja od trazenog broja, kao i svim
275     cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
    je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
277     ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
    cija vrednost je veca od trazenog broja. */
279     tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
    )

```

## 4 Dinamičke strukture podataka

```
281     if (tekuci->sledeci->vrednost == broj) {
282         pomocni = tekuci->sledeci;
283         tekuci->sledeci = tekuci->sledeci->sledeci;
284         free(pomocni);
285     } else {
286         /* Ne treba brisati sledeceg od tekuceg, jer je manji od
287            trazenog i prelazi se na sledeci. */
288         tekuci = tekuci->sledeci;
289     }
290     return;
291 }

293 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
294    liste. Ne salje joj se adresa promenljive koja cuva glavu liste,
295    jer ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
296    pokazivac na glavu liste iz pozivajuće funkcije. */
297 void ispisi_listu(Cvor * glava)
298 {
299     putchar('[');
300     for (; glava != NULL; glava = glava->sledeci) {
301         printf("%d", glava->vrednost);
302         if (glava->sledeci != NULL)
303             printf(", ");
304     }
305     printf("]\n");
306 }
307 }
```

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

5  /* 1) Glavni program */
6  int main()
7  {
8      /* Lista je prazna na pocetku. */
9      Cvor *glava = NULL;
10     Cvor *trazeni = NULL;
11     int broj;

13     /* Testiranje dodavanja novog broja na pocetak liste. */
14     printf("Unesite brojeve: (za kraj CTRL+D)\n");
15     while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1 onda je bilo greske pri alokaciji
17            memorije za nov cvor. Memoriju alociranu za cvorove liste
18            treba osloboditi pre napustanja programa. */
19         if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
20             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21             oslobodi_listu(&glava);
22             exit(EXIT_FAILURE);
23         }
24         printf("\tLista: ");
```

```
25     ispisi_listu(glava);
26 }
27
28 printf("\nUnesite broj koji se trazi: ");
29 scanf("%d", &broj);
30
31 trazeni = pretrazi_listu(glava, broj);
32 if (trazeni == NULL)
33     printf("Broj %d se ne nalazi u listi!\n", broj);
34 else
35     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
36
37 oslobodi_listu(&glava);
38
39 return 0;
40 }
```

```
41 #include <stdio.h>
42 #include <stdlib.h>
43 #include "lista.h"
44
45 /* 2) Glavni program */
46 int main()
47 {
48     Cvor *glava = NULL;
49     int broj;
50
51     /* Testiranje dodavanja novog broja na kraj liste. */
52     printf("Unesite brojeve: (za kraj CTRL+D)\n");
53     while (scanf("%d", &broj) > 0) {
54         /* Ako je funkcija vratila 1 onda je bilo greske pri alokaciji
55            memorije za nov cvor. Memoriju alociranu za cvorove liste
56            treba osloboditi pre napustanja programa. */
57         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
58             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
59             oslobodi_listu(&glava);
60             exit(EXIT_FAILURE);
61         }
62         printf("\tLista: ");
63         ispisi_listu(glava);
64     }
65
66     printf("\nUnesite broj koji se brise: ");
67     scanf("%d", &broj);
68
69     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
70        procitanom sa ulaza */
71     obrisi_cvor(&glava, broj);
72
73     printf("Lista nakon brisanja: ");
74     ispisi_listu(glava);
75 }
```

```
36  oslobodi_listu(&glava);
38  return 0;
}
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  /* 3) Glavni program */
6  int main()
7  {
8      Cvor *glava = NULL;
9      Cvor *trazeni = NULL;
10     int broj;
11
12     /* Testira se dodavanje u listu tako da ona bude neopadajuće
13        uredjena */
14     printf("Unosite brojeve (za kraj CTRL+D)\n");
15     while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1 onda je bilo greske pri alokaciji
17            memorije za nov cvor. Memoriju alociranu za cvorove liste
18            treba osloboditi pre napustanja programa. */
19         if (dodaj_sortirano(&glava, broj) == 1) {
20             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21             oslobodi_listu(&glava);
22             exit(EXIT_FAILURE);
23         }
24         printf("\tLista: ");
25         ispisi_listu(glava);
26     }
27
28     printf("\nUnesite broj koji se trazi: ");
29     scanf("%d", &broj);
30
31     trazeni = pretrazi_listu(glava, broj);
32     if (trazeni == NULL)
33         printf("Broj %d se ne nalazi u listi!\n", broj);
34     else
35         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
36
37     printf("\nUnesite broj koji se brise: ");
38     scanf("%d", &broj);
39
40     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
41        procitanom sa ulaza */
42     obrisi_cvor_sortirane_liste(&glava, broj);
43
44     printf("Lista nakon brisanja: ");
45     ispisi_listu(glava);
46
47     oslobodi_listu(&glava);
48 }
```

```

49     return 0;
}

```

## Rešenje 4.2

```

1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
6  typedef struct cvor {
8      int vrednost;
9      struct cvor *sledeci;
10 } Cvor;

12 Cvor *napravi_cvor(int broj);

14 void oslobodi_listu(Cvor ** adresa_glave);

16 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

18 Cvor *pronadji_poslednji(Cvor * glava);

20 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

22 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

24 void dodaj_iza(Cvor * tekuci, Cvor * novi);

26 int dodaj_sortirano(Cvor ** adresa_glave, int broj);

28 Cvor *pretrazi_listu(Cvor * glava, int broj);

30 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

32 void obrisi_cvor(Cvor ** adresa_glave, int broj);

34 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

36 void ispisi_vrednosti(Cvor * glava);

38 void ispisi_listu(Cvor * glava);

#endif

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

```

```

5  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog cvora
   * inicijalizuje na broj, dok pokazivac na sledeci cvor u novom cvoru
   * postavlja na NULL. Funkcija vraća pokazivac na novokreirani cvor
   * ili NULL ako alokacija nije uspesno izvršena. */
9  Cvor *napravi_cvor(int broj)
   {
11     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
13         return NULL;

15     novi->vrednost = broj;
       novi->sledeci = NULL;
17     return novi;
   }

19     /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   * ciji se pocetni cvor nalazi na adresi adresa_glave. */
21     void oslobodi_listu(Cvor ** adresa_glave)
   {
23         /* Lista je vec prazna */
25         if (*adresa_glave == NULL)
           return;

27         /* Ako lista nije prazna, treba osloboditi memoriju. Pre
29         oslobadjanja memorije za glavu liste, treba osloboditi rep
           liste. */
31         oslobodi_listu(&(*adresa_glave)->sledeci);
           /* Nakon oslobodjenog repa, oslobadja se glava liste, i azurira se
33         glava u pozivajucoj funkciji tako da odgovara praznoj listi */
           free(*adresa_glave);
35         *adresa_glave = NULL;
   }

37     /* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor
39     koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak.
       Funkcija vraća 1 ukoliko je doslo do greske pri alokaciji, inace
41     vraća 0. */
       int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
   {
43         /* Kreira se nov cvor i proverava se da li je bilo greske pri
           alokaciji */
45         Cvor *novi = napravi_cvor(broj);
47         if (novi == NULL)
           return 1;

49         /* Novi cvor se uvezuje na pocetak, i tako postaje nova glave liste
           */
51         novi->sledeci = *adresa_glave;
53         *adresa_glave = novi;
           return 0;
55     }

```



```
57 /* Funkcija dodaje novi cvor na kraj liste. Prilikom dodavanja u
58    listu na kraj u velikoj vecini slucajeva nov broj se dodaje u rep
59    liste u rekurzivnom pozivu. U slucaju da je u rekurzivnom pozivu
60    doslo do greske pri alokaciji, funkcija vraca 1 visem rekurzivnom
61    pozivu koji tu informaciju vraca u rekurzivni poziv iznad, sve dok
62    se ne vrati u main. Ako je funkcija vratila 0, onda nije bilo
63    greske. Tek je iz main funkcije moguće pristupiti pravom pocetku
64    liste i osloboditi je celu, ako ima potrebe. */
65 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
66 {
67     if (*adresa_glave == NULL) {
68         /* Glava liste je upravo novi cvor i ujedno i cela lista. */
69         Cvor *novi = napravi_cvor(broj);
70         if (novi == NULL)
71             return 1;
72
73         /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
74            azurira i pokazivacka promenljiva u pozivajucoj funkciji. */
75         *adresa_glave = novi;
76         return 0;
77     }
78
79     /* Ako lista nije prazna, broj se dodaje u rep liste. */
80     return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
81 }
82
83 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
84    ostane sortirana. Vraca 0, ako je alokacija novog cvora prosla
85    bez greske, inace vraca 1 da bi ta vrednost bila propagirala nazad
86    do prvog poziva. */
87 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
88 {
89     /* U slucaju prazne liste adresa_glave nove liste je upravo novi
90        cvor. */
91     if (*adresa_glave == NULL) {
92         Cvor *novi = napravi_cvor(broj);
93         if (novi == NULL)
94             return 1;
95
96         *adresa_glave = novi;
97         return 0;
98     }
99
100    /* Lista nije prazna. Ako je broj manji ili jednak vrednosti u
101       glavi liste, onda se dodaje na pocetak liste i vraca se
102       informacija o uspesnosti alokacije. */
103    if ((*adresa_glave)->vrednost >= broj)
104        return dodaj_na_pocetak_liste(adresa_glave, broj);
105
106    /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
107       bude sortirana lista. */
108    return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
```

```
109 }

111 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
112    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
113    NULL u slučaju da takav cvor ne postoji u listi. */
114 Cvor *pretrazi_listu(Cvor * glava, int broj)
115 {
116     /* U praznoj listi ga sigurno nema */
117     if (glava == NULL)
118         return NULL;

119     /* Ako glava liste sadrži traženi broj */
121     if (glava->vrednost == broj)
122         return glava;

123     /* Ako nije nijedna od prethodnih situacija, pretraga se nastavlja
124        u repu liste. */
125     return pretrazi_listu(glava->sledeci, broj);
127 }

129 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
130    Funkcija se u pretrazi oslanja na činjenicu da je lista koja se
131    pretražuje neopadajuće sortirana. Vraca pokazivac na cvor liste u
132    kome je sadržan traženi broj ili NULL u slučaju da takav cvor ne
133    postoji u listi. */
134 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
135 {
136     /* Traženog broja nema ako je lista prazna ili ako je broj manji od
137        vrednosti u glavi liste, jer je lista neopadajuće sortirana. */
138     if (glava == NULL || glava->vrednost > broj)
139         return NULL;

141     /* Ako glava liste sadrži traženi broj, vraća se glava. */
142     if (glava->vrednost == broj)
143         return glava;

145     /* Ako nije nijedna od prethodnih situacija, pretraga se nastavlja
146        u repu. */
147     return pretrazi_listu(glava->sledeci, broj);
149 }

151 /* Funkcija briše iz liste sve cvorove koji sadrže dati broj.
152    Funkcija azurira pokazivac na glavu liste, koji može biti
153    promenjen u slučaju da se obriše stara glava liste. */
154 void obrisi_cvor(Cvor ** adresa_glave, int broj)
155 {
156     /* U praznoj listi, nema cvorova za brisanje. */
157     if (*adresa_glave == NULL)
158         return;

159     /* Prvo se brišu cvorovi iz repa koji imaju vrednost broj. */
160     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
```

```

161  /* Preostaje provera da li glavu liste treba obrisati. */
163  if ((*adresa_glave)->vrednost == broj) {
164      /* pomocni pokazuje na cvor koji treba da se obrise. */
165      Cvor *pomocni = *adresa_glave;
166      /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
167         brise se cvor koji je bio glava liste. */
168      *adresa_glave = (*adresa_glave)->sledeci;
169      free(pomocni);
170  }
171  }

173  /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
174     oslanjajuci se na cinjenicu da je prosledjena lista sortirana
175     neopadajuće. Funkcija azurira pokazivac na glavu liste, koji moze
176     biti promenjen ukoliko se obrise stara glava liste. */
177  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
178  {
179      /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
180         veca od broja, kako je lista sortirana rastuce nema potrebe
181         broj traziti u repu liste i zato se funkcija prekida. */
182      if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
183          return;

185      /* Brisu se cvorovi iz repa koji imaju vrednost broj. */
186      obrisi_cvor(&(*adresa_glave)->sledeci, broj);

187      /* Preostaje provera da li glavu liste treba obrisati. */
188      if ((*adresa_glave)->vrednost == broj) {
189          /* pomocni pokazuje na cvor koji treba da se obrise. */
190          Cvor *pomocni = *adresa_glave;
191          /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
192             brise se cvor koji je bio glava liste. */
193          *adresa_glave = (*adresa_glave)->sledeci;
194          free(pomocni);
195      }
196  }

197  }

199  /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
200     zapetama. */
201  void ispisi_vrednosti(Cvor * glava)
202  {
203      /* Prazna lista */
204      if (glava == NULL)
205          return;

207      /* Ispisuje se vrednost u glavi liste. */
208      printf("%d", glava->vrednost);

209      /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
210         se poziva ista funkcija za ispis ostalih. */
211      if (glava->sledeci != NULL) {

```

```
213     printf(", ");
        ispisi_vrednosti(glava->sledeci);
215 }
    }
217
219 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
    liste. Ne salje joj se adresa promenljive koja cuva glavu liste,
    jer ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
221 pokazivac na glavu liste iz pozivajuće funkcije. Ova funkcija
    ispisuje samo zagrade, a rekurzivno ispisivanje vrednosti u listi
223 prepusta rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce
    ispisati elemente razdvojene zapetom i razmakom. */
225 void ispisi_listu(Cvor * glava)
    {
227     putchar('[');
        ispisi_vrednosti(glava);
229     printf("]\n");
    }
```

### Rešenje 4.3

```

1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
    podatak vrednost i pokazivace na sledeci i prethodni cvor liste.
    */
6  typedef struct cvor {
    int vrednost;
8     struct cvor *sledeci;
    struct cvor *prethodni;
10 } Cvor;

12 Cvor *napravi_cvor(int broj);

14 void oslobodi_listu(Cvor ** adresa_glave);

16 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

18 Cvor *pronadji_poslednji(Cvor * glava);

20 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

22 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

24 void dodaj_iza(Cvor * tekuci, Cvor * novi);

26 int dodaj_sortirano(Cvor ** adresa_glave, int broj);

28 Cvor *pretrazi_listu(Cvor * glava, int broj);
```

```

30 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
32 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci);
34 void obrisi_cvor(Cvor ** adresa_glave, int broj);
36 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
38 void ispisi_listu(Cvor * glava);
40 void ispisi_listu_unazad(Cvor * glava);
42 #endif

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

/* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog cvora
inicijalizuje na broj, dok pokazivac na sledeci cvor u novom cvoru
postavlja na NULL. Funkcija vraća pokazivac na novokreirani cvor
ili NULL ako alokacija nije uspesno izvršena. */
Cvor *napravi_cvor(int broj)
{
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;

    novi->vrednost = broj;
    novi->sledeci = NULL;
    return novi;
}

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave)
{
    Cvor *pomocni = NULL;

    /* Ako lista nije prazna, onda treba osloboditi memoriju. */
    while (*adresa_glave != NULL) {
        /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
osloboditi memoriju cvora koji predstavlja glavu liste. */
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        /* Sledeci cvor je nova glava liste. */
        *adresa_glave = pomocni;
    }
}

/* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor
koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak. Vraca

```

```
1  ukoliko je bilo greski pri alokaciji memorije, inace vraca 0. */
40 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
42     Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
44         return 1;

46     /* Sledbenik novog cvora je glava stare liste */
    novi->sledeci = *adresa_glave;
48     /* Ako stara lista nije bila prazna, onda prethodni od glave treba
        da bude nov cvor. */
50     if (*adresa_glave != NULL)
        (*adresa_glave)->prethodni = novi;
52     /* Novi cvor je nova glava liste. */
    *adresa_glave = novi;
54
56     return 0;
}

58 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
    NULL ukoliko je lista prazna. */
60 Cvor *pronadji_poslednji(Cvor * glava)
{
62     /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
    if (glava == NULL)
64         return NULL;

66     /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pokazivac
        glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
68         ce pokazivati na cvor liste koji nema sledeceg, tj, poslednji je
        cvor liste, vraca se vrednost pokazivaca glava. Pokazivac glava
70         je argument funkcije i njegove promene nece se odraziti na
        vrednost pokazivaca glava u pozivajucoj funkciji. */
72     while (glava->sledeci != NULL)
        glava = glava->sledeci;
74
76     return glava;
}

78 /* Funkcija dodaje broj na kraj liste. Ukoliko dodje do greske pri
    alokaciji memorije vratice 1, inace vraca 0. */
80 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
82     Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
84         return 1;

86     /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
        ujedno i cela lista. Azurira se vrednost na koju pokazuje
88         adresa_glave i tako se azurira i pokazivacka promenljivu u
        pozivajucoj funkciji. */
90     if (*adresa_glave == NULL) {
```

```

    *adresa_glave = novi;
92     return 0;
    }

94
    /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
96     se dodaje na kraj liste kao sledbenik poslednjeg. */
    Cvor *poslednji = pronadji_poslednji(*adresa_glave);
98     poslednji->sledeci = novi;
    novi->prethodni = poslednji;

100
    return 0;
102 }

104 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
    nov cvor sa vrednoscu broj. */
106 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
    {
108     /* U praznoj listi nema takvog mesta i vraća se NULL. */
    if (glava == NULL)
110         return NULL;

112     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
        pokazivala na cvor ciji je sledeci ili ne postoji ili ima
114         vrednost vecu ili jednaku vrednosti novog cvora.

        Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
        provera da li se doslo do poslednjeg cvora liste pre nego sto se
118         proveri vrednost njegovog sledeceg cvora, jer u slucaju
        poslednjeg, sledeci ne postoji, pa ni njegova vrednost. */
120     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
        glava = glava->sledeci;

122
        /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
124         poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
        vecu od broj. */
126     return glava;
    }

128
    /* Funkcija uvezuje cvor novi iza cvora tekuci. */
130 void dodaj_iza(Cvor * tekuci, Cvor * novi)
    {
132     novi->sledeci = tekuci->sledeci;
    novi->prethodni = tekuci;

134
    /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik i
136     tekuci dobija novog sledeceg postavljanjem pokazivaca na
    ispravne adrese. */
138     if (tekuci->sledeci != NULL)
        tekuci->sledeci->prethodni = novi;
140     tekuci->sledeci = novi;
    }
142

```

```

144  /* Funkcija dodaje u listu nov cvor na odgovarajuće mesto, tako sto
    pronalazi cvor u listi iza kod treba uvezati nov cvor. Ukoliko
    dodje do greske pri alokaciji memorije vratiće 1, inace vraca 0.
    */
146  int dodaj_sortirano(Cvor ** adresa_glave, int broj)
    {
148      /* Ako je lista prazna, glava nove liste je novi cvor. */
      if (*adresa_glave == NULL) {
150          Cvor *novi = napravi_cvor(broj);
          if (novi == NULL)
152              return 1;
          *adresa_glave = novi;
154          return 0;
      }

156      /* Ukoliko je vrednost glave liste veca ili jednaka od nove
    vrednosti onda nov cvor treba staviti na pocetak liste. */
158      if ((*adresa_glave)->vrednost >= broj) {
160          dodaj_na_pocetak_liste(adresa_glave, broj);
          return 0;
162      }

164      Cvor *novi = napravi_cvor(broj);
      if (novi == NULL)
166          return 1;

168      /* Pronazi se cvor iza koga treba uveziti nov cvor. */
      Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
170      dodaj_iza(pomocni, novi);

172      return 0;
    }
174

176  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
    NULL u slučaju da takav cvor ne postoji u listi. */
178  Cvor *pretrazi_listu(Cvor * glava, int broj)
    {
180      for (; glava != NULL; glava = glava->sledeci)
          if (glava->vrednost == broj)
182              return glava;

184      /* Nema traženog broja u listi i vraca se NULL. */
      return NULL;
186  }

188  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Funkcija se u pretrazi oslanja na činjenicu da je lista koja se
    pretražuje neopadajuće sortirana. Vraca pokazivac na cvor liste u
    kome je sadržan traženi broj ili NULL u slučaju da takav cvor ne
    postoji u listi. */
190
192  Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)

```



```

194 {
195     /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
196     for (; glava != NULL && glava->vrednost <= broj;
197           glava = glava->sledeci)
198         if (glava->vrednost == broj)
199             return glava;
200
201     /* Nema trazenog broja u listi i bice vrateno NULL. */
202     return NULL;
203 }
204
205 /* Funkcija brise u listi na koju pokazuje pokazivac glava onaj cvor
206    na koji pokazuje pokazivac tekuci. Obratiti paznju da je kod
207    dvostruke liste ovo mnogo lakse uraditi jer cvor tekuci sadrzi
208    pokazivace na svog sledbenika i prethodnika u listi. */
209 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
210 {
211     /* Ako je tekuci NULL pokazivac, nema sta da se brise. */
212     if (tekuci == NULL)
213         return;
214
215     /* Ako postoji prethodnik od tekuceg, onda se postavlja da njegov
216        sledeci bude sledeci od tekuceg. */
217     if (tekuci->prethodni != NULL)
218         tekuci->prethodni->sledeci = tekuci->sledeci;
219
220     /* Ako postoji sledbenik tekuceg (cvora koji se brise), onda njegov
221        prethodnik treba da bude prethodnik tekuceg. */
222     if (tekuci->sledeci != NULL)
223         tekuci->sledeci->prethodni = tekuci->prethodni;
224
225     /* Ako je glava cvor koji se brise, glava nove liste ce biti
226        sledbenik od stare glave. */
227     if (tekuci == *adresa_glave)
228         *adresa_glave = tekuci->sledeci;
229
230     /* Oslobadja se dinamicki alociran prostor za cvor tekuci. */
231     free(tekuci);
232 }
233
234 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
235    Funkcija azurira pokazivac na glavu liste, koji može biti
236    promenjen u slučaju da se obriše stara glava. */
237 void obrisi_cvor(Cvor ** adresa_glave, int broj)
238 {
239     Cvor *tekuci = *adresa_glave;
240
241     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
242         obrisi_tekuci(adresa_glave, tekuci);
243 }
244
245 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,

```

```
246     oslanjajući se na činjenicu da je prosledjena lista neopadajuće
248     sortirana. Funkcija azurira pokazivač na glavu liste, koji može
    biti promenjen ukoliko se obriše stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
250 {
    Cvor *tekuci = *adresa_glave;
252
    while ((tekuci =
254         pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, tekuci);
256 }

258 /* Funkcija prikazuje cvorove liste počev od glave ka kraju liste. Ne
    šalje joj se adresa promenljive koja čuva glavu liste, jer ova
260     funkcija neće menjati listu, pa nema ni potrebe da azurira
    pokazivač na glavu liste iz pozivajuće funkcije. */
262 void ispisi_listu(Cvor * glava)
    {
264         putchar('[');
        for (; glava != NULL; glava = glava->sledeci) {
266             printf("%d", glava->vrednost);
                if (glava->sledeci != NULL)
268                 printf(", ");
        }

270         printf("]\n");
272     }

274 /* Funkcija prikazuje cvorove liste počev od kraja ka glavi liste. */
void ispisi_listu_unazad(Cvor * glava)
276 {
    putchar('[');
278     if (glava == NULL) {
        printf("]\n");
280         return;
    }

282     glava = pronadji_poslednji(glava);

284     for (; glava != NULL; glava = glava->prethodni) {
        printf("%d", glava->vrednost);
286         if (glava->prethodni != NULL)
            printf(", ");
288     }
    printf("]\n");
290 }

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

5  /* 1) Glavni program */
```

```

7  int main()
{
    /* Lista je prazna na pocetku. */
9   Cvor *glava = NULL;
    Cvor *trazeni = NULL;
11  int broj;

13  /* Testiranje dodavanja novog broja na pocetak liste. */
    printf("Unesite brojeve: (za kraj CTRL+D)\n");
15  while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1 onda je bilo greske pri alokaciji
17         memorije za nov cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa. */
19         if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
                fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21                 oslobodi_listu(&glava);
                exit(EXIT_FAILURE);
23             }
            printf("\tLista: ");
25             ispisi_listu(glava);
        }

27     printf("\nUnesite broj koji se trazi u listi: ");
29     scanf("%d", &broj);

31     trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
33         printf("Broj %d se ne nalazi u listi!\n", broj);
    else
35         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

37     printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(glava);
39
    oslobodi_listu(&glava);
41
    return 0;
43 }

```

```

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

5  /* 2) Glavni program */
    int main()
7  {
        Cvor *glava = NULL;
9         int broj;

11        /* Testiranje dodavanja novog broja na kraj liste. */
        printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
13        while (scanf("%d", &broj) > 0) {

```

```

15     /* Ako je funkcija vratila 1 onda je bilo greske pri alokaciji
16         memorije za nov cvor. Memoriju alociranu za cvorove liste
17         treba osloboditi pre napustanja programa. */
18     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
19         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
20         oslobodi_listu(&glava);
21         exit(EXIT_FAILURE);
22     }
23     printf("\tLista: ");
24     ispisi_listu(glava);
25 }
26
27 printf("\nUnesite broj koji se brise iz liste: ");
28 scanf("%d", &broj);
29
30 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
31    procitanom sa ulaza */
32 obrisi_cvor(&glava, broj);
33
34 printf("Lista nakon brisanja: ");
35 ispisi_listu(glava);
36
37 printf("\nLista ispisana u nazad: ");
38 ispisi_listu_unazad(glava);
39
40 oslobodi_listu(&glava);
41
42 return 0;
43 }

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  /* 3) Glavni program */
6  int main()
7  {
8      Cvor *glava = NULL;
9      Cvor *trazeni = NULL;
10     int broj;
11
12     /* Testira se dodavanje u listu tako da ona bude neopadajuće
13        uređena */
14     printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
15     while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1 onda je bilo greske pri alokaciji
17            memorije za nov cvor. Memoriju alociranu za cvorove liste
18            treba osloboditi pre napustanja programa. */
19         if (dodaj_sortirano(&glava, broj) == 1) {
20             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21             oslobodi_listu(&glava);
22             exit(EXIT_FAILURE);

```

```

    }
24     printf("\tLista: ");
    ispisi_listu(glava);
26 }

28     printf("\nUnesite broj koji se trazi u listi: ");
    scanf("%d", &broj);
30
    trazen = pretrazi_listu(glava, broj);
32     if (trazen == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
34     else
        printf("Trazeni broj %d je u listi!\n", trazen->vrednost);
36
    printf("\nUnesite broj koji se brise iz liste: ");
38     scanf("%d", &broj);

40     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
        procitanom sa ulaza */
42     obrisi_cvor_sortirane_liste(&glava, broj);

44     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);
46
    printf("\nLista ispisana u nazad: ");
48     ispisi_listu_unazad(glava);

50     oslobodi_listu(&glava);

52     return 0;
}

```

#### Rešenje 4.4

```

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

4  typedef struct cvor {
    char zagrada;
6     struct cvor *sledeci;
} Cvor;

8

10 int main()
{
    Cvor *stek = NULL;
12     FILE *ulaz = NULL;
    char c;
14     Cvor *pomocni = NULL;

16     ulaz = fopen("izraz.txt", "r");
    if (ulaz == NULL) {

```

```
18     fprintf(stderr,
19         "Greska prilikom otvaranja datoteke izraz.txt!\n");
20     exit(EXIT_FAILURE);
21 }
22
23 while ((c = fgetc(ulaz)) != EOF) {
24     /* Ako je učitana otvorena zagrada, stavlja se na stek. */
25     if (c == '(' || c == '{' || c == '[') {
26         pomocni = (Cvor *) malloc(sizeof(Cvor));
27         if (pomocni == NULL) {
28             fprintf(stderr, "Greska prilikom alokacije memorije!\n");
29             return 1;
30         }
31         pomocni->zagrada = c;
32         pomocni->sledeci = stek;
33         stek = pomocni;
34     }
35     /* Ako je učitana zatvorena zagrada, proverava se da li je stek
36        prazan i ako nije, da li se na vrhu steka nalazi odgovarajuća
37        otvorena zagrada. */
38     else {
39         if (c == ')' || c == '}' || c == ']') {
40             if (stek != NULL && ((stek->zagrada == '(' && c == ')')
41                                 || (stek->zagrada == '{' && c == '}')
42                                 || (stek->zagrada == '[' && c == ']')))
43             {
44                 /* Sa vrha steka se uklanja otvorena zagrada */
45                 pomocni = stek->sledeci;
46                 free(stek);
47                 stek = pomocni;
48             } else {
49                 /* Zagrade u izrazu nisu ispravno uparene. */
50                 break;
51             }
52         }
53     }
54 }
55
56 /* Ako je na kraju stek prazan i procitana je cela datoteka,
57    zagrade su ispravno uparene, u suprotnom, nisu. */
58 if (stek == NULL && c == EOF)
59     printf("Zagrade su ispravno uparene.\n");
60 else {
61     printf("Zagrade nisu ispravno uparene.\n");
62 }
63
64 while (stek != NULL) {
65     /* U slučaju neispravnog uparivanja treba osloboditi memoriju
66        koja je ostala zauzeta stekom. */
67     pomocni = stek->sledeci;
68     free(stek);
69     stek = pomocni;
70 }
```

```

    }
70
    fclose(ulaz);
72    return 0;
    }

```

### Rešenje 4.5

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <ctype.h>
5
   #define MAX 100
7
   #define OTVORENA 1
9   #define ZATVORENA 2
11
   #define VAN_ETIKETE 0
   #define PROCITANO_MANJE 1
13  #define U_ETIKETI 2
15
   /* Struktura kojim se predstavlja cvor liste: sadrzi ime etikete i
      pokazivac na sledeci cvor. */
17  typedef struct cvor {
      char etiketa[MAX];
19      struct cvor *sledeci;
   } Cvor;
21
   /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
      adresu ili NULL ako alokacija nije bila uspesna. */
23  Cvor *napravi_cvor(char *etiketa)
25  {
      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27      if (novi == NULL)
          return NULL;
29
      /* Inicijalizacija polja u novom cvoru */
31      if (strlen(etiketa) >= MAX) {
          fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
33          etiketa[MAX - 1] = '\0';
      }
35      strcpy(novi->etiketa, etiketa);
      novi->sledeci = NULL;
37      return novi;
   }
39
   /* Funkcija prazni stek */
41  void oslobodi_stek(Cvor ** adresa_vrha)
   {
43      Cvor *pomocni;

```

```

45     while (*adresa_vrha != NULL) {
        pomocni = *adresa_vrha;
        *adresa_vrha = (*adresa_vrha)->sledeci;
47     free(pomocni);
    }
49 }

51 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
53 zauzeta memorija za listu ciji pocetni cvor se nalazi na adresi
    adresa_vrha. */
55 void prover_i_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
    {
57     if (novi == NULL) {
        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
59     oslobodi_stek(adresa_vrha);
        exit(EXIT_FAILURE);
61     }
    }

63 /* Funkcija postavlja na vrh steka novu etiketu. */
65 void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
    {
67     Cvor *novi = napravi_cvor(etiketa);
        prover_i_alokaciju(adresa_vrha, novi);
69     novi->sledeci = *adresa_vrha;
        *adresa_vrha = novi;
71 }

73 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
    pokazivac razlicit od NULL, tada u niz karaktera na koji on
75 pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
    suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
77 samim tim nije bilo moguće skinuti vrednost sa steka) ili 1 u
    suprotnom. */
79 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
    {
81     Cvor *pomocni;

83     /* Pokusaj skidanja vrednost sa vrha praznog steka rezultuje
        greskom i vraca se 0. */
85     if (*adresa_vrha == NULL)
        return 0;

87     /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
        adresu kopira etiketa sa vrha steka. */
89     if (etiketa != NULL)
        strcpy(etiketa, (*adresa_vrha)->etiketa);

91     /* Element sa vrha steka se uklanja. */
        pomocni = *adresa_vrha;
93     *adresa_vrha = (*adresa_vrha)->sledeci;
95

```



```

    free(pomocni);
97
    return 1;
99 }

101 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
    steka. Ukoliko je stek prazan, vraca NULL. */
103 char *vrh_steka(Cvor * vrh)
    {
105     if (vrh == NULL)
        return NULL;
107     return vrh->etiketa;
    }

109
111 /* Funkcija prikazuje stek pocev od vrha prema dnu. */
    void prikazi_stek(Cvor * vrh)
    {
113     for (; vrh != NULL; vrh = vrh->sledeci)
        printf("<%s>\n", vrh->etiketa);
115 }

117 /* Funkcija iz fajla na koji pokazuje f cita sledecu etiketu, i njeno
    ime upisuje u niz na koji pokazuje pokazivac etiketa. Funkcija
119 vraca EOF u slucaju da se dodje do kraja fajla pre nego sto se
    procita etiketa, vraca OTVORENA ako je procitana otvorena etiketa,
121 odnosno ZATVORENA ako je procitana zatvorena etiketa. */
    int uzmi_etiketu(FILE * f, char *etiketa)
123 {
    int c;
125     int i = 0;

127     /* Stanje predstavlja informaciju dokle se stalo sa citanjem
    etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
129 nije zapoceto citanje. Tip predstavlja informaciju o tipu
    etikete uzima vrednosti OTVORENA ili ZATVORENA. */
131     int stanje = VAN_ETIKETE;
    int tip;

133
135     /* HTML je neosetljiv na razliku izmedju malih i velikih slova. U
    HTML-u etikete BODY i body imaju isto znacenje, dok to u C-u ne
137 slovima. */
    while ((c = fgetc(f)) != EOF) {
139         switch (stanje) {
            case VAN_ETIKETE:
                if (c == '<')
                    stanje = PROCITANO_MANJE;
141                 break;
            case PROCITANO_MANJE:
                if (c == '/') {
143                     /* Cita se zatvorena etiketa. */
145                     tip = ZATVORENA;
147

```

```
149     } else {
150         if (isalpha(c)) {
151             /* Cita se otvorena etiketa */
152             tip = OTVORENA;
153             etiketa[i++] = tolower(c);
154         }
155         /* Od sada se cita etiketa i zato se menja stanje. */
156         stanje = U_ETIKETI;
157         break;
158     case U_ETIKETI:
159         if (isalpha(c) && i < MAX - 1) {
160             /* Ako je procitani karakter slovo i nije premasena
161              * dozvoljena duzina etikete, procitani karakter se smanjuje
162              * i smesta u etiketu. */
163             etiketa[i++] = tolower(c);
164         } else {
165             /* U suprotnom, staje se sa citanjem etikete i stanje se
166              * menja. Korektno se zavrшава niska koja sadrzi procitanu
167              * etiketu i vraća se njen tip. */
168             stanje = VAN_ETIKETE;
169             etiketa[i] = '\0';
170             return tip;
171         }
172         break;
173     }
174 }
175 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
176  * etiketa i vraća se EOF. */
177 return EOF;
178 }
179
180 int main(int argc, char **argv)
181 {
182     /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
183      * nije procitana. */
184     Cvor *vrh = NULL;
185     char etiketa[MAX];
186     int tip;
187     int uparene = 1;
188     FILE *f = NULL;
189
190     /* Ime datoteke se preuzima iz komandne linije. */
191     if (argc < 2) {
192         fprintf(stderr, "Korisenje: %s ime_html_datoteke\n", argv[0]);
193         exit(0);
194     }
195
196     /* Datoteka se otvara za citanje. */
197     if ((f = fopen(argv[1], "r")) == NULL) {
198         fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
199                 argv[1]);
```

```

201     exit(1);
202 }
203
204 /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
205 while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
206     /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
207        etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
208        potrebno zatvoriti. NAPOMENA: U HTML-u postoje jos neke
209        etikete koje nemaju sadrzaj (npr link). Zbog
210        jednostavnosti pretpostavlja se da njih nema u HTML dokumentu.
211        */
212     if (tip == OTVORENA) {
213         if (strcmp(etiketa, "br") != 0
214             && strcmp(etiketa, "hr") != 0
215             && strcmp(etiketa, "meta") != 0)
216             potisni_na_stek(&vrh, etiketa);
217     }
218     /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
219        u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
220        uvek nije zatvorena. Ova etiketa se mora nalaziti na vrhu
221        steka. Ako je taj uslov ispunjen, skida se sa steka, jer je
222        zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
223        nisu pravilno uparene. */
224     else if (tip == ZATVORENA) {
225         if (vrh_steka(vrh) != NULL
226             && strcmp(vrh_steka(vrh), etiketa) == 0)
227             skini_sa_steka(&vrh, NULL);
228         else {
229             printf("Etikete nisu pravilno uparene\n");
230             printf("(nadjena je etiketa </%s>", etiketa);
231             if (vrh_steka(vrh) != NULL)
232                 printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh));
233             else
234                 printf(" koja nije otvorena)\n");
235             uparene = 0;
236             break;
237         }
238     }
239 }
240 /* Zavrшено je citanje datoteke i zatvara se. */
241 fclose(f);
242
243 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
244    da bude prazan. Ukoliko nije, tada postoje etikete koje su
245    ostale otvorene. */
246 if (uparene) {
247     if (vrh_steka(vrh) == NULL)
248         printf("Etikete su pravilno uparene!\n");
249     else {
250         printf("Etikete nisu pravilno uparene\n");
251         printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));

```

## 4 Dinamičke strukture podataka

---

```
251         /* Oslobadja se memorija zauzeta stekom. */
           oslobodi_stek(&vrh);
253     }
       }
255     return 0;
   }
```

### Rešenje 4.6

```
1  #ifndef _RED_H
   #define _RED_H
3
   #include <stdio.h>
5   #include <stdlib.h>
7
   #define MAX 1000
   #define JMBG_DUZINA 14
9
   /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11  opis njegovog zahteva. */
   typedef struct {
13     char jmbg[JMBG_DUZINA];
       char opis[MAX];
15 } Zahtev;
17
   /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
       korisnika i pokazivac na sledeci cvor liste. */
19  typedef struct cvor {
       Zahtev nalog;
21     struct cvor *sledeci;
   } Cvor;
23
   Cvor *napravi_cvor(Zahtev * zahtev);
25
   void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
27
   void prover_i_alokaciju(Cvor ** adresa_pocetka,
29                        Cvor ** adresa_kraja, Cvor * novi);
31
   void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                   Zahtev * zahtev);
33
   int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
35                  Zahtev * zahtev);
37
   Zahtev *pocetak_reda(Cvor * pocetak);
39
   void prikazi_red(Cvor * pocetak);
41 #endif
```

```

1  #include "red.h"

3  /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
   poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
5  greske pri alokaciji. Ako je doslo do greske, trebalo bi
   osloboditi ceo red. To je ostavljeno da to uradi funkcija koja je
7  pozvala funkciju napravi_cvor, a gresku ce biti signalizirana
   vraćanjem NULL. Funkciji se prosledjuje pokazivac na zahtev koji
9  treba smestiti u nov cvor zbog smestanja manjeg podatka na
   sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
11 bajtovima(B) u odnosu na strukturu Zahtev. */
Cvor *napravi_cvor(Zahtev * zahtev)
13 {
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
15   if (novi == NULL)
       return NULL;

17   novi->nalog = *zahtev;
   novi->sledeci = NULL;
19   return novi;
21 }

23 /* Funkcija prazni red */
void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
25 {
   Cvor *pomocni = NULL;

27   while (*pocetak != NULL) {
       pomocni = *pocetak;
29       *pocetak = (*pocetak)->sledeci;
       free(pomocni);
31   }
   *kraj = NULL;
33 }

35 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
   ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
37 zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
   adresa_pocetka. */
void prover_i_alokaciju(Cvor ** adresa_pocetka,
41                      Cvor ** adresa_kraja, Cvor * novi)
{
43   if (novi == NULL) {
       fprintf(stderr, "Neuspela alokacija za nov cvor\n");
45       oslobodi_red(adresa_pocetka, adresa_kraja);
       exit(EXIT_FAILURE);
47   }
}

49 /* Funkcija dodaje na kraj reda novi fajl. */
void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
51

```

```

                                Zahtev * zahtev)
53 {
    Cvor *novi = napravi_cvor(zahtev);
55   proveri_alokaciju(adresa_pocetka, adresa_kraja, novi);

57   /* U red se uvek dodaje na kraj, ali zbog postojanja pokazivaca na
      kraj, dodavanje na kraj je podjednako efikasno kao dodavanje na
59   pocetak. */
   if (*adresa_kraja != NULL) {
61     (*adresa_kraja)->sledeci = novi;
     *adresa_kraja = novi;
63   } else {
     /* Ako je red bio ranije prazan */
65     *adresa_pocetka = novi;
     *adresa_kraja = novi;
67   }
   }
69 }

/* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
71 pokazivac razlicit od NULL, tada se u strukturu na koju on
   pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
73 suprotnom ne upisuje nista. Funkcija vraca 0 ako je red bio prazan
   ili 1 u suprotnom. */
75 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                                Zahtev * zahtev)
77 {
    Cvor *pomocni = NULL;
79
    if (*adresa_pocetka == NULL)
81       return 0;

83   if (zahtev != NULL)
     *zahtev = (*adresa_pocetka)->nalog;
85
    pomocni = *adresa_pocetka;
87   *adresa_pocetka = (*adresa_pocetka)->sledeci;
    free(pomocni);
89
    if (*adresa_pocetka == NULL)
91       *adresa_kraja = NULL;

93   return 1;
   }
95

/* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev korisnika
97 na pocetku reda. Ukoliko je red prazan, vraca NULL. */
Zahtev *pocetak_reda(Cvor * pocetak)
99 {
    if (pocetak == NULL)
101       return NULL;

103   return &(pocetak->nalog);

```

```

105 }
106 /* Funkcija prikazuje red. */
107 void prikazi_red(Cvor * pocetak)
108 {
109     for (; pocetak != NULL; pocetak = pocetak->sledeci)
110         printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
111     printf("\n");
112 }
113
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include "red.h"
6
7 #define VREME_ZA_PAUZU 5
8
9 int main(int argc, char **argv)
10 {
11     /* Red je prazan. */
12     Cvor *pocetak = NULL, *kraj = NULL;
13     Zahtev nov_zahtev;
14     Zahtev *sledeci = NULL;
15     char odgovor[3];
16     int broj_usluzenih = 0;
17     FILE *izlaz = fopen("izvestaj.txt", "a");
18
19     if (izlaz == NULL) {
20         fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
21         exit(EXIT_FAILURE);
22     }
23
24     /* Sluzbenik evidentira korisnicke zahteve unosnjem njihovog JMBG
25        broja i opisa potrebne usluge. */
26     printf("Sluzbenik evidentira korisnicke zahteve:\n");
27
28     /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
29        JMBG broja procitao i da bi fgets nakon toga procitala ispravan
30        red sa opisom zahteva. */
31     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
32     while (scanf("%s", nov_zahtev.jmbg) != EOF) {
33         getchar();
34         printf("\tOpis problema: ");
35         fgets(nov_zahtev.opis, MAX - 1, stdin);
36         /* Ako je poslednji karakter nov red, eliminise se. */
37         if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
38             nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
39         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
40         printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
41     }

```

```
43  /* Datoteka vise nije potrebna i treba je zatvoriti. */
    fclose(izlaz);

45  /* Dokle god ima korisnika u redu, treba ih usluziti. */
    while (1) {
47      sledeci = pocetak_reda(pocetak);
      /* Ako nema nikog vise u redu, prekida se petlja. */
49      if (sledeci == NULL)
          break;

51      printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
53      printf("i zahtevom: %s\n", sledeci->opis);

55      skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

57      broj_usluzenih++;

59      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
      scanf("%s", odgovor);

61      if (strcmp(odgovor, "Da") == 0)
63          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
      else
65          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
                    nov_zahtev.opis);

67      if (broj_usluzenih == VREME_ZA_PAUZU) {
69          printf("\nDa li je kraj smene? [Da/Ne] ");
          scanf("%s", odgovor);

71          if (strcmp(odgovor, "Da") == 0)
73              break;
          else
75              broj_usluzenih = 0;
      }
77  }

79  /*****
    Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:

81  while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
83      printf("\tSledeci je korisnik sa JMBG brojem: %s\n",
              nov_zahtev.jmbg);
85      printf("sa zahtevom: %s\n", nov_zahtev.opis);
      broj_usluzenih++;

87      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
89      scanf("%s", odgovor);
      if (strcmp(odgovor, "Da") == 0)
91          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
      else
93          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
```



```

    nov_zatev.jmbg, nov_zatev.opis);
95
    if (broj_usluzenih == VREME_ZA_PAUZU) {
97        printf("\nDa li je kraj smene? [Da/Ne] ");
        scanf("%s", odgovor);
99        if (strcmp(odgovor, "Da") == 0)
            break;
101        else
            broj_usluzenih = 0;
103    }
    }
105    *****/

107    /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
        neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
109        koju zauzima red sa neobradjenim zahtevima korisnika. */
    oslobodi_red(&pocetak, &kraj);
111
113    return 0;
}

```

#### Rešenje 4.7

```

#include<stdio.h>
2  #include<string.h>
  #include<stdlib.h>
4  #define MAX_DUZINA 20

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

12 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
    ili NULL ako alokacija nije uspesno izvorsena. */
14 Element *napravi_cvor(unsigned br, char *etiketa)
{
16     Element *novi = (Element *) malloc(sizeof(Element));
    if (novi == NULL)
18         return NULL;

20     novi->broj_pojavljivanja = br;
    strcpy(novi->etiketa, etiketa);
22     novi->sledeci = NULL;
    return novi;
24 }

26 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste.
    */
void oslobodi_listu(Element ** glava)

```

```
28 {
    Element *pomocni = NULL;
30
    while (*glava != NULL) {
32         pomocni = (*glava)->sledeci;
        free(*glava);
34         *glava = pomocni;
    }
36 }

38 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
40 zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
    glava. */
42 void prover_a_lokacije(Element * novi, Element ** glava)
{
44     if (novi == NULL) {
        fprintf(stderr, "malloc() greska u funkciji napravi_cvor(!\n");
46         oslobodi_listu(glava);
        exit(EXIT_FAILURE);
48     }
}

50 /* Funkcija dodaje novi cvor na pocetak liste. */
52 void dodaj_na_pocetak_liste(Element ** glava, unsigned br,
    char *etiketa)
54 {
    Element *novi = napravi_cvor(br, etiketa);
56     prover_a_lokacije(novi, glava);
    novi->sledeci = *glava;
58     *glava = novi;
}

60 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu.
    (NULL u suprotnom) */
62 Element *pretrazi_listu(Element * glava, char etiketa[])
64 {
    Element *tekuci;
66     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
        if (strcmp(tekuci->etiketa, etiketa) == 0)
68         return tekuci;
    return NULL;
70 }

72 /* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Element * glava)
74 {
    for (; glava != NULL; glava = glava->sledeci)
76         printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
}

78 int main(int argc, char **argv)
```

```

80 {
81     if (argc != 2) {
82         fprintf(stderr,
83             "Greska! Program se poziva sa: ./a.out datoteka.html!\n");
84         exit(EXIT_FAILURE);
85     }
86
87     FILE *in = NULL;
88     in = fopen(argv[1], "r");
89     if (in == NULL) {
90         fprintf(stderr,
91             "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
92         exit(EXIT_FAILURE);
93     }
94
95     char c;
96     int i = 0;
97     char a[MAX_DUZINA];
98
99     Element *glava = NULL;
100    Element *trazeni = NULL;
101
102    while ((c = fgetc(in)) != EOF) {
103
104        if (c == '<') {
105            /* Cita se zatvorena etiketa. */
106            if ((c = fgetc(in)) == '/') {
107                i = 0;
108                while ((c = fgetc(in)) != '>')
109                    a[i++] = c;
110            }
111            /* Cita se otvorena etiketa. */
112            else {
113                i = 0;
114                a[i++] = c;
115                while ((c = fgetc(in)) != ' ' && c != '>')
116                    a[i++] = c;
117            }
118            a[i] = '\0';
119
120            /* Trazi se ucitana etiketa medju postojećim cvorovima liste.
121             Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
122             sa brojem pojavljivanja 1, inace uvecava se broj
123             pojavljivanja etikete. */
124            trazeni = pretrazi_listu(glava, a);
125            if (trazeni == NULL)
126                dodaj_na_pocetak_liste(&glava, 1, a);
127            else
128                trazeni->broj_pojavljivanja++;
129        }
130    }

```

```
132     fclose(in);
134     ispisi_listu(glava);
        oslobodi_listu(&glava);
136
        return 0;
138 }
```

### Rešenje 4.8

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

5  #define MAX_INDEKS 11
        #define MAX_IME_PREZIME 21
7
        /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
9         studentu. */
        typedef struct _Cvor {
11         char broj_indeksa[MAX_INDEKS];
            char ime[MAX_IME_PREZIME];
13         char prezime[MAX_IME_PREZIME];
            struct _Cvor *sledeci;
15     } Cvor;

17     /* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac na nov
        cvor ili NULL ukoliko alokacija nije prosla. */
19     Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
        {
21         Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
            if (novi == NULL)
23             return NULL;
            strcpy(novi->broj_indeksa, broj_indeksa);
25             strcpy(novi->ime, ime);
            strcpy(novi->prezime, prezime);
27             novi->sledeci = NULL;
            return novi;
29     }

31     /* Funkcija oslobadja memoriju zauzetu za elemente liste. */
        void oslobodi_listu(Cvor ** glava)
33     {
            if (*glava == NULL)
35                 return;
            oslobodi_listu(&(*glava)->sledeci);
37             free(*glava);
            *glava = NULL;
39     }
```

```

41 void prover_i_alokaciju(Cvor ** glava, Cvor * novi)
42 {
43     if (novi == NULL) {
44         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
45         oslobodi_listu(glava);
46         exit(EXIT_FAILURE);
47     }
48 }
49
50 /* Funkcija dodaje novi cvor na pocetak liste. */
51 void dodaj_na_pocetak_liste(Cvor ** glava, char *broj_indeksa,
52                             char *ime, char *prezime)
53 {
54     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
55     prover_i_alokaciju(glava, novi);
56     novi->sledeci = *glava;
57     *glava = novi;
58 }
59
60 void ispisi_listu(Cvor * glava)
61 {
62     for (; glava != NULL; glava = glava->sledeci)
63         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
64               glava->prezime);
65 }
66
67 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu, u
68    suprotnom vraca NULL. */
69 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
70 {
71     if (glava == NULL)
72         return NULL;
73     if (!strcmp(glava->broj_indeksa, broj_indeksa))
74         return glava;
75     return pretrazi_listu(glava->sledeci, broj_indeksa);
76 }
77
78 int main(int argc, char **argv)
79 {
80     if (argc != 2) {
81         fprintf(stderr, "Greska! Program se poziva sa: ./a.out \
82 student_i.txt!\n");
83         exit(EXIT_FAILURE);
84     }
85
86     FILE *in = NULL;
87     in = fopen(argv[1], "r");
88     if (in == NULL) {
89         fprintf(stderr,
90             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
91         exit(EXIT_FAILURE);
92     }

```

```
93     char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
95     char broj_indeksa[MAX_INDEKS];
96     Cvor *glava = NULL;
97     Cvor *trazeni = NULL;
98
99     /* Ucitavanje vrednosti u listu. */
100    while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
101        dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);
102
103    fclose(in);
104
105    while (scanf("%s", broj_indeksa) != EOF) {
106        trazeni = pretrazi_listu(glava, broj_indeksa);
107        if (trazeni == NULL)
108            printf("ne\n");
109        else
110            printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
111    }
112
113    oslobodi_listu(&glava);
114
115    return 0;
116 }
```

### Rešenje 4.9

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include "lista.h"
4
5  Cvor *objedini(Cvor ** glava1, Cvor ** glava2)
6  {
7      Cvor *l3 = NULL;
8      Cvor **tek = &l3;
9
10     if (*glava1 == NULL && *glava2 == NULL)
11         return NULL;
12
13     /* Ako je prva lista prazna, rezultat je druga lista. */
14     if (*glava1 == NULL)
15         return *glava2;
16
17     /* Ako je druga lista prazna, rezultat je prva lista. */
18     if (*glava2 == NULL)
19         return *glava1;
20
21     /* l3 pokazuje na pocetak nove liste, tj. na manji od brojeva
22        sadrzanih u cvorovima na koje pokazuju glava1 i glava2. */
23     l3 = ((*glava1)->vrednost < (*glava2)->vrednost) ? *glava1 :
24         *glava2;
```

```
25
27 while (*glava1 != NULL && *glava2 != NULL) {
28     if ((*glava1)->vrednost < (*glava2)->vrednost) {
29         *tek = *glava1;
30         *glava1 = (*glava1)->sledeci;
31     } else {
32         *tek = *glava2;
33         *glava2 = (*glava2)->sledeci;
34     }
35     tek = &((*tek)->sledeci);
36 }
37
38 /* Ako se iz petlje izašlo jer se stiglo do kraja prve liste, na
39    rezultujuću listu treba nadovezati ostatak druge liste. */
40 if (*glava1 == NULL)
41     *tek = *glava2;
42
43 else if (*glava2 == NULL)
44     *tek = *glava1;
45
46 return l3;
47 }
48
49 int main(int argc, char **argv)
50 {
51     if (argc != 3) {
52         fprintf(stderr,
53             "Greska! Program se poziva sa: ./a.out dat1.txt dat2.txt
54             !\n");
55         exit(EXIT_FAILURE);
56     }
57
58     FILE *in1 = NULL;
59     in1 = fopen(argv[1], "r");
60     if (in1 == NULL) {
61         fprintf(stderr,
62             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
63         exit(EXIT_FAILURE);
64     }
65
66     FILE *in2 = NULL;
67     in2 = fopen(argv[2], "r");
68     if (in2 == NULL) {
69         fprintf(stderr,
70             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
71         exit(EXIT_FAILURE);
72     }
73
74     int broj;
75     Cvor *glava1 = NULL;
76     Cvor *glava2 = NULL;
```

```

Cvor *l3 = NULL;

77
/* Ucitavanje listi */
79 while (fscanf(in1, "%d", &broj) != EOF)
    dodaj_na_kraj_liste(&glava1, broj);
81 while (fscanf(in2, "%d", &broj) != EOF)
    dodaj_na_kraj_liste(&glava2, broj);
83
l3 = objedini(&glava1, &glava2);

85
/* Ispis rezultujuće liste. */
87 ispisi_listu(l3);
    oslobodi_listu(&l3);
89
fclose(in1);
91 fclose(in2);
    return 0;
93 }
```

### Rešenje 4.14

```

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

4 /* a) Struktura kojom se predstavlja cvor binarnog
    pretrazivackog stabla */
6 typedef struct cvor {
    int broj;
8     struct cvor *levo;
    struct cvor *desno;
10 } Cvor;

12 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
    inicijalizuje polja strukture i vraća pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj)
{
16     /* Alokira se memorija za novi cvor i proverava se uspesnost
        alokacije.*/
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
18     if (novi == NULL)
        return NULL;

20
    /* Inicijalizuju se polja novog cvora. */
22     novi->broj = broj;
    novi->levo = NULL;
24     novi->desno = NULL;

26     /* Vraća se adresa novog cvora. */
    return novi;
28 }
```



```

30 /* Funkcija koja proverava uspesnost kreiranja novog cvora
    stabla */
32 void prover_i_alokaciju(Cvor *novi_cvor)
{
34     /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
36         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
            programa */
38         fprintf(stderr, "Malloc greska za novi cvor!\n");
40         exit(EXIT_FAILURE);
    }
42 }

44 /* c) Funkcija koja dodaje zadati broj u stablo */
void dodaj_u_stablo(Cvor **adresa_korena, int broj)
46 {
    /* Ako je stablo prazno */
48     if (*adresa_korena == NULL) {
        /* Kreira se novi cvor */
        Cvor *novi = napravi_cvor(broj);
        prover_i_alokaciju(novi);
52         /* I proglašava se korenom stabla */
        *adresa_korena = novi;
54         return;
    }

56     /* U suprotnom trazi se odgovarajuca pozicija za zadati broj: */

58     /* Ako je zadata vrednost manja od vrednosti korena */
    if (broj < (*adresa_korena)->broj)
60     {
        /* Broj se dodaje u levo podstablo */
        dodaj_u_stablo(&(*adresa_korena)->levo, broj);
62
64     else
        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa
           se dodaje u desno podstablo */
66         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
68     }

70 /* d) Funkcija koja proverava da li se zadati broj nalazi u
    stablu */
72 Cvor *pretrazi_stablo(Cvor *koren, int broj)
{
74     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
        */
76     if (koren == NULL)
78         return NULL;

80     /* Ako je trazena vrednost sadrzana u korenu */
    if (koren->broj == broj) {

```

```
82     /* Prekidamo pretragu */
83     return koren;
84 }

86 /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
87 if (broj < koren->broj)
88
89     /* Pretraga se nastavlja u levom podstablu */
90     return pretrazi_stablo(koren->levo, broj);
91
92 else
93     /* U suprotnom, pretraga se nastavlja u desnom podstablu */
94     return pretrazi_stablo(koren->desno, broj);
95 }

96 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost u
97    stablu */
98 Cvor *pronadji_najmanji(Cvor * koren)
99 {
100     /* Ako je stablo prazno, prekida se pretraga */
101     if (koren == NULL)
102         return NULL;
103
104     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze
105        se levo od njega */
106
107     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
108        najmanju vrednost */
109     if (koren->levo == NULL)
110         return koren;
111
112     /* Inace, pretragu treba nastaviti u levom podstablu */
113     return pronadji_najmanji(koren->levo);
114 }

115 /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u
116    stablu */
117 Cvor *pronadji_najveci(Cvor * koren)
118 {
119     /* Ako je stablo prazno, prekida se pretraga */
120     if (koren == NULL)
121         return NULL;
122
123     /* Vrednosti koje su veće od vrednosti u korenu stabla nalaze
124        se desno od njega */
125
126     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
127        najveću vrednost */
128     if (koren->desno == NULL)
129         return koren;
130
131     /* Inace, pretragu treba nastaviti u desnom podstablu */
132 }
```

```
134     return pronadji_najveci(koren->desno);
135 }
136
137 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
138 void obrisi_element(Cvor ** adresa_korena, int broj)
139 {
140     Cvor *pomocni_cvor = NULL;
141
142     /* Ako je stablo prazno, brisanje nije primenljivo */
143     if (*adresa_korena == NULL)
144         return;
145
146     /* Ako je vrednost koju treba obrisati manja od vrednosti u
147        korenu stabla, ona se eventualno nalazi u levom podstablu,
148        pa treba rekursivno primeniti postupak na levo podstablo.
149        Koren ovako modifikovanog stabla je nepromenjen. */
150     if (broj < (*adresa_korena)->broj) {
151         obrisi_element(&(*adresa_korena)->levo, broj);
152         return;
153     }
154
155     /* Ako je vrednost koju treba obrisati veca od vrednosti u
156        korenu stabla, ona se eventualno nalazi u desnom podstablu
157        pa treba rekursivno primeniti postupak na desno podstablo.
158        Koren ovako modifikovanog stabla je nepromenjen. */
159     if ((*adresa_korena)->broj < broj) {
160         obrisi_element(&(*adresa_korena)->desno, broj);
161         return;
162     }
163
164     /* Slede podslucajevi vezani za slucaj kada je vrednost u
165        korenu jednaka broju koji se brise (tj. slucaj kada treba
166        obrisati koren) */
167
168     /* Ako koren nema sinova, tada se on prosto brise, i rezultat
169        je prazno stablo (vraca se NULL) */
170     if ((*adresa_korena)->levo == NULL
171         && (*adresa_korena)->desno == NULL) {
172         free(*adresa_korena);
173         *adresa_korena = NULL;
174         return;
175     }
176
177     /* Ako koren ima samo levog sina, tada se brisanje vrši tako
178        sto se brise koren, a novi koren postaje levi sin */
179     if ((*adresa_korena)->levo != NULL
180         && (*adresa_korena)->desno == NULL) {
181         pomocni_cvor = (*adresa_korena)->levo;
182         free(*adresa_korena);
183         *adresa_korena = pomocni_cvor;
184         return;
185     }
186 }
```

```

186  /* Ako koren ima samo desnog sina, tada se brisanje vrši tako
188     sto se briše koren, a novi koren postaje desni sin */
189  if ((*adresa_korena)->desno != NULL
190      && (*adresa_korena)->levo == NULL) {
191      pomocni_cvor = (*adresa_korena)->desno;
192      free(*adresa_korena);
193      *adresa_korena = pomocni_cvor;
194      return;
195  }
196
197  /* Slučaj kada koren ima oba sina - najpre se potraži sledbenik
198     korena (u
199     smislu poretka) u stablu. To je upravo po vrednosti
200     najmanji cvor u desnom podstablu. On se može pronaći npr.
201     funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
202     vrednost tog cvora, a u taj cvor se smesti vrednost korena
203     (tj. broj koji se briše). Zatim se prosto rekurzivno
204     pozove funkcija za brisanje na desno podstablo. S obzirom
205     da u njemu treba obrisati najmanji element, a on zasigurno
206     ima najviše jednog potomka, jasno je da će njegovo brisanje
207     biti obavljeno na jedan od jednostavnijih načina koji su
208     gore opisani. */
209  pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
210  (*adresa_korena)->broj = pomocni_cvor->broj;
211  pomocni_cvor->broj = broj;
212  obrisi_element(&(*adresa_korena)->desno, broj);
213  }
214
215  /* h) Funkcija ispisuje stablo u infiksnoj notaciji ( Levo
216     postablo - Koren - Desno podstablo ) */
217  void ispisi_stablo_infiksno(Cvor * koren)
218  {
219      /* Ako stablo nije prazno */
220      if (koren != NULL) {
221
222          /* Prvo se ispisuju svi cvorovi levo od korena */
223          ispisi_stablo_infiksno(koren->levo);
224
225          /* Zatim se ispisuje vrednost u korenu */
226          printf("%d ", koren->broj);
227
228          /* Na kraju se ispisuju cvorovi desno od korena */
229          ispisi_stablo_infiksno(koren->desno);
230      }
231  }
232
233  /* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren -
234     Levo podstablo - Desno podstablo ) */
235  void ispisi_stablo_prefiksno(Cvor * koren)
236  {
237      /* Ako stablo nije prazno */

```

```

238     if (koren != NULL) {
239
240         /* Prvo se ispisuje vrednost u korenu */
241         printf("%d ", koren->broj);
242
243         /* Zatim se ispisuju svi cvorovi levo od korena */
244         ispisi_stablo_prefiksno(koren->levo);
245
246         /* Na kraju se ispisuju svi cvorovi desno od korena */
247         ispisi_stablo_prefiksno(koren->desno);
248     }
249 }
250
251 /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo
252     podstablo - Desno poststablo - Koren) */
253 void ispisi_stablo_postfiksno(Cvor * koren)
254 {
255     /* Ako stablo nije prazno */
256     if (koren != NULL) {
257
258         /* Prvo se ispisuju svi cvorovi levo od korena */
259         ispisi_stablo_postfiksno(koren->levo);
260
261         /* Zatim se ispisuju svi cvorovi desno od korena */
262         ispisi_stablo_postfiksno(koren->desno);
263
264         /* Na kraju se ispisuje vrednost u korenu */
265         printf("%d ", koren->broj);
266     }
267 }
268
269 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
270 void oslobodi_stablo(Cvor ** adresa_korena)
271 {
272     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
273     if (*adresa_korena == NULL)
274         return;
275
276     /* Inace ... */
277     /* Oslobadja se memorija zauzeta levim podstablom */
278     oslobodi_stablo(&(*adresa_korena)->levo);
279     /* Oslobadja se memorija zauzeta desnim podstablom */
280     oslobodi_stablo(&(*adresa_korena)->desno);
281     /* Oslobadja se memorija zauzeta korenom */
282     free(*adresa_korena);
283     /* Proglasava se stablo praznim */
284     *adresa_korena = NULL;
285 }
286
287 int main()
288 {
289     Cvor *koren;

```

```
int n;
290 Cvor *trazeni_cvor;

292 /* Proglasava se stablo praznim */
koren = NULL;

294 /* Dodaju se vrednosti u stablo */
296 printf("Unesite brojeve (CTRL+D za kraj unosa): ");
while (scanf("%d", &n) != EOF) {
298     dodaj_u_stablo(&koren, n);
}

300 /* Generisu se trazeni ispisi: */
302 printf("\nInfiksni ispis: ");
ispisi_stablo_infiksno(koren);
304 printf("\nPrefiksni ispis: ");
ispisi_stablo_prefiksno(koren);
306 printf("\nPostfiksni ispis: ");
ispisi_stablo_postfiksno(koren);

308 /* Demonstrira se rad funkcije za pretragu */
310 printf("\nTrazi se broj: ");
scanf("%d", &n);
312 trazeni_cvor = pretrazi_stablo(koren, n);
if (trazeni_cvor == NULL)
314     printf("Broj se ne nalazi u stablu!\n");
else
316     printf("Broj se nalazi u stablu!\n");

318 /* Demonstrira se rad funkcije za brisanje */
printf("Briše se broj: ");
320 scanf("%d", &n);
obrisi_element(&koren, n);
322 printf("Rezultujuće stablo: ");
ispisi_stablo_infiksno(koren);
324 printf("\n");

326 /* Oslobadja se memorija zauzeta stablom */
oslobodi_stablo(&koren);

328 return 0;
330 }
```

### Rešenje 4.15

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

6 #define MAX 50
```

```

8  /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
   char *rec;
12  int brojac;
   struct cvor *levo;
14  struct cvor *desno;
} Cvor;

16
/* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec)
{
20  /* Alocira se memorija za novi cvor i proverava se uspesnost
   alokacije. */
22  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
   if (novi_cvor == NULL)
24      return NULL;

26  /* Alocira se memorija za zadatu rec: potrebno je rezervisati
   memoriju za svaki karakter reci ukljucujuci i terminirajucu
   nulu */
28  novi_cvor->rec =
30      (char *) malloc((strlen(rec) + 1) * sizeof(char));
   if (novi_cvor->rec == NULL) {
32      free(novi_cvor);
      return NULL;
34  }

36  /* Inicijalizuju se polja u novom cvoru */
   strcpy(novi_cvor->rec, rec);
38  novi_cvor->brojac = 1;
   novi_cvor->levo = NULL;
40  novi_cvor->desno = NULL;

42  /* Vraca se adresa novog cvora */
   return novi_cvor;
44 }

46 /* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
48 void proveri_alokaciju(Cvor * novi_cvor)
{
50  /* Ukoliko je cvor neuspesno kreiran */
   if (novi_cvor == NULL) {
52      /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
       programa */
54      fprintf(stderr, "Malloc greska za novi cvor!\n");
       exit(EXIT_FAILURE);
56  }
58 }

```

```

60  /* Funkcija koja dodaje novu rec u stablo. */
    void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
    {
62      /* Ako je stablo prazno */
        if (*adresa_korena == NULL) {
64          /* Kreira se cvor koji sadrzi zadatu rec */
            Cvor *novi = napravi_cvor(rec);
66            prover_i_alokaciju(novi);

68            /* I proglašava se korenom stabla */
            *adresa_korena = novi;
70            return;
        }

72      /* U suprotnom se traži odgovarajuća pozicija za novu rec */

74      /* Ako je rec leksikografski manja od reci u korenu ubacuje se
76         u levo podstablo */
        if (strcmp(rec, (*adresa_korena)->rec) < 0)
78            dodaj_u_stablo(&(*adresa_korena)->levo, rec);

80      else
        /* Ako je rec leksikografski veća od reci u korenu ubacuje
82         se u desno podstablo */
        if (strcmp(rec, (*adresa_korena)->rec) > 0)
84            dodaj_u_stablo(&(*adresa_korena)->desno, rec);

86      else
        /* Ako je rec jednaka reci u korenu, uvećava se njen broj
88         pojavljivanja */
        (*adresa_korena)->brojac++;
90    }

92  /* Funkcija koja oslobađa memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** adresa_korena)
    {
94      /* Ako je stablo prazno, nepotrebno je oslobađati memoriju */
        if (*adresa_korena == NULL)
96            return;

98      /* Inace ... */

100     /* Oslobađa se memorija zauzeta levim podstablom */
        oslobodi_stablo(&(*adresa_korena)->levo);

102     /* Oslobađa se memorija zauzeta desnim podstablom */
        oslobodi_stablo(&(*adresa_korena)->desno);

104     /* Oslobađa se memorija zauzeta korenom */
        free((*adresa_korena)->rec);
106        free(*adresa_korena);

108        /* Stablo se proglašava praznim */
110    }

```



```

112     *adresa_korena = NULL;
113 }
114 /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec
115    (rec sa najvećim brojem pojavljivanja) */
116 Cvor *nadjij_najfrekventniju_rec(Cvor * koren)
117 {
118     Cvor *max, *max_levo, *max_desno;
119
120     /* Ako je stablo prazno, prekida se sa pretragom */
121     if (koren == NULL)
122         return NULL;
123
124     /* Pronalazi se najfrekventnija rec u levom podstablu */
125     max_levo = nadjij_najfrekventniju_rec(koren->levo);
126
127     /* Pronalazi se najfrekventnija rec u desnom podstablu */
128     max_desno = nadjij_najfrekventniju_rec(koren->desno);
129
130     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
131        podstabla, korena i desnog podstabla */
132     max = koren;
133     if (max_levo != NULL && max_levo->brojac > max->brojac)
134         max = max_levo;
135     if (max_desno != NULL && max_desno->brojac > max->brojac)
136         max = max_desno;
137
138     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
139     return max;
140 }
141
142 /* Funkcija koja ispisuje reci iz stabla u leksikografskom
143    poretku pracene brojem pojavljivanja */
144 void prikazi_stablo(Cvor * koren)
145 {
146     /* Ako je stablo prazno, završava se sa ispisom */
147     if (koren == NULL)
148         return;
149
150     /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz
151        levog podstabla */
152     prikazi_stablo(koren->levo);
153
154     /* Zatim rec iz korena */
155     printf("%s: %d\n", koren->rec, koren->brojac);
156
157     /* I nastavlja se sa ispisom reci iz desnog podstabla */
158     prikazi_stablo(koren->desno);
159 }
160
161 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje
162    je u niz rec. Maksimalna dužina reci je odredjena argumentom

```

```

164     max. Funkcija vraća EOF ako u datoteci nema više reci ili 0 u
suprotnom. Rec je niz malih ili velikih slova. */
166 int procitaj_rec(FILE * f, char rec[], int max)
167 {
168     /* Karakter koji se cita */
169     int c;
170
171     /* Indeks pozicije na koju se smesta procitani karakter */
172     int i = 0;
173
174     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se
god nije stiglo do kraja datoteke... */
175     while (i < max - 1 && (c = fgetc(f)) != EOF) {
176         /* Proverava se da li je procitani karakter slovo */
177         if (isalpha(c))
178             /* Ako jeste, smesta se u niz - pritom se vrši konverzija
u mala slova jer program treba da bude neosetljiv na
180             razliku izmedju malih i velikih slova */
181             rec[i++] = tolower(c);
182
183         else
184             /* Ako nije, proverava se da li je procitano barem jedno
slovo nove reci */
185             /* Ako jeste, prekida se sa citanjem */
186             if (i > 0)
187                 break;
188
189         /* U suprotnom se ide na sledecu iteraciju */
190     }
191
192     /* Dodaje se na rec terminirajuca nula */
193     rec[i] = '\0';
194
195     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
196     return i > 0 ? 0 : EOF;
197 }
198
199 int main(int argc, char **argv)
200 {
201     Cvor *koren = NULL, *max;
202     FILE *f;
203     char rec[MAX];
204
205     /* Provera da li je navedeno ime datoteke prilikom pokretanja
programa */
206     if (argc < 2) {
207         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
208         exit(EXIT_FAILURE);
209     }
210
211     /* Priprema datoteke za citanje */
212     if ((f = fopen(argv[1], "r")) == NULL) {

```

```

216     fprintf(stderr, "fopen() greska pri otvaranju %s\n",
        argv[1]);
218     exit(EXIT_FAILURE);
}

220 /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
    pretrage. */
222 while (procitaj_rec(f, rec, MAX) != EOF)
    dodaj_u_stablo(&koren, rec);

224
226 /* Posto je citanjem reci zavrsono, zatvara se datoteka */
fclose(f);

228 /* Prikazuju se sve reci iz teksta i brojevi njihovih
    pojavljivanja. */
230 prikazi_stablo(koren);

232 /* Pronalazi se najfrekventnija rec */
max = najdi_najfrekventniju_rec(koren);

234
236 /* Ako takve reci nema... */
if (max == NULL)

238     /* Ispisuje se odgovarajuće obavestenje */
    printf("U tekstu nema reci!\n");

240
242 else
    /* Inace, ispisuje se broj pojavljivanja reci */
    printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
244         max->rec, max->brojac);

246 /* Oslobadja se dinamički alociran prostor za stablo */
oslobodi_stablo(&koren);

248
250 return 0;
}

```

#### Rešenje 4.16

```

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

6 #define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
8 #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
    broj telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {

```

```

14     char ime_i_prezime[MAX_IME_I_PREZIME];
15     char telefon[MAX_CIFARA];
16     struct cvor *levo;
17     struct cvor *desno;
18 } Cvor;

19
20 /* Funkcija koja kreira novi cvora stabla */
21 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
22 {
23     /* Alocira se memorija za novi cvor i proverava se uspesnost
24        alokacije. */
25     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
26     if (novi_cvor == NULL)
27         return NULL;

28     /* Inicijalizuju se polja novog cvora */
29     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
30     strcpy(novi_cvor->telefon, telefon);
31     novi_cvor->levo = NULL;
32     novi_cvor->desno = NULL;

33     /* Vraca se adresa novog cvora */
34     return novi_cvor;
35 }

36
37 /* Funkcija koja proverava uspesnost kreiranja novog cvora
38    stabla */
39 void proveri_alokaciju(Cvor * novi_cvor)
40 {
41     /* Ukoliko je cvor neuspesno kreiran */
42     if (novi_cvor == NULL) {
43         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
44            programa */
45         fprintf(stderr, "Malloc greska za novi cvor!\n");
46         exit(EXIT_FAILURE);
47     }
48 }

49
50 /* Funkcija koja dodaje novu osobu i njen broj telefona u
51    stablo. */
52 void
53 dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
54               char *telefon)
55 {
56     /* Ako je stablo prazno */
57     if (*adresa_korena == NULL) {
58         /* Kreira se novi cvor */
59         Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
60         proveri_alokaciju(novi);

61         /* I proglašava se korenom stabla */
62         *adresa_korena = novi;
63     }
64 }

```

```

66     return;
67 }
68
69 /* U suprotnom trazi se odgovarajuca pozicija za novi unos.
70    Kako pretragu treba vrsiti po imenu i prezimenu, stablo
71    treba da bude pretrazivacko po ovom polju */
72
73 /* Ako je zadato ime i prezime leksikografski manje od imena i
74    prezimena sadrzanog u korenu, podaci se dodaju u levo
75    podstablo */
76 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
77     < 0)
78     dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
79                  telefon);
80
81 else
82     /* Ako je zadato ime i prezime leksikografski vece od imena
83        i prezimena sadrzanog u korenu, podaci se dodaju u desno
84        podstablo */
85 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
86     dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
87                  telefon);
88 }
89
90 /* Funkcija koja oslobadja memoriju zauzetu stablom */
91 void oslobodi_stablo(Cvor ** adresa_korena)
92 {
93     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
94     if (*adresa_korena == NULL)
95         return;
96
97     /* Inace ... */
98     /* Oslobadja se memorija zauzeta levim podstablom */
99     oslobodi_stablo(&(*adresa_korena)->levo);
100
101     /* Oslobadja se memorija zauzeta desnim podstablom */
102     oslobodi_stablo(&(*adresa_korena)->desno);
103
104     /* Oslobadja se memorija zauzeta korenom */
105     free(*adresa_korena);
106
107     /* Stablo se proglašava praznim */
108     *adresa_korena = NULL;
109 }
110
111 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
112 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
113    koristiti za proveru da li je stablo lepo kreirano ili ne */
114 void prikazi_stablo(Cvor * koren)
115 {
116     /* Ako je stablo prazno, završava se sa ispisom */
117     if (koren == NULL)

```

```
        return;
118
    /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz
120     * levog podstabla */
    prikazi_stablo(koren->levo);
122
    /* Zatim se ispisuju podaci iz korena */
124    printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

    /* I nastavlja se sa ispisom podataka iz desnog podstabla */
126    prikazi_stablo(koren->desno);
128 }

/* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
130  * ime i prezime i broj telefona u odgovarajuće nizove.
132  * Maksimalna dužina imena i prezimena određena je konstantom
134  * MAX_IME_PREZIME, a maksimalna dužina broja telefona
136  * konstantom MAX_CIFARA. Funkcija vraća EOF ako nema više
138  * kontakata ili 0 u suprotnom. */
int procitaj_kontakt(FILE * f, char *ime_i_prezime,
140                     char *telefon)
{
    /* Karakter koji se cita */
142    int c;

    /* Indeks pozicije na koju se smesta procitani karakter */
144    int i = 0;

    /* Linije datoteke koje se obradjuju su formata Ime Prezime
146     * BrojTelefona */

    /* Preskacu se eventualne praznine sa pocetka linije datoteke */
148    while ((c = fgetc(f)) != EOF && isspace(c));
150

    /* Prvo procitano slovo upisuje se u ime i prezime */
152    if (!feof(f))
        ime_i_prezime[i++] = c;
154

    /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
156     * cifre tako da se citanje vrsi sve dok se ne naidje na
158     * cifru. Pritom treba voditi racuna da li ima dovoljno mesta
160     * za smestanje procitanog karaktera i da se slucajno ne dodje
162     * do kraja datoteke */
    while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
        if (!isdigit(c))
            ime_i_prezime[i++] = c;
164
        else if (i > 0)
            break;
166    }

    /* Upisuje se terminirajuca nula na mesto poslednjeg
168
```

```

170     procitanog blanko karaktera */
171     ime_i_prezime[--i] = '\0';

172     /* I pocinje se sa citanjem broja telefona */
173     i = 0;

174     /* Upisuje se cifra koja je vec procitana */
175     telefon[i++] = c;

176     /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
177        karaktera cije prisustvo nije dozvoljeno u broju telefona */
178     while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
179     {
180         if (c == '/' || c == '-' || isdigit(c))
181             telefon[i++] = c;
182         else
183             break;

184     /* Upisuje se terminirajuca nula */
185     telefon[i] = '\0';

186     /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
187     return !feof(f) ? 0 : EOF;
188 }

189 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
190    prezimenom */
191 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
192 {
193     /* Ako je imenik prazan, zavrшава se sa pretragom */
194     if (koren == NULL)
195         return NULL;

196     /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
197        zavrшава sa pretragom */
198     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
199         return koren;

200     /* Ako je zadato ime i prezime leksikografski manje od
201        vrednosti u korenu pretraga se nastavlja levo */
202     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
203         return pretrazi_imenik(koren->levo, ime_i_prezime);

204     else
205         /* u suprotnom, pretraga se nastavlja desno */
206         return pretrazi_imenik(koren->desno, ime_i_prezime);
207 }

208 int main(int argc, char **argv)
209 {
210     char ime_datoteke[MAX_IME_DATOTEKE];
211     Cvor *koren = NULL;
212     Cvor *trazeni;

```

```
FILE *f;
222 char ime_i_prezime[MAX_IME_I_PREZIME];
char telefon[MAX_CIFARA];
224 char c;
int i;
226
/* Ucitava se ime datoteke i vrsi se njena priprema za citanje
228 */
printf("Unesite ime datoteke: ");
230 scanf("%s", ime_datoteke);
if ((f = fopen(ime_datoteke, "r")) == NULL) {
232     fprintf(stderr, "Greska prilikom otvaranja datoteke
%s!\n", ime_datoteke);
234     exit(EXIT_FAILURE);
}
236
/* Podaci se citaju iz datoteke i smestaju u binarno stablo
238 pretrage. */
while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
240     dodaj_u_stablo(&koren, ime_i_prezime, telefon);

242 /* Zatvara se datoteka */
fclose(f);
244

/* Omogucava se pretraga imenika */
while (1) {
246     /* Ucitavaja se ime i prezime */
printf("Unesite ime i prezime: ");
248     i = 0;
while ((c = getchar()) != '\n')
250         ime_i_prezime[i++] = c;
ime_i_prezime[i] = '\0';
252

254     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja
se funkcionalnost */
if (strcmp(ime_i_prezime, "KRAJ") == 0)
256         break;
258

/* Inace se ispisuje rezultat pretrage */
trazeni = pretrazi_imenik(koren, ime_i_prezime);
260 if (trazeni == NULL)
printf("Broj nije u imeniku!\n");
262 else
printf("Broj je: %s \n", trazeni->telefon);
264 }
266

/* Oslobadja se memorija zauzeta imenikom */
oslobodi_stablo(&koren);
268

270 return 0;
}
```



## Rešenje 4.17

```

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

5  #define MAX 51

7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
   studenta, ukupan uspeh, uspeh iz matematike, uspeh iz
9  maternjeg jezika i redom pokazivace na levo i desno podstablo
   */
11 typedef struct cvor_stabla {
12     char ime[MAX];
13     char prezime[MAX];
14     double uspeh;
15     double matematika;
16     double jezik;
17     struct cvor_stabla *levo;
18     struct cvor_stabla *desno;
19 } Cvor;

21 /* Funkcija kojom se kreira cvor stabla */
Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
23                  double matematika, double jezik)
{
25     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
27     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
28     if (novi == NULL)
29         return NULL;

31     /* Inicijalizuju se polja strukture */
32     strcpy(novi->ime, ime);
33     strcpy(novi->prezime, prezime);
34     novi->uspeh = uspeh;
35     novi->matematika = matematika;
36     novi->jezik = jezik;
37     novi->levo = NULL;
38     novi->desno = NULL;

39     /* Vraca se adresa kreiranog cvora */
41     return novi;
42 }

43 /* Funkcija kojom se proverava uspesnost alociranja memorije */
44 void proveri_alokaciju(Cvor * novi_cvor)
45 {
46     /* Ukoliko je cvor neuspesno kreiran */
47     if (novi_cvor == NULL) {
48         /* Ispisuje se odgovarajuca poruka i prekida se izvršavanje
           programa */

```

```
51     fprintf(stderr, "Malloc greska za novi cvor!\n");
    exit(EXIT_FAILURE);
53 }
}

55 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
57 void oslobodi_stablo(Cvor ** koren)
{
59     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*koren == NULL)
61         return;

63     /* Inace ... */
    /* Oslobadja se memorija zauzeta levim podstablom */
65     oslobodi_stablo(&(*koren)->levo);

67     /* Oslobadja se memorija zauzeta desnim podstablom */
    oslobodi_stablo(&(*koren)->desno);

69     /* Oslobadja se memorija zauzeta korenom */
71     free(*koren);

73     /* Stablo se proglašava praznim */
    *koren = NULL;
75 }

77 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
79                 double uspeh, double matematika,
                    double jezik)
81 {
    /* Ako je stablo prazno */
83     if (*koren == NULL) {
        /* Kreira se novi cvor */
85         Cvor *novi =
            napravi_cvor(ime, prezime, uspeh, matematika, jezik);
87         prover_i_alokaciju(novi);

89         /* I proglašava se korenom stabla */
        *koren = novi;

91         return;
93     }

95     /* Inace, dodaje se cvor u stablo tako da bude sortirano po
        ukupnom broju poena */
97     if (uspeh + matematika + jezik >
        (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
99         dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
                        matematika, jezik);
101     else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
```

```

103         matematika, jezik);
104     }
105
106     /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost
107     argumenta polozili jednaka 0 ispisuju se informacije o
108     ucenicima koji nisu polozili prijemni, a ako je vrednost
109     argumenta razlicita od nule, ispisuju se informacije o
110     ucenicima koji su polozili prijemni */
111     void stampaj(Cvor * koren, int polozili)
112     {
113         /* Stablo je prazno - prekida se sa ispisom */
114         if (koren == NULL)
115             return;
116
117         /* Stampaju se informacije iz levog podstabla */
118         stampaj(koren->levo, polozili);
119
120         /* Stampaju se informacije iz korenog cvora */
121         if (polozili && koren->matematika + koren->jezik >= 10)
122             printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
123                 koren->prezime, koren->uspeh, koren->matematika,
124                 koren->jezik,
125                 koren->uspeh + koren->matematika + koren->jezik);
126         else if (!polozili && koren->matematika + koren->jezik < 10)
127             printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
128                 koren->prezime, koren->uspeh, koren->matematika,
129                 koren->jezik,
130                 koren->uspeh + koren->matematika + koren->jezik);
131
132         /* Stampaju se informacije iz desnog podstabla */
133         stampaj(koren->desno, polozili);
134     }
135
136     /* Funkcija koja odredjuje koliko studenata nije polozilo
137     prijemni ispit */
138     int nisu_polozili(Cvor * koren)
139     {
140         /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
141         if (koren == NULL)
142             return 0;
143
144         /* Pretraga se vrshi i u levom i u desnom podstablu - ako uslov
145         za polaganje nije ispunjen za koreni cvor, broj studenata se
146         uvecava za 1 */
147         if (koren->matematika + koren->jezik < 10)
148             return 1 + nisu_polozili(koren->levo) +
149                 nisu_polozili(koren->desno);
150
151         return nisu_polozili(koren->levo) +
152             nisu_polozili(koren->desno);
153     }

```

```

155 int main(int argc, char **argv)
156 {
157     FILE *in;
158     Cvor *koren;
159     char ime[MAX], prezime[MAX];
160     double uspeh, matematika, jezik;
161
162     /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
163     in = fopen("prijemni.txt", "r");
164     if (in == NULL) {
165         fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
166         exit(EXIT_FAILURE);
167     }
168
169     /* Citanje podataka i dodavanje u stablo */
170     koren = NULL;
171     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
172                  &matematika, &jezik) != EOF) {
173         dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
174                      jezik);
175     }
176
177     /* Zatvaranje datoteke */
178     fclose(in);
179
180     /* Stampaju se prvo podaci o ucenicima koji su položili
181        prijemni */
182     stampaj(koren, 1);
183
184     /* Linij se iscrtava samo ako postoje učenici koji nisu
185        položili prijemni */
186     if (nisu_položili(koren) != 0)
187         printf("-----\n");
188
189     /* Stampaju se podaci o ucenicima koji nisu položili prijemni */
190     stampaj(koren, 0);
191
192     /* Oslobadja se memorija zauzeta stablom */
193     oslobodi_stablo(&koren);
194
195     return 0;
196 }

```

### Rešenje 4.18

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 #define MAX_NISKA 51
6 #define MAX_DATUM 3

```

```
8  /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i
   prezime osobe, dan, mesec i godinu rođenja i redom
10  pokazivace na levo i desno podstablo */
typedef struct cvor_stabla {
12  char ime[MAX_NISKA];
   char prezime[MAX_NISKA];
14  int dan;
   int mesec;
16  int godina;
   struct cvor_stabla *levo;
18  struct cvor_stabla *desno;
} Cvor;

20
/* Funkcija koja kreira novi cvor */
22 Cvor *napravi_cvor(char ime[], char prezime[], int dan,
   int mesec, int godina)
24 {
   /* Alocira se memorija */
26 Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   if (novi == NULL)
28     return NULL;

   /* Inicijalizuju se polja strukture */
30 strcpy(novi->ime, ime);
32 strcpy(novi->prezime, prezime);
   novi->dan = dan;
34 novi->mesec = mesec;
   novi->godina = godina;
36 novi->levo = NULL;
   novi->desno = NULL;

38
   /* Vraca se adresa novog cvora */
40 return novi;
}

42
/* Funkcija koja proverava uspesnost alokacije */
44 void prover_i_alokaciju(Cvor * novi_cvor)
{
46   /* Ako memorija nije uspesno alocirana */
   if (novi_cvor == NULL) {
48     /* Ispisuje se poruka i prekida se sa izvršavanjem programa */
     fprintf(stderr, "Malloc greska za novi cvor!\n");
50     exit(EXIT_FAILURE);
   }
52 }

54 /* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** koren)
56 {
   /* Stablo je prazno */
58   if (*koren == NULL)
```

```

        return;
60
    /* Oslobadja se memorija zauzeta levim podstablom (ako
62     postoji) */
    if ((*koren)->levo)
64         oslobodi_stablo(&(*koren)->levo);

66     /* Oslobadja se memorija zauzeta desnim podstablom (ako
        postoji) */
68     if ((*koren)->desno)
        oslobodi_stablo(&(*koren)->desno);

70     /* Oslobadja se memorija zauzeta korenom */
72     free(*koren);

74     /* Proglasava se stablo praznim */
    *koren = NULL;
76 }

78 /* Funkcija koja dodaje novi cvor u stablo - stablo treba da
    bude uredjeno po datumu */
80 void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                        int dan, int mesec, int godina)
82 {
    /* Ako je stablo prazno */
84     if (*koren == NULL) {

86         /* Kreira se novi cvor */
        Cvor *novi_cvor =
88             napravi_cvor(ime, prezime, dan, mesec, godina);
        prover_i_alokaciju(novi_cvor);

90         /* I proglasava se korenom */
92         *koren = novi_cvor;

94         return;
    }

96
    /* Kako se ne unosi godina za pretragu, stablo se uredjuje
98     samo po mesecu (i danu u okviru istog meseca) */
    if (mesec < (*koren)->mesec)
100         dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
                        godina);
    else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
102         dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
                        godina);
104     else
106         dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec,
                        godina);
108 }

110 /* Funkcija vrši pretragu stabla i vraća cvor sa traženim

```

```

112     datumom (null ako takav ne postoji). U promenljivu pom ce
    biti smesten prvi datum (dan i mesec) veci od trazenog datuma
    (null ako takav ne postoji) */
114 Cvor *pretrazi(Cvor * koren, int dan, int mesec)
    {
116     /* Stablo je prazno, obustavlja se pretraga */
    if (koren == NULL)
118         return NULL;

120     /* Ako je trazeni datum u korenu */
    if (koren->dan == dan && koren->mesec == mesec)
122         return koren;

124     /* Ako je mesec trazenog datuma manji od meseca sadrzanog u
    korenu ili ako su meseci isti ali je dan trazenog datuma
126     manji od aktuelnog datuma, pretrazuje se levo podstablo -
    pre toga se svakako proverava da li leva grana postoji -
128     ako ne postoji treba vratiti prvi sledeci, a to je bas
    vrednost uocenog korena */
130     if (mesec < koren->mesec
        || (mesec == koren->mesec && dan < koren->dan)) {
132         if (koren->levo == NULL)
            return koren;
134         else
            return pretrazi(koren->levo, dan, mesec);
136     }

138     /* Inace se nastavlja pretraga u desnom delu */
    return pretrazi(koren->desno, dan, mesec);
140 }

142 int main(int argc, char **argv)
    {
144     FILE *in;
    Cvor *koren;
146     Cvor *slavljenik;
    char ime[MAX_NISKA], prezime[MAX_NISKA];
148     int dan, mesec, godina;

150     /* Provera da li je zadato ime ulazne datoteke */
    if (argc < 2) {
152         /* Ako nije, ispisuje se poruka i prekida se sa izvrsavanjem
        programa */
154         printf("Nedostaje ime ulazne datoteke!\n");
        return 0;
156     }

158     /* Inace, priprema se datoteka za citanje */
    in = fopen(argv[1], "r");
160     if (in == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
162         exit(EXIT_FAILURE);
    }

```

```
164 }
165
166 /* I stablo se popunjava podacima */
167 koren = NULL;
168 while (fscanf
169     (in, "%s %s %d.%d.%d.", ime, prezime, &dan, &mesec,
170      &godina) != EOF)
171     dodaj_u_stablo(&koren, ime, prezime, dan, mesec, godina);
172
173 /* Zatvaranje datoteke */
174 fclose(in);
175
176 /* Omogućuje se pretraga podataka */
177 while (1) {
178     /* Učitava se novi datum */
179     printf("Unesite datum: ");
180     if (scanf("%d.%d.", &dan, &mesec) == EOF)
181         break;
182
183     /* Pretražuje se stablo */
184     slavljenik = pretrazi(koren, dan, mesec);
185
186     /* Ispisuju se pronadjeni podaci */
187     if (slavljenik == NULL) {
188         printf
189             ("Nema podataka o oviom ni o sledecem rođjendanu.\n");
190         continue;
191     }
192
193     /* Slučaj kada su pronadjeni pravi podaci */
194     if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
195         printf("Slavljenik: %s %s\n", slavljenik->ime,
196             slavljenik->prezime);
197         continue;
198     }
199
200     /* Slučaj su pronadjeni podaci o prvom sledecem rođjendanu */
201     printf("Slavljenik: %s %s %d.%d.\n", slavljenik->ime,
202         slavljenik->prezime, slavljenik->dan,
203         slavljenik->mesec);
204 }
205
206 /* Oslobadja se memorija zauzeta stablom */
207 oslobodi_stablo(&koren);
208
209 return 0;
210 }
```

### Rešenje 4.19



```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1

4  /* Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
6  typedef struct cvor {
   int broj;
   struct cvor *levo, *desno;
8  } Cvor;

10
12 /* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
   Cvor *napravi_cvor(int broj);

14
16 /* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
   void prover_i_alokaciju(Cvor * novi_cvor);

18
20 /* Funkcija koja dodaje zadati broj u stablo */
   void dodaj_u_stablo(Cvor ** adresa_korena, int broj);

22
24 /* Funkcija koja proverava da li se zadati broj nalazi u stablu */
   Cvor *pretrazi_stablo(Cvor * koren, int broj);

26
28 /* Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
   stablu */
   Cvor *pronadji_najmanji(Cvor * koren);

30
32 /* Funkcija koja pronalazi cvor koji sadrzi najveću vrednost u
   stablu */
   Cvor *pronadji_najveci(Cvor * koren);

34
36 /* Funkcija koja brise cvor stabla koji sadrzi zadati broj. */
   void obrisi_element(Cvor ** adresa_korena, int broj);

38
40 /* Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
   podstablo - Koren - Desno podstablo) */
   void prikazi_stablo(Cvor * koren);

42
44 /* Funkcija koja oslobadja memoriju zauzetu stablom */
   void oslobodi_stablo(Cvor ** adresa_korena);

46
48 #endif

```

```

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

5  Cvor *napravi_cvor(int broj)
   {
7  /* Alocira se memorija za novi cvor i proverava se uspesnost

```

```
    alokacije. */
9   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
11      return NULL;
    /* Inicijalizuju se polja novog cvora. */
13   novi->broj = broj;
    novi->levo = NULL;
15   novi->desno = NULL;
    /* Vraca se adresa novog cvora. */
17   return novi;
}

19
void prover_i_alokaciju(Cvor * novi_cvor)
21 {
    /* Ukoliko je cvor neuspesno kreiran */
23   if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
25      programa */
        fprintf(stderr, "Malloc greska za novi cvor!\n");
27      exit(EXIT_FAILURE);
    }
29 }

31 void dodaj_u_stablo(Cvor ** koren, int broj)
{
33   /* Ako je stablo prazno */
    if (*koren == NULL) {
35      /* Kreira se novi cvor */
        Cvor *novi = napravi_cvor(broj);
37      prover_i_alokaciju(novi);
        /* I proglašava se korenom stabla */
39      *koren = novi;
        return;
41   }
    /* U suprotnom se trazi odgovarajuca pozicija za zadati broj: */
43
    /* Ako je zadata vrednost manja od vrednosti korena */
45   if (broj < (*koren)->broj)
        /* Broj se dodaje u levo podstablo */
47      dodaj_u_stablo(&(*koren)->levo, broj);
    else
49      /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa
        se dodaje u desno podstablo */
51      dodaj_u_stablo(&(*koren)->desno, broj);
}

53
Cvor *pretrazi_stablo(Cvor * koren, int broj)
55 {
    /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
57   */
    if (koren == NULL)
59      return NULL;
```

```
61  /* Ako je trazena vrednost sadrzana u korenu */
62  if (koren->broj == broj) {
63      /* Prekida se pretraga */
64      return koren;
65  }
66  /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
67  if (broj < koren->broj)
68      /* Pretraga se nastavlja u levom podstablu */
69      return pretrazi_stablo(koren->levo, broj);
70  else
71      /* U suprotnom, pretraga se nastavlja u desnom podstablu */
72      return pretrazi_stablo(koren->desno, broj);
73  }
74
75  Cvor *pronadji_najmanji(Cvor * koren)
76  {
77      /* Ako je stablo prazno, prekida se pretraga */
78      if (koren == NULL)
79          return NULL;
80
81      /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze
82       se levo od njega */
83
84      /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
85       najmanju vrednost */
86      if (koren->levo == NULL)
87          return koren;
88
89      /* Inace, pretragu treba nastaviti u levom podstablu */
90      return pronadji_najmanji(koren->levo);
91  }
92
93  Cvor *pronadji_najveci(Cvor * koren)
94  {
95      /* Ako je stablo prazno, prekida se pretraga */
96      if (koren == NULL)
97          return NULL;
98
99      /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze
100       se desno od njega */
101
102      /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
103       najvecu vrednost */
104      if (koren->desno == NULL)
105          return koren;
106
107      /* Inace, pretragu treba nastaviti u desnom podstablu */
108      return pronadji_najveci(koren->desno);
109  }
110
111  void obrisi_element(Cvor ** adresa_korena, int broj)
```

```

113 Cvor *pomocni_cvor = NULL;

115 /* Ako je stablo prazno, brisanje nije primenljivo */
116 if (*adresa_korena == NULL)
117     return;

119 /* Ako je vrednost koju treba obrisati manja od vrednosti u
120    korenu stabla, ona se eventualno nalazi u levom podstablu,
121    pa treba rekurzivno primeniti postupak na levo podstablo.
122    Koren ovako modifikovanog stabla je nepromenjen. */
123 if (broj < (*adresa_korena)->broj) {
124     obrisi_element(&(*adresa_korena)->levo, broj);
125     return;
126 }

127 /* Ako je vrednost koju treba obrisati veca od vrednosti u
128    korenu stabla, ona se eventualno nalazi u desnom podstablu
129    pa treba rekurzivno primeniti postupak na desno podstablo.
130    Koren ovako modifikovanog stabla je nepromenjen. */
131 if ((*adresa_korena)->broj < broj) {
132     obrisi_element(&(*adresa_korena)->desno, broj);
133     return;
134 }

135 /* Slede podslucajevi vezani za slucaj kada je vrednost u
136    korenu jednaka broju koji se brise (tj. slucaj kada treba
137    obrisati koren) */

139 /* Ako koren nema sinova, tada se on prosto brise, i rezultat
140    je prazno stablo (vraca se NULL) */
141 if ((*adresa_korena)->levo == NULL
142     && (*adresa_korena)->desno == NULL) {
143     free(*adresa_korena);
144     *adresa_korena = NULL;
145     return;
146 }

147 /* Ako koren ima samo levog sina, tada se brisanje vrši tako
148    sto se brise koren, a novi koren postaje levi sin */
149 if ((*adresa_korena)->levo != NULL
150     && (*adresa_korena)->desno == NULL) {
151     pomocni_cvor = (*adresa_korena)->levo;
152     free(*adresa_korena);
153     *adresa_korena = pomocni_cvor;
154     return;
155 }

156 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako
157    sto se brise koren, a novi koren postaje desni sin */
158 if ((*adresa_korena)->desno != NULL
159     && (*adresa_korena)->levo == NULL) {
160     pomocni_cvor = (*adresa_korena)->desno;
161 }

```

```

165     free(*adresa_korena);
166     *adresa_korena = pomocni_cvor;
167     return;
168 }

169 /* Slučaj kada koren ima oba sina - najpre se potraži
170    sledbenik korena (u smislu poretka) u stablu. To je upravo
171    po vrednosti najmanji cvor u desnom podstablu. On se može
172    pronaći npr. funkcijom pronadji_najmanji(). Nakon toga se u
173    koren smesti vrednost tog cvora, a u taj cvor se smesti
174    vrednost korena (tj. broj koji se briše). Zatim se prosto
175    rekursivno pozove funkcija za brisanje na desno podstablo.
176    S obzirom da u njemu treba obrisati najmanji element, a on
177    zasigurno ima najviše jednog potomka, jasno je da će
178    njegovo brisanje biti obavljeno na jedan od jednostavnijih
179    načina koji su gore opisani. */
180 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
181 (*adresa_korena)->broj = pomocni_cvor->broj;
182 pomocni_cvor->broj = broj;
183 obriši_element(&(*adresa_korena)->desno, broj);
184 }

185 void prikazi_stablo(Cvor * koren)
186 {
187     /* Ako je stablo prazno, prekida se ispis */
188     if (koren == NULL)
189         return;
190
191     /* Prvo se ispisuju svi cvorovi levo od korena */
192     prikazi_stablo(koren->levo);
193
194     /* Zatim se ispisuje vrednost u korenu */
195     printf("%d ", koren->broj);
196
197     /* Na kraju se ispisuju svi cvorovi desno od korena */
198     prikazi_stablo(koren->desno);
199 }

200 void oslobodi_stablo(Cvor ** koren)
201 {
202     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
203     if (*koren == NULL)
204         return;
205     /* Inace ... */
206     /* Oslobadja se memorija zauzeta levim podstablom */
207     if ((*koren)->levo)
208         oslobodi_stablo(&(*koren)->levo);
209     /* Oslobadja se memorija zauzeta desnim podstablom */
210     if ((*koren)->desno)
211         oslobodi_stablo(&(*koren)->desno);
212     /* Oslobadja se memorija zauzeta korenom */
213     free(*koren);
214 }

```

```
217  /* Proglasava se stablo praznim */
    *koren = NULL;
}

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

4 /* Uključuje se biblioteka za rad sa stablima - pogledati uvodni
   zadatak ove glave */
6 #include "stabla.h"

8
10 /* Funkcija koja proverava da li su dva stabla koja sadrže cele
   brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
12 int identitet(Cvor * koren1, Cvor * koren2)
{
14     /* Ako su oba stabla prazna, jednaka su */
    if (koren1 == NULL && koren2 == NULL)
16         return 1;

18     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
       jednaka */
20     if (koren1 == NULL || koren2 == NULL)
        return 0;

22     /* Ako su oba stabla neprazna i u korenu se nalaze različite
       vrednosti, može se zaključiti da se razlikuju */
24     if (koren1->broj != koren2->broj)
        return 0;

26     /* Inače, proverava se da li vazi i jednakost levih i desnih
       podstabala */
28     return (identitet(koren1->levo, koren2->levo)
        && identitet(koren1->desno, koren2->desno));
32 }

34 int main()
{
36     int broj;
    Cvor *koren1, *koren2;

38
40     /* Učitavaju se elementi prvog stabla */
    koren1 = NULL;
    printf("Prvo stablo: ");
42     scanf("%d", &broj);
    while (broj != 0) {
44         dodaj_u_stablo(&koren1, broj);
        scanf("%d", &broj);
46     }

48     /* Učitavaju se elementi drugog stabla */
```

```

    koren2 = NULL;
50 printf("Drugo stablo: ");
    scanf("%d", &broj);
52 while (broj != 0) {
        dodaj_u_stablo(&koren2, broj);
54     scanf("%d", &broj);
    }
56
    /* Poziva se funkcija koja ispituje identitet stabala i
58     ispisuje se njen rezultat. */
    if (identitet(koren1, koren2))
60     printf("Stabla jesu identicna.\n");
    else
62     printf("Stabla nisu identicna.\n");

64     /* Oslobadja se memorija zauzeta stablima */
    oslobodi_stablo(&koren1);
66     oslobodi_stablo(&koren2);

68     return 0;
}

```

#### Rešenje 4.20

```

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

4  /* Uklucuje se biblioteka za rad sa stablima */
#include "stabla.h"

6
/* Funkcija kreira novo stablo identicno stablu koje je dato
8  korenom. */
void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
10 {
    /* Izlaz iz rekurzije */
12     if (koren == NULL) {
        *duplikat = NULL;
14         return;
    }
16
    /* Duplira se koren stabla i postavlja da bude koren novog
18     stabla */
    *duplikat = napravi_cvor(koren->broj);
20     prover_i_alokaciju(*duplikat);

22     /* Rekurzivno se duplira levo podstablo i njegova adresa se
        cuva u pokazivacu na levo podstablo korena duplikata. */
24     kopiraj_stablo(koren->levo, &(*duplikat)->levo);

26     /* Rekurzivno se duplira desno podstablo i njegova adresa se
        cuva u pokazivacu na desno podstablo korena duplikata. */

```

```

28     kopiraj_stablo(koren->desno, &(*duplikat)->desno);
29 }
30
31 /* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
32    dobija modifikacijom prvog stabla */
33 void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
34 {
35     /* Ako drugo stablo nije prazno */
36     if (koren2 != NULL) {
37         /* Dodaje se njegov koren u prvo stablo */
38         dodaj_u_stablo(adresa_korena1, koren2->broj);
39
40         /* Rekurzivno se racuna unija levog i desnog podstabla
41            drugog stabla sa prvim stablom */
42         kreiraj_uniju(adresa_korena1, koren2->levo);
43         kreiraj_uniju(adresa_korena1, koren2->desno);
44     }
45 }
46
47 /* Funkcija izracunava presek dva stabla - rezultujuce stablo se
48    dobija modifikacijom prvog stabla */
49 void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
50 {
51     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
52        */
53     if (*adresa_korena1 == NULL)
54         return;
55
56     /* Inace... */
57     /* Kreira se presek levog i desnog podstabla sa drugim
58        stablom, tj. iz levog i desnog podstabla prvog stabla
59        brisu se svi oni elementi koji ne postoje u drugom stablu */
60     kreiraj_presek(&(*adresa_korena1)->levo, koren2);
61     kreiraj_presek(&(*adresa_korena1)->desno, koren2);
62
63     /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se
64        on uklanja iz prvog stabla */
65     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
66         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
67 }
68
69 /* Funkcija izracunava razliku dva stabla - rezultujuce stablo
70    se dobija modifikacijom prvog stabla */
71 void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
72 {
73     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
74        */
75     if (*adresa_korena1 == NULL)
76         return;
77
78     /* Inace... */
79     /* Kreira se razlika levog i desnog podstabla sa drugim

```



```

80     stablom, tj. iz levog i desnog podstabla prvog stabla se
      brisu svi oni elementi koji postoje i u drugom stablu */
82     kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
      kreiraj_razliku(&(*adresa_korena1)->desno, koren2);

84
      /* Ako se koren prvog stabla nalazi i u drugom stablu tada se
86       isti uklanja iz prvog stabla */
      if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
88         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
      }

90
91     int main()
92     {
      Cvor *koren1;
94     Cvor *koren2;
      Cvor *pomocni = NULL;
96     int n;

98     /* Ucitavaju se elementi prvog stabla */
      koren1 = NULL;
      printf("Prvo stablo: ");
100     while (scanf("%d", &n) != EOF) {
102         dodaj_u_stablo(&koren1, n);
      }

104
      /* Ucitavaju se elementi drugog stabla */
106     koren2 = NULL;
      printf("Drugo stablo: ");
108     while (scanf("%d", &n) != EOF) {
110         dodaj_u_stablo(&koren2, n);
      }

112     /* Kreira se unija stabala: prvo se napravi kopija prvog
      stabla kako bi se isto moglo iskoristiti i za preostale
114     operacije */
      kopiraj_stablo(koren1, &pomocni);
116     kreiraj_uniju(&pomocni, koren2);
      printf("Unija: ");
118     prikazi_stablo(pomocni);
      putchar('\n');

120
      /* Oslobadja se stablo sa rezultatom operacije */
122     oslobodi_stablo(&pomocni);

124
      /* Kreira se presek stabala: prvo se napravi kopija prvog
      stabla kako bi se isto moglo iskoristiti i za preostale
126     operacije */
      kopiraj_stablo(koren1, &pomocni);
128     kreiraj_presek(&pomocni, koren2);
      printf("Presek: ");
130     prikazi_stablo(pomocni);
      putchar('\n');

```

```
132  /* Oslobadja se stablo sa rezultatom operacije */
134  oslobodi_stablo(&pomocni);

136  /* Kreira se razlika stabala: prvo se napravi kopija prvog
138     stabla kako bi se isto moglo iskoristiti i za preostale
        operacije; */
    kopiraj_stablo(koren1, &pomocni);
140    kreiraj_razliku(&pomocni, koren2);
    printf("Razlika: ");
142    prikazi_stablo(pomocni);
    putchar('\n');
144
146  /* Oslobadja se stablo sa rezultatom operacije */
    oslobodi_stablo(&pomocni);

148  /* Oslobadjaju se i polazna stabla */
    oslobodi_stablo(&koren2);
150    oslobodi_stablo(&koren1);

152  return 0;
}
```

### Rešenje 4.21

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

3
   /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

7  #define MAX 50

9  /* Funkcija koja obilazi stablo sa leva na desno i smesta
   vrednosti cvorova u niz. Povratna vrednost funkcije je broj
11  vrednosti koje su smestene u niz. */
   int kreiraj_niz(Cvor * koren, int a[])
13  {
       int r, s;

15
       /* Stablo je prazno - u niz je smesteno 0 elemenata */
17       if (koren == NULL)
           return 0;

19
       /* Dodaju se u niz elementi iz levog podstabla */
21       r = kreiraj_niz(koren->levo, a);

23
       /* Tekuca vrednost promenljive r je broj elemenata koji su
       upisani u niz i na osnovu nje se moze odrediti indeks novog
25       elementa */
   }
```

```

27  /* Smesta se vrednost iz korena */
    a[r] = koren->broj;
29
    /* Dodaju se elementi iz desnog podstabla */
31  s = kreiraj_niz(koren->desno, a + r + 1);
33
    /* Racuna se indeks na koji treba smestiti naredni element */
    return r + s + 1;
35 }

37 /* Funkcija sortira niz tako sto najpre elemente niza smesti u
    stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva
39    u desno.

    Ovaj nacin sortiranja je primer sortiranja koje nije "u
41    mestu" kao sto je to slucaj sa ostalim opisanim algoritmima
43    sortiranja jer se sortiranje vrshi u pomocnoj dinamicnoj
    strukturi, a ne razmenom elemenata niza. */
45 void sortiraj(int a[], int n)
    {
47     int i;
        Cvor *koren;
49
        /* Kreira se stablo smestanjem elemenata iz niza u stablo */
51     koren = NULL;
        for (i = 0; i < n; i++)
53         dodaj_u_stablo(&koren, a[i]);

55     /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju
        u niz a */
57     kreiraj_niz(koren, a);

59     /* Stablo vise nije potrebno pa se oslobadja memorija koju
        zauzima */
61     oslobodi_stablo(&koren);
    }

63
65 int main()
    {
        int a[MAX];
        int n, i;

69     /* Ucitavaju se dimenzija i elementi niza */
        printf("n: ");
        scanf("%d", &n);
71     if (n < 0 || n > MAX) {
73         printf("Greska: pogresna dimenzija niza!\n");
        return 0;
75     }

77     printf("a: ");
        for (i = 0; i < n; i++)

```

```
79     scanf("%d", &a[i]);

81     /* Poziva se funkcija za sortiranje */
    sortiraj(a, n);

83     /* Ispisuje se rezultat */
85     for (i = 0; i < n; i++)
        printf("%d ", a[i]);
87     printf("\n");

89     return 0;
}
```

### Rešenje 4.22

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

3

   /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

7  /* a) Funkcija koja izračunava broj cvorova stabla */
   int broj_cvorova(Cvor * koren)
9  {
   /* Ako je stablo prazno, broj cvorova je nula */
11     if (koren == NULL)
        return 0;

13

   /* U suprotnom je broj cvorova stabla jednak zbiru broja
15     cvorova u levom podstablu i broja cvorova u desnom
        podstablu - 1 se dodaje zato što treba računati i koren */
17     return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) +
        1;

19 }

21 /* b) Funkcija koja izračunava broj listova stabla */
   int broj_listova(Cvor * koren)
23 {
   /* Ako je stablo prazno, broj listova je nula */
25     if (koren == NULL)
        return 0;

27

   /* Proverava se da li je tekuci cvor list */
29     if (koren->levo == NULL && koren->desno == NULL)
        /* Ako jeste vraća se 1 - to će kasnije zbog rekurzivnih
31         poziva uvećati broj listova za 1 */
        return 1;

33

   /* U suprotnom se prebrojavaju listovi koje se nalaze u
35     podstablima */
   return broj_listova(koren->levo) + broj_listova(koren->desno);
}
```

```
37 }
38
39 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
40 void pozitivni_listovi(Cvor * koren)
41 {
42     /* Slucaj kada je stablo prazno */
43     if (koren == NULL)
44         return;
45
46     /* Ako je cvor list i sadrzi pozitivnu vrednost */
47     if (koren->levo == NULL && koren->desno == NULL
48         && koren->broj > 0)
49         /* Stampa se */
50         printf("%d ", koren->broj);
51
52     /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
53     pozitivni_listovi(koren->levo);
54     pozitivni_listovi(koren->desno);
55 }
56
57 /* d) Funkcija koja izracunava zbir cvorova stabla */
58 int zbir_svih_cvorova(Cvor * koren)
59 {
60     /* Ako je stablo prazno, zbir cvorova je 0 */
61     if (koren == NULL)
62         return 0;
63
64     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i
65       svih elemenata u podstablima */
66     return koren->broj + zbir_svih_cvorova(koren->levo) +
67            zbir_svih_cvorova(koren->desno);
68 }
69
70 /* e) Funkcija koja izracunava najveći element stabla */
71 Cvor *najveci_element(Cvor * koren)
72 {
73     /* Ako je stablo prazno, obustavlja se pretraga */
74     if (koren == NULL)
75         return NULL;
76
77     /* Zbog prirode pretrazivackog stabla, vrednosti veće od
78       korena se nalaze u desnom podstablu */
79
80     /* Ako desnog podstabla nema */
81     if (koren->desno == NULL)
82         /* Najveća vrednost je koren */
83         return koren;
84
85     /* Inace, najveća vrednost se traži desno */
86     return najveci_element(koren->desno);
87 }
```

```
89  /* f) Funkcija koja izracunava dubinu stabla */
91  int dubina_stabla(Cvor * koren)
92  {
93      /* Dubina praznog stabla je 0 */
94      if (koren == NULL)
95          return 0;
96
97      /* Izracunava se dubina levog podstabla */
98      int dubina_levo = dubina_stabla(koren->levo);
99
100     /* Izracunava se dubina desnog podstabla */
101     int dubina_desno = dubina_stabla(koren->desno);
102
103     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se
104        dodaje jer se racuna i koren */
105     return dubina_levo >
106            dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
107 }
108
109 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou
110     stabla */
111 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
112 {
113     /* Ideja je spustanje kroz stablo sve dok se ne stigne do
114        trazenog nivoa */
115
116     /* Ako nema vise cvorova, nema spustanja niz stablo */
117     if (koren == NULL)
118         return 0;
119
120     /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije
121        zbog rekurzivnih poziva uvecati broj cvorova za 1 */
122     if (i == 0)
123         return 1;
124
125     /* Inace, spusta se jedan nivo nize i u levom i u desnom
126        postablu */
127     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
128            + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
129 }
130
131 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
132 void ispis_nivo(Cvor * koren, int i)
133 {
134     /* Ideja je slicna ideji iz prethodne funkcije */
135
136     /* Nema vise cvorova, nema spustanja kroz stablo */
137     if (koren == NULL)
138         return;
139
140     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
```

```

141     if (i == 0) {
142         printf("%d ", koren->broj);
143         return;
144     }
145     /* Inace, spustanje se nastavlja za jedan nivo nize i u levom
146        i u desnom podstablu */
147     ispis_nivo(koren->levo, i - 1);
148     ispis_nivo(koren->desno, i - 1);
149 }

151 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom
152     nivou stabla */
153 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
154 {
155     /* Ako je stablo prazno, obustavlja se pretraga */
156     if (koren == NULL)
157         return NULL;

159     /* Ako se stiglo do trazenog nivoa, takodje se prekida
160        pretraga */
161     if (i == 0)
162         return koren;

163     /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
164     Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);

166     /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
167     Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);

169     /* Trazi se i vraca maksimum izracunatih vrednosti */
170     if (a == NULL && b == NULL)
171         return NULL;
172     if (a == NULL)
173         return b;
174     if (b == NULL)
175         return a;
176     return a->broj > b->broj ? a : b;
177 }

179 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
180 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
181 {
182     /* Ako je stablo prazno, zbir je nula */
183     if (koren == NULL)
184         return 0;

186     /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
187     if (i == 0)
188         return koren->broj;

190     /* Inace, spustanje se nastavlja za jedan nivo nize i traže se
191        sume iz levog i desnog podstabla */

```

```
193     return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
194         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
195 }

197 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje
198     su manje ili jednake od date vrednosti x */
199 int zbir_manjih_od_x(Cvor * koren, int x)
200 {
201     /* Ako je stablo prazno, zbir je nula */
202     if (koren == NULL)
203         return 0;
204
205     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
206         prirode pretrazivackog stabla treba obici i levo i desno
207         podstablo */
208     if (koren->broj <= x)
209         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
210             zbir_manjih_od_x(koren->desno, x);
211
212     /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
213         medju njima jedino moze biti onih koje zadovoljavaju uslov */
214     return zbir_manjih_od_x(koren->levo, x);
215 }

217 int main(int argc, char **argv)
218 {
219     /* Analiza argumenata komandne linije */
220     if (argc != 3) {
221         fprintf(stderr,
222             "Greska! Program se poziva sa: ./a.out nivo
223             broj_za_pretragu\n");
224         exit(EXIT_FAILURE);
225     }
226     int i = atoi(argv[1]);
227     int x = atoi(argv[2]);
228
229     /* Kreira se stablo */
230     Cvor *koren = NULL;
231     int broj;
232     while (scanf("%d", &broj) != EOF)
233         dodaj_u_stablo(&koren, broj);
234
235     /* ispisuju se rezultati rada funkcija */
236     printf("broj cvorova: %d\n", broj_cvorova(koren));
237     printf("broj listova: %d\n", broj_listova(koren));
238     printf("pozitivni listovi: ");
239     pozitivni_listovi(koren);
240     printf("\n");
241     printf("zbir cvorova: %d\n", zbir_svih_cvorova(koren));
242     if (najveci_element(koren) == NULL)
243         printf("najveci element: ne postoji\n");
```



```

245     else
        printf("najveci element: %d\n",
               najveci_element(koren)->broj);
247
        printf("dubina stabla: %d\n", dubina_stabla(koren));
249
        printf("broj cvorova na %d. nivou: %d\n", i,
               broj_cvorova_na_itom_nivou(koren, i));
251        printf("elementi na %d. nivou: ", i);
253        ispis_nivo(koren, i);
        printf("\n");
255        if (najveci_element_na_itom_nivou(koren, i) == NULL)
            printf("Nema elemenata na %d. nivou!\n", i);
257        else
            printf("maksimalni na %d. nivou: %d\n", i,
                   najveci_element_na_itom_nivou(koren, i)->broj);
259
        printf("zbir na %d. nivou: %d\n", i,
               zbir_cvorova_na_itom_nivou(koren, i));
261        printf("zbir elemenata manjih ili jednakih od %d: %d\n", x,
               zbir_manjih_od_x(koren, x));
263
265        /* Oslobadja se memorija zauzeta stablom */
267        oslobodi_stablo(&koren);
269
        return 0;
    }

```

### Rešenje 4.23

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

6
/* Funkcija koja izracunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
{
10     /* Dubina praznog stabla je 0 */
    if (koren == NULL)
12         return 0;

14     /* Izracunava se dubina levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

16     /* Izracunava se dubina desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);

20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se
        dodaje jer se racuna i koren */

```

```
22     return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }

26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
28 {
    /* Ideja je slicna ideji iz prethodne funkcije */
    /* Nema vise cvorova, nema spustanja niz stablo */
    if (koren == NULL)
32         return;

34     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
    if (i == 0) {
36         printf("%d ", koren->broj);
        return;
38     }
    /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u
40     desnom podstablu */
    ispisi_nivo(koren->levo, i - 1);
    ispisi_nivo(koren->desno, i - 1);
42 }

44 /* Funkcija koja ispisuje stablo po nivoima */
void ispisi_stablo_po_nivoima(Cvor * koren)
46 {
48     int i;

50     /* Prvo se izracunava dubina stabla */
    int dubina;
    dubina = dubina_stabla(koren);
52

54     /* Ispisuje se nivo po nivo stabla */
    for (i = 0; i < dubina; i++) {
56         printf("%d. nivo: ", i);
        ispisi_nivo(koren, i);
58         printf("\n");
    }
60 }

62 int main(int argc, char **argv)
{
64     Cvor *koren;
    int broj;

66     /* Citaju se vrednosti sa ulaza i dodaju se u stablo */
    koren = NULL;
    while (scanf("%d", &broj) != EOF) {
68         dodaj_u_stablo(&koren, broj);
70     }

72     /* Ispisuje se stablo po nivoima */
```

```

74     ispisi_stablo_po_nivoima(koren);
76     /* Oslobadja se memorija zauzeta stablom */
       oslobodi_stablo(&koren);
78
       return 0;
80 }

```

### Rešenje 4.24

### Rešenje 4.25

```

1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
7
   /* Funkcija koja izracunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
9  {
       /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
           return 0;
13
       /* Izracunava se dubina levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
17
       /* Izracunava se dubina desnog podstabla */
       int dubina_desno = dubina_stabla(koren->desno);
19
       /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se
21         dodaje jer se racuna i koren */
       return dubina_levo >
23         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
   }
25
   /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za
27     AVL stablo */
   int avl(Cvor * koren)
29  {
       int dubina_levo, dubina_desno;
31
       /* Ako je stablo prazno, zaustavlja se brojanje */
33     if (koren == NULL) {
           return 0;
35     }
37
       /* Izracunava se dubina levog podstabla korena */
       dubina_levo = dubina_stabla(koren->levo);

```

```

39  /* Izracunava se dubina desnog podstabla korena */
41  dubina_desno = dubina_stabla(koren->desno);

43  /* Ako je uslov za AVL stablo ispunjen */
44  if (abs(dubina_desno - dubina_levo) <= 1) {
45      /* Racuna se broj AVL cvorova u levom i desnom podstablu i
46         uvecava za jedan iz razloga sto koren ispunjava uslov */
47      return 1 + avl(koren->levo) + avl(koren->desno);
48  } else {
49      /* Inace, racuna se samo broj AVL cvorova u podstablama */
50      return avl(koren->levo) + avl(koren->desno);
51  }
52  }

53  int main(int argc, char **argv)
54  {
55      Cvor *koren;
56      int broj;

59      /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo */
60      koren = NULL;
61      while (scanf("%d", &broj) != EOF) {
62          dodaj_u_stablo(&koren, broj);
63      }

65      /* Racuna se i ispisuje broj AVL cvorova */
66      printf("%d\n", avl(koren));

67      /* Oslobadja se memorija zauzeta stablom */
68      oslobodi_stablo(&koren);

71      return 0;
72  }

```

### Rešenje 4.26

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

6

8  /* Funkcija proverava da li je zadato binarno stablo celih
9     pozitivnih brojeva heap. Ideja koja ce biti implementirana u
10    osnovi ima pronalazenje maksimalne vrednosti levog i
11    maksimalne vrednosti desnog podstabla - ako je vrednost u
12    korenu veca od izracunatih vrednosti uoceni fragment stabla
13    zadovoljava uslov za heap. Zato ce funkcija vratiti
14    maksimalne vrednosti iz uocenog podstabala ili vrednost -1
15    ukoliko se zakljuci da stablo nije heap. */

```

```

16 int heap(Cvor * koren)
17 {
18     int max_levo, max_desno;
19
20     /* Prazno stablo je heap - kao rezultat se vraca 0 kao najmanji
21        pozitivan broj */
22     if (koren == NULL) {
23         return 0;
24     }
25     /* Ukoliko je stablo list... */
26     if (koren->levo == NULL && koren->desno == NULL) {
27         /* Vraca se njegova vrednost */
28         return koren->broj;
29     }
30     /* Inace... */
31
32     /* Proverava se svojstvo za levo podstablo. */
33     max_levo = heap(koren->levo);
34
35     /* Proverava se svojstvo za desno podstablo. */
36     max_desno = heap(koren->desno);
37
38     /* 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 proverava se da li svojstvo vazi za uoceni
45        cvor. */
46     if (koren->broj > max_levo && koren->broj > max_desno) {
47         /* Ako vazi, vraca se vrednost korena */
48         return koren->broj;
49     }
50
51     /* U suprotnom zakljucuje se 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     /* Kreira se stablo koje sadrzi brojeve 100 19 36 17 3 25 1 2
61        7 */
62     koren = NULL;
63     koren = napravi_cvor(100);
64     koren->levo = napravi_cvor(19);
65     koren->levo->levo = napravi_cvor(17);
66     koren->levo->levo->levo = napravi_cvor(2);
67     koren->levo->levo->desno = napravi_cvor(7);

```

```
68 koren->levo->desno = napravi_cvor(3);
koren->desno = napravi_cvor(36);
koren->desno->levo = napravi_cvor(25);
70 koren->desno->desno = napravi_cvor(1);

72 /* Poziva se funkcija kojom se proverava da li je stablo heap */
heap_indikator = heap(koren);
74
/* Ispisuje se rezultat */
76 if (heap_indikator == -1) {
    printf("Zadato tablo nije heap\n");
78 } else {
    printf("Zadato stablo je heap!\n");
80 }

82 /* Oslobadja se memorija zauzeta stablom. */
oslobodi_stablo(&koren);
84
return 0;
86 }
```

Rešenje 4.27

## Glava 5

# Ispitni rokovi

### 5.1 Programiranje 2, praktični deo ispita, jun 2015.

#### Zadatak 5.1

Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost -1 i prekinuti izvršavanje programa.

#### *Primer 1*

```
Poziv: ./a.out ulaz.txt
ULAZNA DATOTEKA (ULAZ.TXT)
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
INTERAKCIJA PROGRAMA:
Ispit
Matematika
Programiranje
```

#### *Primer 2*

```
Poziv: ./a.out ulaz.txt
ULAZNA DATOTEKA (ULAZ.TXT)
2
maksimalano
poena
INTERAKCIJA PROGRAMA:
```

### Primer 3

```
POZIV: ./a.out ulaz.txt
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA PROGRAMA:
-1
```

### Primer 4

```
POZIV: ./a.out
INTERAKCIJA PROGRAMA:
-1
```

[Rešenje 5.1]

### Zadatak 5.2

Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na  $n$ -tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj  $n$ , a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

### Test 1

```
ULAZ:
2 8 10 3 6 14 13 7 4 0
IZLAZ:
20
```

### Test 2

```
ULAZ:
0 50 14 5 2 4 56 8 52 7 1 0
IZLAZ:
50
```

[Rešenje 5.2]

**Zadatak 5.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.



<p><i>Test 1</i></p> <pre> ULAZ: 4 5 1 2 3 4 5 -1 2 -3 4 -5 -5 -4 -3 -2 1 -1 0 0 0 0 IZLAZ: 0 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 2 3 0 0 -5 1 2 -4 IZLAZ: 2 </pre>	<p><i>Test 3</i></p> <pre> ULAZ: -2 IZLAZ: -1 </pre>
--	--	--

[Rešenje 5.3]

## 5.2 Programiranje 2, praktični deo ispita, jul 2015.

### Zadatak 5.4

Napisati program koji kao prvi arugment komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera. Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

<p><i>Primer 1</i></p> <pre> POZIV: ./a.out ulaz.txt test  ULAZNA DATOTEKA (ULAZ.TXT) Ovo je test primer. U njemu se rec test javlja vise puta. testtesttest  INTERAKCIJA PROGRAMA: 5 </pre>	<p><i>Primer 2</i></p> <pre> POZIV: ./a.out  INTERAKCIJA PROGRAMA: (na stderr) -1 </pre>
<p><i>Primer 3</i></p> <pre> POZIV: ./a.out ulaz.txt foo  DATOTEKA ULAZ.TXT NE POSTOJI  INTERAKCIJA PROGRAMA: (na stderr) -1 </pre>	<p><i>Primer 4</i></p> <pre> POZIV: ./a.out ulaz.txt .  ULAZNA DATOTEKA (ULAZ.TXT) JE PRAZNA  INTERAKCIJA PROGRAMA: 0 </pre>

[Rešenje 5.4]

**Zadatak 5.5** Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji 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.

*Primer 1*

```

|| ULAZNA DATOTEKA (TROUGLOVI.TXT)
|| 4
|| 0 0 0 1.2 1 0
|| 0.3 0.3 0.5 0.5 0.9 1
|| -2 0 0 0 0 1
|| -2 0 0 0 0 1
||
|| INTERAKCIJA PROGRAMA:
|| 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

```

*Primer 2*

```

|| ULAZNA DATOTEKA
|| (TROUGLOVI.TXT)
|| 3
|| 1.2 3.2 1.1 4.3
||
|| INTERAKCIJA PROGRAMA:
|| -1

```

*Primer 3*

```

|| DATOTEKA (TROUGLOVI.TXT) NE POSTOJI
||
|| INTERAKCIJA PROGRAMA:
|| -1

```

*Primer 2*

```

|| ULAZNA DATOTEKA
|| (TROUGLOVI.TXT)
|| 0
||
|| INTERAKCIJA PROGRAMA:
||

```

[Rešenje 5.5]

**Zadatak 5.6** Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeba. Napisati funkciju

```
int f3(Cvor *koren, int n)
```

koja u datom stablu prebrojava čvorove na  $n$ -tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou 0. Ispravnost napisane funkcije testirati na osnovu zadate main funkcije i biblioteke za rad sa stablima.

*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 5.6]

## 5.3 Rešenja

### Rešenje 5.1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #define MAX 50
5
6  void greska()
7  {
8      printf("-1\n");
9      exit(EXIT_FAILURE);
10 }
11
12 int main(int argc, char *argv[])
13 {
14     FILE *ulaz;
15     char **linije;
16     int i, j, n;
17
18     /* Proverava argumenata komandne linije. */
19     if (argc != 2) {
20         greska();
21     }
22
23     /* Otvaranje datoteke cije ime je navedeno kao argument
24        komandne linije neposredno nakon imena programa koji se
25        poziva. */
26     ulaz = fopen(argv[1], "r");
27     if (ulaz == NULL) {
28         greska();
29     }
30
31     /* Ucitavanje broja linija. */
32     fscanf(ulaz, "%d", &n);
33
34

```

```
36  /* Alociranje memorije na osnovu ucitanog broja linija. */
    linije = (char **) malloc(n * sizeof(char *));
    if (linije == NULL) {
38      greska();
    }
    for (i = 0; i < n; i++) {
40      linije[i] = malloc(MAX * sizeof(char));
42      if (linije[i] == NULL) {
          for (j = 0; j < i; j++) {
44          free(linije[j]);
          }
46      free(linije);
          greska();
48      }
    }
50
    /* Ucitavanje svih n linija iz datoteke. */
52    for (i = 0; i < n; i++) {
          fscanf(ulaz, "%s", linije[i]);
54    }

56    /* Ispisivanje u odgovarajucem poretku ucitane linije koje
        zadovoljavaju kriterijum. */
58    for (i = n - 1; i >= 0; i--) {
          if (isupper(linije[i][0])) {
60      printf("%s\n", linije[i]);
          }
62    }

64    /* Oslobadjanje memorije koja je dinamicki alocirana. */
    for (i = 0; i < n; i++) {
66      free(linije[i]);
    }

68    free(linije);

70    /* Zatvaranje datoteku. */
72    fclose(ulaz);

74    return 0;
76 }
```

### Rešenje 5.2

```
#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    scanf("%d", &nivo);

24

26    while(1){

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

30        /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
        if(n==0){
32            break;
        }

34        /* A ako nije, dodaje se procitani broj u stablo. */
        dodaj_u_stablo(&koren, n);

36

38    }

40    /* Ispisuje se rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren,nivo));

42

    /* Oslobadja se memorija */
44    oslobodi_stablo(&koren);

46    return 0;
}

```

```

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

5  Cvor* napravi_cvor(int b ) {
    Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
7    if( novi == NULL)
        return NULL;

9

    /* Inicijalizacija polja novog Cvora */

```

```

11     novi->broj = b;
12     novi->levo = NULL;
13     novi->desno = NULL;

14
15     return novi;
16 }

17
18
19 void oslobodi_stablo(Cvor** adresa_korena) {
20     /* Prazno stablo i nema sta da se oslobadja */
21     if( *adresa_korena == NULL)
22         return;
23
24     /* Rekurzivno se oslobadja najpre levo, a onda i desno podstablo */
25     if( (*adresa_korena)->levo )
26         oslobodi_stablo(&(*adresa_korena)->levo);
27     if( (*adresa_korena)->desno )
28         oslobodi_stablo(&(*adresa_korena)->desno);
29
30     free(*adresa_korena);
31     *adresa_korena = NULL;
32 }
33
34
35 void prover_i_alokaciju( Cvor* novi) {
36     if( novi == NULL) {
37         fprintf(stderr, "Malloc greska za nov cvor!\n");
38         exit(EXIT_FAILURE);
39     }
40 }
41
42
43 void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
44     /* Postojece stablo je prazno*/
45     if( *adresa_korena == NULL){
46         Cvor* novi = napravi_cvor(broj);
47         prover_i_alokaciju(novi);
48         *adresa_korena = novi; /* Kreirani Cvor novi ce biti od
49         sada koren stabla*/
50         return;
51     }
52
53     /* Brojevi se smestaju u uredjeno binarno stablo, pa
54     ako je broj koji se ubacuje manji od broja koji je u korenu onda
55     se dodaje u levo podstablo. */
56     if( broj < (*adresa_korena)->broj)
57         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
58     /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
59     dodaje se nov Cvor desno od korena. */
60     else
61         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
62 }

```

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura kojom se predstavlja Cvor stabla */
5  typedef struct dcvor{
6      int broj;
7      struct dcvor* levo, *desno;
8  } Cvor;
9
10 /* Funkcija alocira prostor za novi Cvor stabla, inicijalizuje polja
11    strukture i vraća pokazivac na nov Cvor */
12 Cvor* napravi_cvor(int b );
13
14 /* Funkcija oslobadja dinamički alociran prostor za stablo
15    * Nakon oslobađanja se u pozivajućoj funkciji koren
16    * postavlja NULL, jer je stablo prazno */
17 void oslobodi_stablo(Cvor** adresa_korena);
18
19
20 /* Funkcija proverava da li je novi Cvor ispravno alociran,
21    * i nakon toga prekida program */
22 void prover_i_alokaciju( Cvor* novi);
23
24
25 /* Funkcija dodaje nov Cvor u stablo i
26    * azurira vrednost korena stabla u pozivajućoj funkciji.
27    */
28 void dodaj_u_stablo(Cvor** adresa_korena, int broj);
29
30 #endif

```

### Rešenje 5.3

```

1  #include <stdio.h>
2  #define MAX 50
3
4
5
6  int main()
7  {
8      int m[MAX][MAX];
9      int v, k;
10     int i, j;
11     int max_broj_negativnih, max_indeks_kolone;
12     int broj_negativnih;
13
14     /* Učitavanje dimenzije matrice */
15     scanf("%d", &v);
16     if (v < 0 || v > MAX) {
17         fprintf(stderr, "-1\n");
18         return 0;
19     }

```

```
18 }
20 scanf("%d", &k);
21 if (k < 0 || k > MAX) {
22     fprintf(stderr, "-1\n");
23     return 0;
24 }
25
26 /* Ucitavanje elemenata matrice */
27 for (i = 0; i < v; i++) {
28     for (j = 0; j < k; j++) {
29         scanf("%d", &m[i][j]);
30     }
31 }
32
33 /* Pronalazenje kolone koja sadrzi najveći broj negativnih
34    elemenata */
35 max_indeks_kolone = 0;
36
37 max_broj_negativnih = 0;
38 for (i = 0; i < v; i++) {
39     if (m[i][0] < 0) {
40         max_broj_negativnih++;
41     }
42 }
43
44 for (j = 0; j < k; j++) {
45     broj_negativnih = 0;
46     for (i = 0; i < v; i++) {
47         if (m[i][j] < 0) {
48             broj_negativnih++;
49         }
50         if (broj_negativnih > max_broj_negativnih) {
51             max_indeks_kolone = j;
52         }
53     }
54 }
55
56 }
57
58 /* Ispisivanje traženog rezultata */
59 printf("%d\n", max_indeks_kolone);
60
61 return 0;
62 }
```

#### Rešenje 5.4

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



```

4 #define MAX 128

6 int main(int argc, char **argv)
{
8     FILE *f;
9     int brojac = 0;
10    char linija[MAX], *p;

12    if (argc != 3) {
13        fprintf(stderr, "-1\n");
14        exit(EXIT_FAILURE);
15    }

16    if ((f = fopen(argv[1], "r")) == NULL) {
17        fprintf(stderr, "-1\n");
18        exit(EXIT_FAILURE);
19    }

20    while (fgets(linija, MAX, f) != NULL) {
21        p = linija;
22        while (1) {
23            p = strstr(p, argv[2]);
24            if (p == NULL)
25                break;
26            brojac++;
27            p = p + strlen(argv[2]);
28        }
29    }

30    fclose(f);

31    printf("%d\n", brojac);

32    return 0;
33 }

```

### Rešenje 5.5

```

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

5 typedef struct _trougao {
6     double xa, ya, xb, yb, xc, yc;
7 } trougao;

9 double duzina(double x1, double y1, double x2, double y2) {
10    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
11 }

13 double povrsina(trougao t) {

```

```
double a = duzina(t.xb, t.yb, t.xc, t.yc);
15 double b = duzina(t.xa, t.ya, t.xc, t.yc);
double c = duzina(t.xa, t.ya, t.xb, t.yb);
17 double s = (a + b + c) / 2;
return sqrt(s * (s - a) * (s - b) * (s - c));
19 }

21 int poredi(const void *a, const void *b) {
trougao x = *(trougao*)a;
23 trougao y = *(trougao*)b;
double xp = povrsina(x);
25 double yp = povrsina(y);
if (xp < yp)
27     return 1;
if (xp > yp)
29     return -1;
return 0;
31 }

33 int main() {
FILE *f;
35 int n, i;
trougao *niz;

37 if ((f = fopen("trouglovi.txt", "r")) == NULL) {
39     fprintf(stderr, "-1\n");
exit(EXIT_FAILURE);
41 }

43 if (fscanf(f, "%d", &n) != 1) {
45     fprintf(stderr, "-1\n");
exit(EXIT_FAILURE);
47 }

49 if ((niz = malloc(n * sizeof(trougao))) == NULL) {
51     fprintf(stderr, "-1\n");
exit(EXIT_FAILURE);
53 }

55 for (i = 0; i < n; i++) {
if (fscanf(f, "%lf%lf%lf%lf%lf%lf",
57     &niz[i].xa, &niz[i].ya,
&niz[i].xb, &niz[i].yb,
&niz[i].xc, &niz[i].yc) != 6) {
59     fprintf(stderr, "-1\n");
exit(EXIT_FAILURE);
61 }
}

63 qsort(niz, n, sizeof(trougao), &poredi);

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

```

67     printf("%g %g %g %g %g %g\n",
        niz[i].xa, niz[i].ya,
69     niz[i].xb, niz[i].yb,
        niz[i].xc, niz[i].yc);

71     free(niz);
    fclose(f);
73
    return 0;
75 }

```

### Rešenje 5.6

```

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

4 int f3(Cvor * koren, int n)
{
6     if (koren == NULL || n < 0)
        return 0;
8     if (n == 0) {
        if (koren->levi == NULL && koren->desni != NULL)
10         return 1;
        if (koren->levi != NULL && koren->desni == NULL)
12         return 1;
        return 0;
14     }
    return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
16 }

18 int main()
{
20     Cvor *koren;
    int n;
22
    scanf("%d", &n);
24     koren = ucitaj_stablo();

26     printf("%d\n", f3(koren, n));

28     oslobodi_stablo(&koren);

30     return 0;
}

```

```

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)
        dodaj_u_stablo(&(*koren)->levi, broj);
45

47 }

49 void prikazi_stablo(Cvor * koren)
{
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)
69 {
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 vraca
        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.
21     Cvor se kreira funkcijom napravi_cvor(). */
        void dodaj_u_stablo(Cvor **koren, int broj);
23
        /* Funkcija prikazuje stablo s leva u desno (tj.
        prikazuje elemente u rastucem poretku) */
25     void prikazi_stablo(Cvor * koren);
27

```

```
/* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
   vraca
   pokazican na njegov koren */
29 Cvor* ucitaj_stablo();
31
/* Funkcija oslobadja prostor koji je alociran za
   cvorove stabla. */
33 void oslobodi_stablo(Cvor **koren);
35
#endif
```