

Univerzitet u Beogradu  
Matematički fakultet

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

## **PROGRAMIRANJE 2**

### **Zbirka zadataka sa rešenjima**

**Beograd  
2015.**

Autori:

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

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

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Anđelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu

Studentski trg 16, 11000 Beograd

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

Recenzenti:

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

*dr Mladen Nikolić*, docent na Matematičkom fakultetu u Beogradu

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

Obrada teksta, crteži i korice: *autori*

Štampa:

Tiraž:

CIP Каталогизација у публикацији

Народна библиотека Србије, Београд

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

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



# Predgovor

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

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

...

*Autori*

# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>2</b>
1.1	Podela koda po datotekama . . . . .	2
1.2	Algoritmi za rad sa bitovima . . . . .	5
1.3	Rekurzija . . . . .	10
1.4	Rešenja . . . . .	17
<b>2</b>	<b>Pokazivači</b>	<b>59</b>
2.1	Pokazivačka aritmetika . . . . .	59
2.2	Višedimenzioni nizovi . . . . .	63
2.3	Dinamička alokacija memorije . . . . .	67
2.4	Pokazivači na funkcije . . . . .	72
2.5	Rešenja . . . . .	74
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>110</b>
3.1	Pretraživanje . . . . .	110
3.2	Sortiranje . . . . .	115
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	124
3.4	Rešenja . . . . .	129
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>203</b>
4.1	Liste . . . . .	203
4.2	Stabla . . . . .	213
4.3	Rešenja . . . . .	223
<b>5</b>	<b>Ispitni rokovi</b>	<b>314</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	314
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	316
5.3	Programiranje 2, praktični deo ispita, septembar 2015. . . . .	318
5.4	Rešenja . . . . .	320

# Glava 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

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

- (a) Definirati strukturu `KompleksanBroj` koja opisuje kompleksan broj njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `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 vraća vrednost realnog dela broja.
- (e) Napisati funkciju `imaginarni_deo` koja vraća vrednost imaginarnog dela broja.
- (f) Napisati funkciju `moduo` koja računa moduo 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.

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

Napisati program koji testira prethodno napisane funkcije. Program najpre za kompleksan broj  $z_1$  koji se unosi sa standardnog ulaza ispisuje njegov realni deo, imaginarni deo i moduo. Zatim za naredni kompleksan broj  $z_2$  koji se unosi sa standardnog ulaza ispisuje njegov konjugovano-kompleksan broj i argument. Na kraju program 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 biblioteku za rad sa polinomima.

- (a) Definirati strukturu `Polinom` koja opisuje polinom stepena najviše 20. UPUTSTVO: *Struktura sadrži stepen i niz koeficijenata. Redosled navođenja koeficijenata u nizu treba da bude takav da na nultoj poziciji u nizu bude koeficijent uz slobodan član, na prvoj koeficijent uz prvi stepen, itd.*
- (b) Napisati funkciju koja ispisuje polinom na standardni izlaz.
- (c) Napisati funkciju koja učitava polinom sa standardnog ulaza. Za polinom se najpre unosi stepen pa njegovi koeficijenti.

- (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. Program najpre učitava polinom  $p$  sa standardnog ulaza i ispisuje ga na standardni izlaz u odgovarajućem obliku. Nakon toga program računa i ispisuje vrednost tog polinoma (zaokruženu na dve decimale) u tački koju unosi korisnik. Potom se unosi polinom  $q$ , i ispisuju se zbir i proizvod polinoma  $p$  i  $q$ . Na kraju se sa standardnog ulaza unosi broj  $n$ , i 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
Prozvod 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** Napisati biblioteku za rad sa razlomcima.

- (a) Definirati strukturu za reprezentovanje razlomaka.
- (b) Napisati funkcije za učitavanje i ispis razlomaka.
- (c) Napisati funkcije koje vraćaju brojilac i imenilac razlomka.
- (d) Napisati funkciju koja vraća odgovarajuću realnu vrednost razlomka (tipa `double`).
- (e) Napisati funkciju koja izračunava recipročnu vrednost razlomka.
- (f) Napisati funkciju koja skraćuje dati razlomak.
- (g) Napisati funkcije koje sabiraju, oduzimaju, množe i dele dva razlomka.





Napisati program koji testira te funkcije za brojeve koji se zadaju u heksadekaskom formatu sa standardnog ulaza.

### 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:
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 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 zadaju u heksadekaskom formatu sa standardnog ulaza.

### Test 1

```
ULAZ:
0x7F
IZLAZ:
Najveci:
11111110000000000000000000000000
Najmanji:
00000000000000000000000001111111
```

### Test 2

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

### Test 3

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

### Test 4

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

[Rešenje 1.7]

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

- Napisati funkciju koja određuje broj koji se dobija kada se  $n$  bitova datog broja, 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  $p$ .
- Napisati funkciju koja vraća broj koji se dobija invertovanjem  $n$  bitova broja  $x$  počevši od pozicije  $p$ .

Napisati program koji 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 bit najmanje težine na poziciji nula.*

*Test 1*

```
ULAZ:  
    235 5 10 127  
  
IZLAZ:  
    Broj 235 = 000000000000000000000000000011101011  
    reset(235, 5, 10) = 0000000000000000000000000000101011  
    set(235, 5, 10) = 00000000000000000000000000001111101011  
    get_bits(235, 5, 10) = 0000000000000000000000000000000011  
    y = 127 = 0000000000000000000000000000000011111111  
    set_n_bits(235, 5, 10, 127) = 000000000000000000000000000011111101011  
    invert(235, 5, 10) = 0000000000000000000000000000000011100101011
```

[Rešenje 1.8]

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

- (a) Napisati funkciju `rotate_left` koja određuje broj koji se dobija rotiranjem `k` puta ulevo datog celog broja `x`.
- (b) Napisati funkciju `rotate_right` koja određuje broj koji se dobija rotiranjem `k` puta udesno datog celog neoznačenog broja `x`.

- (c) Napisati funkciju `rotate_right_signed` koja određuje broj koji se dobija rotiranjem `k` puta udesno datog celog broja `x`.

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

*Test 1*

```

|| ULAZ:
|| B10011A7 5
|| IZLAZ:
|| x = 10110001000000000001000110100111
|| rotate_left(2969571751, 5) = 0010000000000100011010011110110
|| rotate_right(2969571751, 5) = 00111101100010000000000010001101
|| rotate_right_signed(2969571751, 5) = 0011110110001000000000010001101

```

[Rešenje 1.9]

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

*Test 1*

```

|| ULAZ:
|| 255
|| IZLAZ:
|| 0000000000000000000000001001010101
|| 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:
|| 2147377146
|| IZLAZ:
|| 1

```

*Test 3*

```

|| ULAZ:
|| 1111111115
|| IZLAZ:
|| 0

```

[Rešenje 1.11]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           ULAZ: 11 IZLAZ: 1         </div> </div>	<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           ULAZ: 1024 IZLAZ: 0         </div> </div>	<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           ULAZ: 2147377146 IZLAZ: 22         </div> </div>

[Rešenje 1.12]

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

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 2</i>
<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           Poziv: ./a.out 1 2             INTERAKCIJA PROGRAMA: 11 13         </div> </div>	<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           Poziv: ./a.out 1 2             INTERAKCIJA PROGRAMA: 1024 1024         </div> </div>	<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           Poziv: ./a.out 12 12             INTERAKCIJA PROGRAMA: 12345 12345         </div> </div>

**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>
<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           ULAZ: 11 IZLAZ: 0000000B         </div> </div>	<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           ULAZ: 1024 IZLAZ: 00000400         </div> </div>	<div style="display: inline-block; vertical-align: top;"> <div style="display: inline-block; vertical-align: top; margin-right: 5px;">  </div> <div style="display: inline-block; vertical-align: top;">           ULAZ: 12345 IZLAZ: 00003039         </div> </div>

[Rešenje 1.14]

**Zadatak 1.15** Napisati funkciju koja za data dva neoznačena broja  $x$  i  $y$  invertuje u podatku  $x$  one bitove koji se poklapaju sa odgovarajućim bitovima

u broju  $y$ . Ostali bitovi ostaju nepromenjeni. Napisati program koji tu funkciju testira za brojeve koji se zadaju sa standardnog ulaza.

Test 1	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. NAPOMENA: *Zadatak rešiti isključivo korišćenjem bitskih operatora.*

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:

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

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

*Test 1*

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

*Test 2*

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

[Rešenje 1.18]

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

*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 napisanu funkciju za vrednosti koeficijenata i prirodan broj  $n$  koji se unose 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.

*Test 1*

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

*Test 2*

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

*Test 3*

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

*Test 4*

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

*Test 5*

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

[Rešenje 1.21]

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

*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. Njegovi elementi se unose sve do unosa kraja ulaza (EOF).

*Test 1*

```
|| ULAZ:
|| 3 2 1 4 21
|| IZLAZ:
|| 21
```

*Test 2*

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

*Test 3*

```
|| ULAZ:
|| 1 11 3 5 8 1
|| IZLAZ:
|| 11
```

[Rešenje 1.23]

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>    ULAZ:   3 1 2 3 1 2 3   IZLAZ:   14 </pre>	<pre>    ULAZ:   2 3 5 2 6   IZLAZ:   36 </pre>	<pre>    ULAZ:   0   IZLAZ:   0 </pre>

[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 broj  $x$  i niz  $a$  koji se unose sa standardnog ulaza. Prvo se unosi  $x$ , a zatim elementi niza sve do unosa kraja ulaza. Niz neće imati više od 256 elemenata.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>    ULAZ:   4 1 2 3 4   IZLAZ:   1 </pre>	<pre>    ULAZ:   11 3 2 11 14 11 43 1   IZLAZ:   2 </pre>	<pre>    ULAZ:   1 3 21 5 6   IZLAZ:   0 </pre>

[Rešenje 1.25]

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

<i>Test 1</i>	<i>Test 2</i>
<pre>    ULAZ:   1 2 3 4 1 2 3 4 5   IZLAZ:   da </pre>	<pre>    ULAZ:   1 2 3 11 1 2 4 3 6   IZLAZ:   ne </pre>

[Rešenje 1.26]



[Rešenje 1.27]

[Rešenje 1.29]

*Test 1*

```

|| ULAZ:
|| 5
|| IZLAZ:
|| 5

```

*Test 2*

```

|| ULAZ:
|| 16
|| IZLAZ:
|| 1

```

*Test 3*

```

|| ULAZ:
|| 18
|| IZLAZ:
|| 2

```

[Rešenje 1.30]

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

*Test 1*

```

|| ULAZ:
|| a
|| IZLAZ:
|| da

```

*Test 2*

```

|| ULAZ:
|| aa
|| IZLAZ:
|| da

```

*Test 3*

```

|| ULAZ:
|| aba
|| IZLAZ:
|| da

```

*Test 4*

```

|| ULAZ:
|| programiranje
|| IZLAZ:
|| ne

```

*Test 5*

```

|| ULAZ:
|| anavolimilovana
|| IZLAZ:
|| da

```

[Rešenje 1.31]

**\* Zadatak 1.32** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za proizvoljan 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 sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbira jednog polja levo i jednog polja iznad. Izuzetak su jedinice na krajevima. Vrednosti brojeva Paskalovog trougla odgovaraju binomnim koeficijentima tj. vrednost polja  $(n, k)$ , gde je  $n$  redni broj hipotenuze, a  $k$  redni broj elementa u tom redu (na toj hipotenuzi) odgovara binomnom koeficijentu  $\binom{n}{k}$ , pri čemu brojanje počinje od nule. Na primer, vrednost polja  $(4, 2)$  je 6.

```

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

```

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

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

*Test 1*

```

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

```

[Rešenje 1.33]

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

*Test 1*

```

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

```

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

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

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

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

## 1.4 Rešenja

### Rešenje 1.1

```

1 #include <stdio.h>
  #include <math.h>
3
4 /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
5    imaginaran deo kompleksnog broja */
6 typedef struct {
7     float real;

```

```
9     float imag;
10 } KompleksanBroj;

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

21 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
22    se salje kao argument u obliku (x + i y) Ovoj funkciji se
23    kompleksan broj prenosi po vrednosti (za ispis nije neophodna
24    adresa) */
25 void ispisi_kompleksan_broj(KompleksanBroj z)
26 {
27     printf("(");
28     if (z.real != 0) {
29         printf("%.2f", z.real);
30         if (z.imag > 0)
31             printf(" +");
32     }
33     if (z.imag != 0)
34         printf(" %.2f i ", z.imag);
35
36     if (z.imag == 0 && z.real == 0)
37         printf("0 ");
38
39     printf(")");
40 }

43 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
44 float realan_deo(KompleksanBroj z)
45 {
46     return z.real;
47 }

49 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
50 float imaginaran_deo(KompleksanBroj z)
51 {
52     return z.imag;
53 }

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

```

    }
61
    /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
63     odgovara kompleksnom broju poslatom kao argument */
    KompleksanBroj konjugovan(KompleksanBroj z)
65    {
        KompleksanBroj z1 = z;
67
        z1.imag *= -1;
69
        return z1;
71    }

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

79        z.real += z2.real;
        z.imag += z2.imag;
81
        return z;
83    }

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

91        z.real -= z2.real;
        z.imag -= z2.imag;
93
        return z;
95    }

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

103        z.real = z1.real * z2.real - z1.imag * z2.imag;
        z.imag = z1.real * z2.imag + z1.imag * z2.real;
105
        return z;
107    }

109    /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
        kao argument */
111    float argument(KompleksanBroj z)
```

```
113 {
    return atan2(z.imag, z.real);
115 }
117 int main()
118 {
119     /* Deklaracija 2 promenljive tipa KompleksanBroj */
120     KompleksanBroj z1, z2;
121
122     /* Ucitavanje prvog kompleksnog broja u promenljivu z1, a potom
123        njegovo ispisivanje na standardni izlaz */
124     ucitaj_kompleksan_broj(&z1);
125     ispisi_kompleksan_broj(z1);
126
127     /* Ispisuje se na standardni izlaz realan, imaginaran deo i moduo
128        kompleksnog broja z1 */
129     printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
130           realan_deo(z1), imaginaran_deo(z1), moduo(z1));
131     printf("\n");
132
133     /* Ucitavanje drugog kompleksnog broja u promenljivu z2, a potom
134        njegovo ispisivanje na standardni izlaz */
135     ucitaj_kompleksan_broj(&z2);
136     ispisi_kompleksan_broj(z2);
137
138     /* Racunanje i ispisivanje konjugovano kompleksan broj od z2 */
139     printf("\nNjegov konjugovano kompleksan broj: ");
140     ispisi_kompleksan_broj(konjugovan(z2));
141     printf("\n");
142
143     /* Sabiranje kompleksnih brojeva */
144     printf("\n");
145     ispisi_kompleksan_broj(z1);
146     printf(" + ");
147     ispisi_kompleksan_broj(z2);
148     printf(" = ");
149     ispisi_kompleksan_broj(saberi(z1, z2));
150     printf("\n");
151
152     /* Oduzimanje kompleksnih brojeva */
153     printf("\n");
154     ispisi_kompleksan_broj(z1);
155     printf(" - ");
156     ispisi_kompleksan_broj(z2);
157     printf(" = ");
158     ispisi_kompleksan_broj(oduzmi(z1, z2));
159     printf("\n");
160
161     /* Mnozenje kompleksnih brojeva */
162     printf("\n");
163     ispisi_kompleksan_broj(z1);
164     printf(" * ");
```

```

165     ispisi_kompleksan_broj(z2);
    printf(" = ");
    ispisi_kompleksan_broj(mnozi(z1, z2));
167
    /* Testiranje funkcije koja racuna argument kompleksnih brojeva */
169     printf("\n");
    ispisi_kompleksan_broj(z2);
171     printf("\nArgument kompleksnog broja %f\n", argument(z2));
173
    return 0;
}

```

## Rešenje 1.2

```

1  /* Ukljucuje se zaglavlje neophodno za rad sa kompleksnim brojevima.
   Ovdje je to neophodno jer nam je neophodno da bude poznata
3  definicija tipa KompleksanBroj. Takodje, time su ukljucena
   zaglavlja standardne biblioteke koja su navedena u complex.h */
5  #include "complex.h"

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

17 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
   se salje kao argument u obliku (x + y i) */
19 void ispisi_kompleksan_broj(KompleksanBroj z)
{
21     printf("(");
    if (z.real != 0) {
23         printf("%.2f", z.real);

25         if (z.imag > 0)
            printf(" + %.2f i", z.imag);
27         else if (z.imag < 0)
            printf(" - %.2f i", -z.imag);
29     } else
        printf("%.2f i", z.imag);

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

35     printf(")");
}
37

```



```
/* Funkcija vraca vrednosti realnog dela kompleksnog broja */
39 float realan_deo(KompleksanBroj z)
{
41     return z.real;
43 }

/* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
45 float imaginaran_deo(KompleksanBroj z)
{
47     return z.imag;
49 }

/* Funkcija vraca vrednost modula kompleksnog broja koji joj se salje
51 kao argument */
float moduo(KompleksanBroj z)
53 {
55     return sqrt(z.real * z.real + z.imag * z.imag);
57 }

/* Funkcija vraca vrednost konjugovano kompleksnog broja koji
odgovara kompleksnom broju poslatom kao argument */
59 KompleksanBroj konjugovan(KompleksanBroj z)
{
61     KompleksanBroj z1 = z;
63     z1.imag *= -1;
65     return z1;
67 }

/* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
argumenata funkcije */
KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
69 {
71     KompleksanBroj z = z1;
73     z.real += z2.real;
75     z.imag += z2.imag;
77     return z;
79 }

/* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
argumenata funkcije */
KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
81 {
83     KompleksanBroj z = z1;
85     z.real -= z2.real;
87     z.imag -= z2.imag;
89     return z;
91 }
```

```

91  /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
    argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
93  {
    KompleksanBroj z;

95      z.real = z1.real * z2.real - z1.imag * z2.imag;
97      z.imag = z1.real * z2.imag + z1.imag * z2.real;

99      return z;
101  }

/* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
103  kao argument */
float argument(KompleksanBroj z)
105  {
    return atan2(z.imag, z.real);
107  }

```

```

1  /*
    Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
3  deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavlje
    nikada ne treba da sadrzi definicije funkcija. Da bi neki program
5  mogao da koristi ove brojeve i funkcije iz ove biblioteke,
    neophodno je da ukljuci ovo zaglavlje. */

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

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

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

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

29
void ispisi_kompleksan_broj(KompleksanBroj z);

31
float realan_deo(KompleksanBroj z);
33

```

```

float imaginaran_deo(KompleksanBroj z);
35 float moduo(KompleksanBroj z);
37 KompleksanBroj konjugovan(KompleksanBroj z);
39 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
41 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
43 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
45 float argument(KompleksanBroj z);
47 /* Kraj zakljucanog dela */
49 #endif

```

---

```

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

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

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

15

17 #include <stdio.h>
   /* Ukljucuje aw zaglavlje neophodno za rad sa kompleksnim brojevima
   */
19 #include "complex.h"

21 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
   polarni oblik */
23 int main()
   {
25     KompleksanBroj z;

27     /* Ucitavamo kompleksan broj */
     ucitaj_kompleksan_broj(&z);

29     printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f~n",
31           moduo(z), argument(z));

33     return 0;
   }

```

## Rešenje 1.3

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "polinom.h"
4
5
6  /* Funkcija koja ispisuje polinom na standardan izlaz u citljivom
7   obliku. Kako bi uštedeli kopiranje cele strukture, polinom
8   prenosimo po adresi */
9  void ispisi(const Polinom * p)
10 {
11     int i;
12     for (i = p->stepen; i >= 0; i--) {
13         if (p->koef[i]) {
14             if (p->koef[i] >= 0 && i != p->stepen)
15                 putchar('+');
16             if (i > 1)
17                 printf("%.2fx^%d", p->koef[i], i);
18             else if (i == 1)
19                 printf("%.2fx", p->koef[i]);
20             else
21                 printf("%.2f", p->koef[i]);
22         }
23     }
24     putchar('\n');
25 }
26
27 /* Funkcija koja ucitava polinom sa tastature */
28 Polinom ucitaj()
29 {
30     int i;
31     Polinom p;
32
33     /* Ucitavamo stepen polinoma */
34     scanf("%d", &p.stepen);
35
36     /* Ponavljamo učitavanje stepena sve dok ne unesemo stepen iz
37      dozvoljenog opsega */
38     while (p.stepen > MAX_STEPEN || p.stepen < 0) {
39         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
40         scanf("%d", &p.stepen);
41     }
42
43     /* Unosimo koeficijente polinoma */
44     for (i = p.stepen; i >= 0; i--)
45         scanf("%lf", &p.koef[i]);
46     return p;
47 }
```

```

49  /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
    algoritmom */
51  /*  $x^4+2x^3+3x^2+2x+1 = ((x+2)x + 3)x + 2)x + 1$  */
    double izracunaj(const Polinom * p, double x)
53  {
        double rezultat = 0;
55     int i = p->stepen;
        for (; i >= 0; i--)
57         rezultat = rezultat * x + p->koeff[i];
        return rezultat;
59  }

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

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

69     for (i = 0; i <= rez.stepen; i++)
        rez.koeff[i] =
71         (i > p->stepen ? 0 : p->koeff[i]) + (i >
                                                    q->
73         stepen ? 0 : q->koeff[i]);

        return rez;
75  }

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

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

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

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

93         return r;
95     }
97 }
99

```

```

101 /* Funkcija racuna izvod polinoma p */
102 Polinom izvod(const Polinom * p)
103 {
104     int i;
105     Polinom r;
106
107     if (p->stepen > 0) {
108         r.stepen = p->stepen - 1;
109
110         for (i = 0; i <= r.stepen; i++)
111             r.koef[i] = (i + 1) * p->koef[i + 1];
112     } else
113         r.koef[0] = r.stepen = 0;
114
115     return r;
116 }
117
118 /* Funkcija racuna n-ti izvod polinoma p */
119 Polinom nIzvod(const Polinom * p, int n)
120 {
121     int i;
122     Polinom r;
123
124     if (n < 0) {
125         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
126         exit(EXIT_FAILURE);
127     }
128
129     if (n == 0)
130         return *p;
131
132     r = izvod(p);
133     for (i = 1; i < n; i++)
134         r = izvod(&r);
135
136     return r;
137 }

```

```

2  /* Ovim pretrocesorskim direktivama zakljucavamo zaglavlje i time
3     onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci */
4  #ifndef _POLINOM_H
5  #define _POLINOM_H
6
7  #include <stdio.h>
8  #include <stdlib.h>
9
10 /* Maksimalni stepen polinoma */
11 #define MAX_STEPEN 20
12
13
14 /* Polinome predstavljamo strukturom koja cuva koeficijente (koef[i]

```

```

    je koeficijent uz clan x^i) i stepen polinoma */
16 typedef struct {
    double koef[MAX_STEPEN + 1];
18     int stepen;
    } Polinom;

20
22 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku
    Polinom prenosimo po adresi, da bi uštedeli kopiranje cele
    strukture, vec samo prenosimo adresu na kojoj se nalazi polinom
24     kog ispisujemo */
    void ispisi(const Polinom * p);

26
28 /* Funkcija koja ucitava polinom sa tastature */
    Polinom ucitaj();

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

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

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

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

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

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

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

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

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

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

```

```

20  p = učitaj();

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

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

26

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

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

34

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

38

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

42

44

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

48

50

52  /* Uspesno završavamo program */
    return 0;

54 }

```

## Rešenje 1.5

```

2  #include <stdio.h>

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

10     /* Broj bitova celog broja */
        unsigned velicina = sizeof(unsigned) * 8;

```



```

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

14

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 tako sto
18  se jedini bit jedinica pomera udesno, kako bi se ocitao naredni
    bit broja x koji je argument funkcije. Odgovarajuci karakter,
    ('0' ili '1'), ispisuje se na standardnom izlazu. Neophodno je
20  da promenljiva maska bude deklarirana kao neoznacena ceo broj
    kako bi se siftovanjem u desno vrsilo logicko siftovanje
    (popunjavanje nulama) a ne aritmeticko siftovanje (popunjavanje
22  znakom broja). */
    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
24  putchar(x & maska ? '1' : '0');

26

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 odgovarajuće maske i njenim pomeranjem */
18  int count_bits1(int x)
    {
20     int br = 0;

```

```

22     unsigned wl = sizeof(unsigned) * 8 - 1;

24     /* Formiranje se maska čija binarna reprezentacija izgleda
    1000000...0000000, koja služi za očitavanje bita najveće težine.
    U svakoj iteraciji maska se pomera u desno za 1 mesto, i
    26     očitavamo sledeći bit. Petlja se završava kada više nema
    jedinica tj. kada maska postane nula. */
28     unsigned maska = 1 << wl;
    for (; maska != 0; maska >>= 1)
30         x & maska ? br++ : 1;

32     return br;
    }

34     /* Funkcija vraća broj jedinica u binarnoj reprezentaciji broja x
    formiranjem odgovarajuće maske i pomeranjem promenljive x */
36     int count_bits2(int x)
38     {
        int br = 0;
        unsigned wl = sizeof(int) * 8 - 1;

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

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

52         return br;
    }

54 }

56 int main()
58 {
    int x;
    60     scanf("%x", &x);
    printf("Broj jedinica u zapisu je\n");
    62     printf("funkcija count_bits1: %d\n", count_bits1(x));
    printf("funkcija count_bits2: %d\n", count_bits2(x));
    64     return 0;
}

```

### Rešenje 1.7

```

1  #include <stdio.h>

3  /* Funkcija vraća najveći neoznačeni broj sastavljen od istih bitova

```

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

9     /* Formira se maska 100000...00000000 */
    unsigned maska = 1 << (velicina - 1);

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

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

25     return rezultat;
27 }

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

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

47     return rezultat;
49 }

51 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
53 */
55
```

```

        celog broja u memoriji */
57 void print_bits(int x)
    {
59     unsigned velicina = sizeof(int) * 8;
        unsigned maska;

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

65     putchar('\n');
    }

67 int main()
69 {
    int broj;
71     scanf("%x", &broj);

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

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

77     return 0;
79 }

```

## 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 */
8 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 postavljenih na 0 pocev od pozicije p, dok su svi ostali
    postavljeni na 1.

16

    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
33 *****/
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
43 *****/
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
57 *****/
58 Funkcija vraca celobrojno polje bitova, desno poravnato, koje
   predstavlja n bitova pocev od pozicije p u binarnoj
   reprezentaciji broja x, pri cemu se pozicija broji sa desna
60 ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
   i broj cija je binarna reprezentacija:
62   1010 1011 1100 1101 1110 1010 1110 0111
   trazi se
64   0000 0000 0000 0000 0000 0000 0000 1011
   *****/
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
78  polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
   zeljenih n i 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  /******
   Kreira se maska kod kod koje su poslednjih n bitova 1, a
90  ostali su 0. Na primer za n=5
   0000 0000 0000 0000 0000 0001 1111
92  *****/
   unsigned last_n_1 = ~(~0 << n);
94  /******
   Kao sto je i u funkciji reset, i ovde se kreira masku koja ima n
96  bitova postavljenih na 0 pocevsi od pozicije p, dok su
   ostali bitovi 1. Na primer za n=5 i p =10
98  1111 1111 1111 1111 1111 1000 0011 1111
   *****/
100  unsigned middle_n_0 = ~(~0 << n) << (p - n + 1);

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

106
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
110  bitovi su nule. (y & last_n_1) Resetuju se svi bitovi osim
   najnizih n */
112  unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);

114  return x_reset ^ y_shift_middle;
   }

116
118  /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
   njih n */
120  unsigned invert(unsigned x, unsigned n, unsigned p)
   {
122  /******
   Formira se maska sa n jedinica pocev od pozicije p.
124  Na primer za n=5 i p=10
   0000 0000 0000 0000 0000 0111 1100 0000
126  *****/
   unsigned maska = ~(~0 << n) << (p - n + 1);

```

```
128     /* Operator ekskluzivno ili invertuje sve bitove gde je
130        odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
131     return maska ^ x;
132 }
133
134 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
135    celog broja u memoriji */
136 void print_bits(int x)
137 {
138     unsigned velicina = sizeof(int) * 8;
139     unsigned maska;
140
141     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
142         putchar(x & maska ? '1' : '0');
143
144     putchar('\n');
145 }
146
147
148
149
150 int main()
151 {
152     unsigned broj, p, n, y;
153     scanf("%u%u%u%u", &broj, &n, &p, &y);
154     printf("Broj %u %s= ", broj, "");
155     print_bits(broj);
156
157
158     printf("reset(%u,%u,%u)%s = ", broj, n, p, "");
159     print_bits(reset(broj, n, p));
160
161     printf("set(%u,%u,%u)%s = ", broj, n, p, "");
162     print_bits(set(broj, n, p));
163
164     printf("get_bits(%u,%u,%u)%s = ", broj, n, p, "");
165     print_bits(get_bits(broj, n, p));
166
167     printf("y = %u = ", y);
168     print_bits(y);
169     printf("set_n_bits(%u,%u,%u,%u) = ", broj, n, p, y);
170     print_bits(set_n_bits(broj, n, p, y));
171
172     printf("invert(%u,%u,%5u)%s = ", broj, n, p, "");
173     print_bits(invert(broj, n, p));
174
175     return 0;
176 }
```

## 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 najviši bit postavljen na 1 neophodna da bi
17     pre siftovanja u levo za 1 najviši bit bio sačuvan. */
        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
23     trenutne vrednosti promenljive x u levo za 1. Nakon toga, potom
        najniži bit se postavlja na vrednost koju je imao prvi bit koji
25     je istisnut siftovanjem */
        for (i = 0; i < n; i++) {
27             first_bit = x & first_bit_mask;
                x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
29         }
        return x;
31     }

33  /*****
   Funkcija neoznačen broj x rotira u desno za n.
35  Na primer za n=5 i x čija je binarna reprezentacija
   1010 1011 1100 1101 1110 0001 0010 0011
37  funkcija vraća neoznačen ceo broj čija je binarna
   reprezentacija:
39  0001 1101 0101 1110 0110 1111 0000 1001
   *****/
41  unsigned rotate_right(unsigned x, unsigned n)
   {
43     unsigned last_bit;
        int i;

45     /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
        iteraciji se određuje bit najmanje težine broja x, zatim tako
47     određeni bit se siftuje u levo tako da najniži bit dođe do
        pozicije najvišeg bita. Zatim, nakon siftovanja binarne
49     reprezentacije trenutne vrednosti promenljive x za 1 u desno,
        najviši bit se postavlja na vrednost već zapamćenog bita koji je
51

```



```

    bio na poziciji najmanje tezine. */
53 for (i = 0; i < n; i++) {
    last_bit = x & 1;
55     x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
    }
57
    return x;
59 }

61 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
    argument funkcije x oznaceni ceo broj */
63 int rotate_right_signed(int x, unsigned n)
{
65     unsigned last_bit;
    int i;
67

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

79         if (x < 0)
            /*****
81             Siftovanjem u desno broja koji je negativan dobija se 1
                kao bit na najvisoj poziciji. Na primer ako je x
83             1010 1011 1100 1101 1110 0001 0010 001b
                (sa b je oznacen ili 1 ili 0 na najnižoj poziciji)
85             Onda je sadržaj promenljive last_bit:
                0000 0000 0000 0000 0000 0000 0000 000b
87             Nakon siftovanja sadržaja promenljive x za 1 u desno
                1101 0101 1110 0110 1111 0000 1001 0001
89             Kako bi umesto 1 na najvisoj poziciji u trenutnoj
                binarnoj reprezentaciji x bilo postavljeno b nije
91             dovoljno da se siftuje na najvisu poziciju jer bi se
                time dobile 0, a u ovom slucaju su potrebne jedinice
93             zbog bitovskog & zato se prvo vrsi komplementiranje, a
                zatim siftovanje
95             ~last_bit << (sizeof(int)*8 -1)
                B000 0000 0000 0000 0000 0000 0000 0000
97             gde B oznacava ~b.
                Potom se ponovo vrsi komplementiranje kako bi se b
99             nalazilo na najvisoj poziciji i sve jedinice na ostalim
                pozicijama
101             ~(~last_bit << (sizeof(int)*8 -1))
                b111 1111 1111 1111 1111 1111 1111 1111
103 *****/

```

```

105     x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
    else
107         x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
    }

109     return x;
    }
111

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

121     putchar('\n');
    }
123

125 int main()
    {
127     unsigned x, k;
    scanf("%x%x", &x, &k);
129     printf("x %s = ", "");
    print_bits(x);
131     printf("rotate_left(%u,%u)%s = ", x, k, "");
    print_bits(rotate_left(x, k));

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

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

    return 0;
141 }

```

### Rešenje 1.10

```

1  #include <stdio.h>

3  /*****
    Funkcija vraca vrednost cija je binarna reprezentacija slika
5  u ogledalu binarne reprezentacije broja x koji se prosledjuje
    kao argument funkcije. Na primer za x
7  cija binarna reprezentacija izgleda ovako
    101010111100110111100100100100011
9  funkcija treba da vrati broj cija binarna reprezentacija
    izgleda:

```

```

11      11000100100001111011001111010101
*****
13 unsigned mirror(unsigned x)
{
15     unsigned najnizi_bit;
    unsigned rezultat = 0;
17
    int i;
19     /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
        vrednosti broja x se određuje i pamti u promenljivoj
21     najnizi_bit, nakon čega se na promenljivu x primeni siftovanje u
        desno. */
23     for (i = 0; i < sizeof(x) * 8; i++) {
        najnizi_bit = x & 1;
25         x >>= 1;
        /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
27         postavljeni bitovi dobijaju veću poziciju. Novi bit se
            postavlja na najnižu poziciju */
29         rezultat <<= 1;
        rezultat |= najnizi_bit;
31     }
    return rezultat;
33 }

35
/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
37 celog broja u memoriji */
void print_bits(int x)
39 {
    unsigned velicina = sizeof(int) * 8;
    unsigned maska;
41     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
43         putchar(x & maska ? '1' : '0');

    putchar('\n');
45 }
47
int main()
49 {
    int broj;
51     scanf("%x", &broj);

53     /* Ispisuje se binarna reprezentaciju unetog broja */
    print_bits(broj);

55     /* Ispisuje se binarna reprezentaciju broja dobijenog pozivom
        funkcije mirror */
57     print_bits(mirror(broj));

59     return 0;
61 }

```

## Rešenje 1.11

```
1  #include <stdio.h>
2
3  /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
4     jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
5  int Broj01(unsigned int n)
6  {
7
8     int broj_nula, broj_jedinica;
9     unsigned int maska;
10
11     broj_nula = 0;
12     broj_jedinica = 0;
13
14     /* Maska je inicijalizovana tako da moze da analizira bit najvece
15        tezine */
16     maska = 1 << (sizeof(unsigned int) * 4 - 1);
17
18     /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
19        iteraciji pomera u desno pa ce jedini bit koji je postavljen na
20        1 biti na svim pozicijama u binarnoj reprezentaciji maske */
21     while (maska != 0) {
22
23         /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili
24            1 i uveca se odgovarajuci brojac */
25         if (n & maska) {
26             broj_jedinica++;
27         } else {
28             broj_nula++;
29         }
30
31         /* Pomera se maska u desnu stranu */
32         maska = maska >> 1;
33     }
34
35     /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
36        suprotnom vraca 0 */
37     return (broj_jedinica > broj_nula) ? 1 : 0;
38 }
39
40 int main()
41 {
42     unsigned int n;
43
44     scanf("%u", &n);
45
46     printf("%d\n", Broj01(n));
47
48     return 0;
49 }
50 }
```

## Rešenje 1.12

```
1  #include <stdio.h>
2
3  int broj_parova(unsigned int x)
4  {
5
6      int broj_parova;
7      unsigned int maska;
8
9      /* Vrednost promenljive koja predstavlja broj parova se
10       inicijalizuje na 0 */
11      broj_parova = 0;
12
13      /* Postavlja se maska tako da moze da procitamo da li su dva
14       najmanja bita u zapisu broja x 11 */
15      /* Binarna reprezentacija broja 3 je 000...00011 */
16      maska = 3;
17
18      while (x != 0) {
19
20          /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
21           */
22          if ((x & maska) == maska) {
23              broj_parova++;
24          }
25
26          /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
27           proveravao sledeci par bitova. Pomeranjem u desno bit najvece
28           tezine se popunjava nulom jer je x neoznaceni broj. */
29          x = x >> 1;
30      }
31
32      return broj_parova;
33  }
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 1.14

```

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 #define MAX_DUZINA sizeof(unsigned int)*2 +1
11
12 void prevod(unsigned int x, char s[])
13 {
14
15     int i;
16     unsigned int maska;
17     int vrednost;
18
19     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
20        maska za citanje 4 uzastopne cifre */
21     maska = 15;
22
23     /******
24        Broj se posmatra od pozicije najmanje tezine ka poziciji
25        najvece tezine. Na primer za broj
26        00000000001101000100001111010101
27        u prvom koraku se citaju bitovi izdvojeni sa <...>:
28        0000000000110100010000111101<0101>
29        u drugom koraku:
30        0000000000011010001000011<1101>0101
31        u trecem koraku:
32        00000000001101000100<0011>11010101 i tako redom
33
34        Indeks i oznacava poziciju na koju se smesta vrednost.
35
36     */
37     for (i = MAX_DUZINA - 2; i >= 0; i--) {
38         /* Vrednost izdvojene cifre */
39         vrednost = x & maska;
40
41         /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
42            dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
43            10 do 15 odgovarajuci karakter se dobija tako sto se prvo
44            oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na tako
45            dobijenu vrednost doda ASCII kod 'A' (time se dobija
46            odgovarajuce slovo 'A', 'B', ... 'F') */
47         if (vrednost < 10) {
48             s[i] = vrednost + '0';
49         } else {
50

```

```

52     s[i] = vrednost - 10 + 'A';
    }

54     /* Primenljiva x se pomera za 4 bita u desnu stranu i time ce u
56     narednoj iteraciji biti posmatrane sledece 4 cifre */
    x = x >> 4;
    }

58     s[MAX_DUZINA - 1] = '\0';
60 }

62 int main()
63 {
64     unsigned int x;
65     char s[MAX_DUZINA];

66     scanf("%u", &x);

67     prevod(x, s);

68     printf("%s\n", s);

69     return 0;
70 }

```

### Rešenje 1.17

```

#include <stdio.h>

2

4  /******
   Linearno resenje se zasniva na cinjenici:
    $x^0 = 1$   $x^k = x * x^{(k-1)}$ 
   *****/
8  int stepen(int x, int k)
9  {
10     // printf("Racunam stepen (%d, %d)\n", x, k);
    if (k == 0)
11         return 1;

12     return x * stepen(x, k - 1);
13 }

14

16  /******
   Celo telo funkcije se moze ovako kratko zapisati
   return k == 0 ? 1 : x * stepen(x,k-1);
17

18     Druga verzija prethodne funkcije. Obratiti paznju na
19     efikasnost u odnosu na prvu verziju!
20     Logaritamsko resenje je zasnovano na cinjenicama:
21
22

```

```

24     x^0 =1;
      x^k = x * (x^2 )^(k/2) , za neparno k
26     x^k = (x^2)^(k/2) , za parno k
      Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
28     doslo do rezultata, i stoga je efikasnije.
      *****/
30 int stepen2(int x, int k)
    {
32     // printf("Racunam stepen2 (%d, %d)\n",x,k);
      if (k == 0)
34         return 1;

36     /* Ako je stepen paran */
      if ((k % 2) == 0)
38         return stepen2(x * x, k / 2);
      /* Inace (ukoliko je stepen neparan) */
40     return x * stepen2(x * x, k / 2);
    }

42     /* U prethodnim funkcijama iskomentarisano je poziv funkcije printf
44     koji ispisuje odgovarajucu poruku prilikom svakog ulaska u
      funkciju. Odkomentarisati pozive printf funkcije u obe funkcije da
46     uocite razliku u broju rekurzivnih poziva obe verzije. */

48 int main()
    {
50     int x, k;
      scanf("%d%d", &x, &k);

52     printf("%d\n", stepen(x, k));
      // printf("\n-----\n");
      // printf("%d\n",stepen2(2,10));
54     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
12     paran. */

14 unsigned neparan(unsigned n);

```



```

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

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

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

```

### Rešenje 1.19

```

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

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

13  /* U sledece dve funkcije je prikazan postupak oslobadjanja od repne
    15     rekurzije koja postoji u funkciji faktoriijelRepna, koristeći
        algoritam sa predavanja.

17     Najpre, funkcija se transformise tako sto rekurzivni poziv zemeni

```

```
19     sa naredbama kojima se vrednost argumenta funkcije postavlja na
20     vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
21     goto naredbe za vracanje na pocetak tela funkcije. */

23     int faktorijelRepna_v1(int n, int result)
24     {
25     pocetak:
26         if (n == 0)
27             return result;

28         result = n * result;
29         n = n - 1;
30         goto pocetak;
31     }

32     /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
33     praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
34     iterativno resenje bez bezuslovnih skokova: */
35     int faktorijelRepna_v2(int n, int result)
36     {
37         while (n != 0) {
38             result = n * result;
39             n = n - 1;
40         }

41         return result;
42     }

43     /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
44     broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
45     ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
46     udobnosti korisnika, jer je sasvim prirodno da za faktorijel
47     zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
48     sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
49     faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
50     funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
51     poziv faktorijelRepna sa pozivom faktorijelRepna_v1, a zatim sa
52     pozivom funkcije faktorijelRepna_v2. */
53     int faktorijel(int n)
54     {
55         return faktorijelRepna(n, 1);
56     }

57     /* Test program */
58     int main()
59     {
60         int n;

61         printf("Unesite n (<= 12): ");
62         scanf("%d", &n);
63         printf("%d! = %d\n", n, faktorijel(n));
64     }
```

```

71 while(1);
73 return 0;
}

```

### Rešenje 1.21

```

#include <stdio.h>

2 int zbir_cifara(unsigned int x)
4 {
    /* Izlazak iz rekurzije: ako je broj jednocifren */
6     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()
15 {
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>
2 #define MAX_DIM 1000

4 /*****
    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 uvecenaj
    za poslednji element celog niza.
    *****/
10 int sumaNiza(int *a, int n)
11 {
12     /* Nije postavljena stroga jednakost n==0, za slucaj da
        korisnik prilikom prvog poziva, posalje negativan broj
        za velicinu niza. */
14

```

```

16     if (n <= 0)
        return 0;

18     return a[n - 1] + sumaNiza(a, n - 1);
19 }

20
21 /*****
22  Funkcija napisana na drugi nacin:
23  n==0, suma(a,0) = 0
24  n >0, suma(a,n) = a[0] + suma(a+1,n-1)
25  Suma celog niza je jednaka zbiru prvog elementa niza i sume
26  preostalih n-1 elementa.
27  *****/
28 int sumaNiza2(int *a, int n)
29 {
30     if (n <= 0)
31         return 0;
32
33     return a[0] + sumaNiza2(a + 1, n - 1);
34 }

35
36 int main()
37 {
38     int a[MAX_DIM];
39     int n, i = 0;
40
41     /* Ucitava se broj elemenata niza */
42     printf("Unesite dimenziju niza:");
43     scanf("%d", &n);
44
45     /* Ucitava se n elemenata niza. */
46     printf("Unesite elemente niza:");
47     for (i = 0; i < n; i++)
48         scanf("%d", &a[i]);
49
50     printf("Suma elemenata je %d\n", sumaNiza(a, n));
51     /* printf("Suma elemenata je %d\n",sumaNiza2(a, n)); */
52
53     return 0;
54 }

```

### Rešenje 1.23

```

1  #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)
7  {
8      /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je

```

```

    ujedno i jedini element niza */
10  if (n == 1)
    return niz[0];
12
    /* Resavanje problema manje dimenzije */
14  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
int main()
22 {
    int brojevi[MAX_DIM];
24  int n;
26
    /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
       Promenljiva i predstavlja indeks tekućeg 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

```

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

int skalarno(int a[], int b[], int n)
4  {
    /* Izlazak iz rekurzije */
6    if (n == 0)
        return 0;
8
    /* Na osnovu resenja problema dimenzije n-1, resava se problem
       dimenzije n */
10   else
        return a[n - 1] * b[n - 1] + skalarno(a, b, n - 1);
12   }
14
int main()
16 {
    int i, a[MAX_DIM], b[MAX_DIM], n;
18

```

```

20  /* Unosi se dimenzija nizova. */
    printf("Unesite dimenziju nizova:");
22  scanf("%d", &n);

24  /* A zatim i elementi nizova. */
    printf("Unesite elemente prvog niza:");
26  for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

28
    printf("Unesite elemente drugog niza:");
30  for (i = 0; i < n; i++)
        scanf("%d", &b[i]);

32
    /* Ispisuje se rezultat skalarnog proizvoda dva ucitana niza. */
34  printf("Skalarni proizvod je %d\n", skalarno(a, b, n));

36  return 0;
}

```

### Rešenje 1.25

```

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

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

        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
        printf("Unesite ceo broj:");
20        scanf("%d", &x);

22        /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz;
           Promenljiva i predstavlja indeks tekućeg broja */
24        printf("Unesite elemente niza:");
        i = 0;
26        while (scanf("%d", &a[i]) != EOF) {
            i++;
28        }
        n = i;
30

```

```

32     printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
    return 0;
}

```

### Rešenje 1.26

```

2  #include<stdio.h>
   #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     printf("Unesite tri cela broja:");
24     scanf("%d%d%d", &x, &y, &z);

26     printf("Unesite elemente niza:");
     int i = 0;
28     while (scanf("%d", &a[i]) != EOF) {
         i++;
30     }
     n = i;

32     if (tri_uzastopna_clana(x, y, z, a, n))
34         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
     else
36         printf("Uneti brojevi nisu uzastopni clanovi niza.\n");

38     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
   inacе count(x) = najvisi_bit +count(x<<1)

8

10  Za svaki naredni rekurzivan poziv prosleđuje se x<<1. Kako se
   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 x je
      postavljen na 1. Koriscenjem odgovarajuće maske proverava se
      vrednost najviseg bita. Rezultat koliko ima jedinica u ostatku
20   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
22   broja x<<1, jer se siftovanjem u levo sa desne strane dopisuju 0.
      Za rekurzivni poziv se salje vrednost koja se dobija kada se x
24   siftuje u levo. Napomena: argument funkcije x je oznacen ceo
      broj, usled cega se ne koristi siftovanje udesno, jer funkciji
26   moze biti prosleden i negativan broj. Iz tog razloga, odlucujemo
      se da proveramo najvisi, umesto najnizeg bita */
30   if (x & (1 << (sizeof(x) * 8 - 1)))
       return 1 + count(x << 1);
32   else
       return count(x << 1);

34 }

36 /*****
38  Telo prethodne funkcije je moglo biti zapisano i krace:
      jednolinijska return naredba sa proverom i rekurzivnim pozivom
40   return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1);
   *****/

42

44 int main()
46 {
   int x;
   scanf("%x", &x);
48   printf("%d\n", count(x));

50   return 0;
}

```



## Rešenje 1.29

```
1  #include<stdio.h>
2
3  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
4     broju */
5
6  int max_oktalna_cifra(unsigned x)
7  {
8      /* Izlazak iz rekurzije */
9      if (x == 0)
10         return 0;
11     /* Odredjivanje poslednje heksadekadne cifre u broju */
12     int poslednja_cifra = x & 7;
13     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
14        izbrise poslednja oktalna cifra */
15     int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
16     return poslednja_cifra >
17         max_bez_poslednje_cifre ? poslednja_cifra :
18         max_bez_poslednje_cifre;
19 }
20
21 int main()
22 {
23     unsigned x;
24     scanf("%u", &x);
25     printf("%d\n", max_oktalna_cifra(x));
26     return 0;
27 }
```

## Rešenje 1.30

```
1  #include<stdio.h>
2
3  /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
4     */
5
6  int max_heksadekadna_cifra(unsigned x)
7  {
8      /* Izlazak iz rekurzije */
9      if (x == 0)
10         return 0;
11     /* Odredjivanje poslednje heksadekadne cifre u broju */
12     int poslednja_cifra = x & 15;
13     /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
14        njega izbrise poslednja heksadekadna cifra */
15     int max_bez_poslednje_cifre = max_heksadekadna_cifra(x >> 4);
16     return poslednja_cifra >
17         max_bez_poslednje_cifre ? poslednja_cifra :
18         max_bez_poslednje_cifre;
19 }
```

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

### Rešenje 1.31

```
#include<stdio.h>
2 #include<string.h>
/* Niska moze imati najvise 32 karaktera + 1 za terminalnu nulu */
4 #define MAX_DIM 33

6 int palindrom(char s[], int n)
7 {
8     if ((n == 1) || (n == 0))
9         return 1;
10    return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
11 }

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

17
18     scanf("%s", s);

19
20     /* Odredjuje se duzina niske */
21     n = strlen(s);

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

### Rešenje 1.32

```
#include <stdio.h>
2 #include <stdlib.h>
#define MAX_DUZINA_NIZA 50
4
void ispisiNiz(int a[], int n)
```

```

6 {
8     int i;

10     for (i = 1; i <= n; i++)
11         printf("%d ", a[i]);
12     printf("\n");
13 }

14 /* Funkcija proverava da li se x vec nalazi u permutaciji na
15    prethodnih 1...n mesta */
16 int koriscen(int a[], int n, int x)
17 {
18     int i;
19     for (i = 1; i <= n; i++)
20         if (a[i] == x)
21             return 1;
22
23     return 0;
24 }

26 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[] je niz
27    u koji smesta permutacije m - oznacava da se na m-tu poziciju u
28    permutaciji smesta jedan od preostalih celih brojeva n- je
29    velicina skupa koji se permutuje Funkciju se poziva sa argumentom
30    m=1 jer formiranje permutacije pocinje od 1. pozicije. Stoga, nece
31    se koristiti a[0]. */
32 void permutacija(int a[], int m, int n)
33 {
34     int i;

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

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

54     for (i = 1; i <= n; i++) {
55         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
56            pozicije, onda se on postavlja na poziciju m i poziva se

```

```

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

70 int main(void)
71 {
72     int n;
73     int a[MAX_DUZINA_NIZA];
74
75     scanf("%d", &n);
76     if (n < 0 || n >= MAX_DUZINA_NIZA) {
77         fprintf(stderr,
78             "Duzina permutacije mora biti broj veci od 0 i manji od %
79             d!\n",
80             MAX_DUZINA_NIZA);
81         exit(EXIT_FAILURE);
82     }
83
84     permutacija(a, 1, n);
85
86     exit(EXIT_SUCCESS);
87 }

```

### Rešenje 1.33

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Rekurzivna funkcija za racunanje binomnog koeficijenta. */
5  /* ako je k=0 ili k=n, onda je binomni koeficijent 1 ako je k izmedju
6     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 /*****
16  Iterativno izracunavanje datog binomnog koeficijenta.
17
18  int binomniKoeficijent (int n, int k) {

```

```

19     int i, j, b;
    for (b=i=1, j=n; i<=k; b =b * j-- / i++)
21         ;
    return b;
23 }
    *****/
25
    /* Prostim opažanjem se uocava da se svaki element n-te hipotenuze
27     (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1 hipotenuze. Uz
    pomenute dve nove ivicne jedinice lako se zakljucuje da ce suma
29     elementa n-te hipotenuze biti tacno 2 puta veca. */
    int sumaElemenataHipotenuze(int n)
31 {
    return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
33 }

35 int main()
    {
37     int n, k, i, d, r;

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

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

59     exit(EXIT_SUCCESS);
    }

```

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

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

### 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: neodgovarajuća dimenzija niza.
```

### Primer 4

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 101
Greska: neodgovarajuća 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 koje od ova dva rešenja treba koristiti prilikom ispisa.

### Primer 1

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

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

### Primer 2

```
Poziv: ./a.out

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

[Rešenje 2.4]

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

### Primer 1

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

INTERAKCIJA 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 završavaju na ke

INTERAKCIJA PROGRAMA:
Broj reci koje se završavaju 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 dimenziju kvadratnih matrica  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrica. Na standardni izlaz ispisati da li su matrice jednake, a zatim ispisati zbir i proizvod učitanih matrica.

*Primer 1*

```
INTERAKCIJA 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 sporedne dijagonale je 7.
Indeks kolone sa najmanjim elementom je 2.
Indeks vrste sa najvećim elementom je 2.
Broj negativnih elemenata matrice je 3.

```

*Primer 2*

```

Poziv: ./a.out 4

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

```

[Rešenje 2.11]

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

*Primer 1*

```

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

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

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

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

### Primer 1

```
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:
```

```
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 elemenata 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 sličica.
Reprezentacija za koju je sakupio
najmanji broj sličica 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.
```

### Primer 2

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

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

## 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 10000 1.570795
|| Limes funkcije tan je -10134.46.
```

Primer 2

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

**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.
```

Primer 1

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

**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.

```

*Primer 2*

```

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

```

## 2.5 Rešenja

### Rešenje 2.1

```

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

```

```

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

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

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

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

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

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

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

67
68         /* Proverava se da li je doslo do prekoračenja ograničenja
69         dimenzije */
70         if (n <= 0 || n > MAX) {
71             fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72             exit(EXIT_FAILURE);
73         }

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

78
79         obrni_niz_v1(a, n);

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

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

86
87         obrni_niz_v2(a, n);

```

```
89     printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
91     for (p = a; p - a < n; p++)
92         printf("%d ", *p);
93     printf("\n");
95     return 0;
96 }
```

## Rešenje 2.2

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 100
5
6  /* Funkcija izracunava zbir elemenata niza */
7  double zbir(double *a, int n)
8  {
9      double s = 0;
10     int i;
11
12     for (i = 0; i < n; s += *(a + i++));
13
14     return s;
15 }
16
17 /* Funkcija izracunava proizvod elemenata niza */
18 double proizvod(double *a, int n)
19 {
20     double p = 1;
21
22     for (; n; n--)
23         p *= *(a + n - 1);
24
25     return p;
26 }
27
28 /* Funkcija izracunava minimalni element niza */
29 double min(double *a, int n)
30 {
31     /* Na pocetku, minimalni element je prvi element */
32     double min = *a;
33     int i;
34
35     /* Ispituje se da li se medju ostalim elementima niza nalazi
36        minimalni */
37     for (i = 1; i < n; i++)
38         if (*(a + i) < min)
39             min = *(a + i);
40 }
```

```
40     return min;
42 }

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

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

56     return max;
58 }

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

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

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

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

78     /* Vrsi se testiranje definisanih funkcija */
80     printf("Zbir elemenata niza je %.3f.\n", zbir(a, n));
    printf("Proizvod elemenata niza je %.3f.\n", proizvod(a, n));
82     printf("Minimalni element niza je %.3f.\n", min(a, n));
    printf("Maksimalni element niza je %.3f.\n", max(a, n));

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

```

printf("Unesite elemente niza:\n");
53 for (p = a; p - a < n; p++)
    scanf("%d", p);
55
povecaj_smanji(a, n);
57
printf("Transformisan niz je:\n");
59 for (p = a; p - a < n; p++)
    printf("%d ", *p);
61 printf("\n");
63
return 0;
}

```

## Rešenje 2.4

```

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

```

```

    argc = i;
35 }

37 printf("Pocetna slova argumenata komandne linije:\n");
    if (tip_ispisa == 'I') {
39         /* koristeći indeksnu sintaksu */
        for (i = 0; i < argc; i++)
41             printf("%c ", argv[i][0]);
        printf("\n");
43     } else if (tip_ispisa == 'P') {
        /* koristeći pokazivačku sintaksu */
45         for (i = 0; i < argc; i++)
            printf("%c ", **argv++);
47         printf("\n");
    }
49
    return 0;
51 }

```

## Rešenje 2.5

```

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

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

15 int main(int argc, char **argv)
    {
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
25     ("Broj argumenata komandne linije koji su palindromi je %d.\n",
        n);
27     return 0;
    }

```

## Rešenje 2.6

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

```
50     return 0;
    }
```

## Rešenje 2.7

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Implementacija funkcije strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
8  {
9      int i;
10     for (i = 0; *(src + i); i++)
11         *(dest + i) = *(src + i);
12 }
13
14 /* Implementacija funkcije strcmp() iz standardne biblioteke */
15 int poredjenje_niski(char *s, char *t)
16 {
17     int i;
18     for (i = 0; *(s + i) == *(t + i); i++)
19         if (*(s + i) == '\0')
20             return 0;
21     return *(s + i) - *(t + i);
22 }
23
24 /* Implementacija funkcije strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
26 {
27     int i;
28     for (i = 0; *(s + i); i++);
29     return i;
30 }
31
32 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
33    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 funkcije
44    prefiks niske zadate prvi argumentom funkcije */
45 int prefiks_niske(char *niska, char *prefiks)
46 {
```

```
int i;
48 if (duzina_niske(prefiks) <= duzina_niske(niska)) {
    for (i = 0; i < duzina_niske(prefiks); i++)
50         if (*(prefiks + i) != *(niska + i))
                return 0;
52     return 1;
} else
54     return 0;
}

56 int main(int argc, char **argv)
58 {
    /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
60     greska */
    if (argc < 4) {
62         printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
64         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
                argv[0]);
        exit(EXIT_FAILURE);
66     }

68     FILE *in;
70     int br = 0;
    char rec[MAX_KARAKTERA];

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

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

94     fclose(in);
    return 0;
96 }
```

## Rešenje 2.8

```

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

5  #define MAX 100

7  /* Deklaracija funkcija koje ce kasnije biti definisane */
   double euklidska_norma(int M[][MAX], int n);
9  int trag(int M[][MAX], int n);
   int gornja_vandijagonalna_norma(int M[][MAX], int n);
11
   int main()
13 {
   int A[MAX][MAX];
15 int i, j, n;

17 printf("Unesite dimenziju matrice: ");
   scanf("%d", &n);
19
   /* Provera prekoracenja dimenzije matrice */
21 if (n > MAX || n <= 0) {
   fprintf(stderr, "Greska: neodgovarajuca dimenzija matrice.\n");
23 exit(EXIT_FAILURE);
   }

25
   printf("Unesite elemente matrice, vrstu po vrstu:\n ");
27 for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
29     scanf("%d", &A[i][j]);

31 /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
   for (i = 0; i < n; i++) {
33     /* Ispis elemenata i-te vrste */
   for (j = 0; j < n; j++)
35     printf("%d ", A[i][j]);
   printf("\n");
37 }

39 /******
   Ispisuju se elemenati matrice koriscenjem pokazivacke sintakse.
41 Kod ovako definisane matrice, elementi su uzastopno smesteni u
   memoriju, kao na traci. To znaci da su svi elementi prve vrste
43 redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
   prve vrste smesten je prvi element druge vrste za kojim slede
45 svi elementi te vrste i tako dalje redom.

47 for( i = 0; i < n ; i++) {
   for ( j=0 ; j<n ; j++)
49     printf("%d ", (*(A+i)+j));
   printf("\n");

```

```
51     }
52     *****/
53
54     /* Ispisuje se rezultat na standardni izlaz */
55     int tr = trag(A, n);
56     printf("Trag matrice je %d.\n", tr);
57
58     printf("Euklidska norma matrice je %.2f.\n", euklidska_norma(A, n))
59     ;
60     printf("Vandijagonalna norma matrice je = %d.\n",
61           gornja_vandijagonalna_norma(A, n));
62
63     return 0;
64 }
65
66 /* Definicija funkcija koje su ranije bile deklarirane */
67
68 /* Funkcija izracunava trag matrice */
69 int trag(int M[][MAX], int n)
70 {
71     int trag = 0, i;
72     for (i = 0; i < n; i++)
73         trag += M[i][i];
74     return trag;
75 }
76
77 /* Funkcija izracunava euklidsku normu matrice */
78 double euklidska_norma(int M[][MAX], int n)
79 {
80     double norma = 0.0;
81     int i, j;
82
83     for (i = 0; i < n; i++)
84         for (j = 0; j < n; j++)
85             norma += M[i][j] * M[i][j];
86
87     return sqrt(norma);
88 }
89
90 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
91 int gornja_vandijagonalna_norma(int M[][MAX], int n)
92 {
93     int norma = 0;
94     int i, j;
95
96     for (i = 0; i < n; i++) {
97         for (j = i + 1; j < n; j++)
98             norma += abs(M[i][j]);
99     }
100
101     return norma;
102 }
```



## Rešenje 2.9

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 100
5
   /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
   standardnog ulaza */
7  void ucitaj_matricu(int m[][MAX], int n)
9  {
   int i, j;
11
   for (i = 0; i < n; i++)
13     for (j = 0; j < n; j++)
        scanf("%d", &m[i][j]);
15 }

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

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

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

41     /* Prosla je provera jednakosti za sve parove elemenata koji su na
       istim pozicijama. To znaci da su matrice jednake */
43     return 1;
   }

45
47 /* Funkcija izracunava zbir dve kvadratne matrice */
   void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
   {
49     int i, j;
```

```

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

```

```

103     else
104         printf("Matrice nisu jednake.\n");
105
106         printf("Zbir matrica je:\n");
107         ispisi_matricu(zbir, n);
108
109         printf("Proizvod matrica je:\n");
110         ispisi_matricu(proizvod, n);
111
112     return 0;
113 }

```

### Rešenje 2.10

```

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

#define MAX 64

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

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

    return 1;
}

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

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

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

```

```
38 /* Funkcija proverava da li je relacija simetricna. Relacija je
40    simetricna ako za svaki par elemenata vazi: ako je element "i" u
    relaciji sa elementom "j", onda je i element "j" u relaciji sa
    elementom "i". Ovakve matrice su simetricne u odnosu na glavnu
42    dijagonalu */
int simetricnost(int m[][MAX], int n)
44 {
    int i, j;
46
    /* Obilaze se elementi ispod glavne dijagonale matrice i upoređuju
    se sa njima simetricnim elementima */
48    for (i = 0; i < n; i++)
        for (j = 0; j < i; j++)
50            if (m[i][j] != m[j][i])
52                return 0;
54    return 1;
56 }
58 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono je
    odredjeno matricom koja sadrzi sve elemente polazne matrice
    dopunjene tako da matrica postane simetricna u odnosu na glavnu
60    dijagonalu */
void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
62 {
    int i, j;
64
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
66            zatvorenje[i][j] = m[i][j];
68
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
70            if (zatvorenje[i][j] == 1)
72                zatvorenje[j][i] = 1;
74 }
76 /* Funkcija proverava da li je relacija tranzitivna. Relacija je
    tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
    relaciji sa elementom "j" i element "j" u relaciji sa elementom
    "k", onda je i element "i" u relaciji sa elementom "k" */
80 int tranzitivnost(int m[][MAX], int n)
    {
82         int i, j, k;
84
86         for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                /* Ispituje se da li postoji element koji narusava *
                tranzitivnost */
88                 for (k = 0; k < n; k++)
                    if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
```

```
90         return 0;
92     return 1;
93 }
94
95 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
96    relacije koriscenjem Varsalovog algoritma */
97 void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
98 {
99     int i, j, k;
100
101     /* Prepisuju se vrednosti elemenata pocetne matrice */
102     for (i = 0; i < n; i++)
103         for (j = 0; j < n; j++)
104             zatvorenje[i][j] = m[i][j];
105
106     /* Odredjuje se reflektivno zatvorenje matrice */
107     for (i = 0; i < n; i++)
108         zatvorenje[i][i] = 1;
109
110     /* Primenom Varsalovog algoritma odredjuje se tranzitivno
111        zatvorenje matrice */
112     for (k = 0; k < n; k++)
113         for (i = 0; i < n; i++)
114             for (j = 0; j < n; j++)
115                 if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
116                     && (zatvorenje[i][j] == 0))
117                     zatvorenje[i][j] = 1;
118 }
119
120 /* Funkcija ispisuje elemente matrice */
121 void pisi_matricu(int m[][MAX], int n)
122 {
123     int i, j;
124
125     for (i = 0; i < n; i++) {
126         for (j = 0; j < n; j++)
127             printf("%d ", m[i][j]);
128         printf("\n");
129     }
130 }
131
132 int main(int argc, char *argv[])
133 {
134     FILE *ulaz;
135     int m[MAX][MAX];
136     int pomocna[MAX][MAX];
137     int n, i, j;
138
139     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
140        */
```

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

```

194  /* Zatvara se datoteka */
    fclose(ulaz);
196  return 0;
    }

```

### Rešenje 2.11

```

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

4  #define MAX 32

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

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

18    return max_na_sporednoj_dijagonali;
    }

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

26     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
28         if (m[i][j] < min) {
            min = m[i][j];
            indeks_kolone = j;
30         }
32     }

34     return indeks_kolone;
    }

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

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

```

```

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

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

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

61     return broj_negativnih;
62 }

64 }

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

71     /* Proverava se broj argumenata komandne linije */
72     if (argc < 2) {
73         printf("Greska: ");
74         printf("Nedovoljan broj argumenata komandne linije.\n");
75         printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
76         exit(EXIT_FAILURE);
77     }

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

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

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

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

```



```

96         max_sporedna_dijagonala(m, n));
98     printf("Indeks kolone sa najmanjim elementom je %d.\n",
           indeks_min(m, n));
100
102     printf("Indeks vrste sa najvećim elementom je %d.\n",
           indeks_max(m, n));
104
106     printf("Broj negativnih elemenata matrice je %d.\n",
           broj_negativnih(m, n));
108
106     return 0;
108 }

```

### Rešenje 2.12

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 32
5
   /* Funkcija ucitava elemente kvadratne matrice sa standardnog ulaza
      */
7  void ucitaj_matricu(int m[][MAX], int n)
   {
9      int i, j;
11
13     for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
             scanf("%d", &m[i][j]);
15 }
17
   /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
      */
17 void ispis_matricu(int m[][MAX], int n)
   {
19     int i, j;
21
23     for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++)
             printf("%d ", m[i][j]);
         printf("\n");
25     }
27 }
29
   /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
      da li je normirana i ortogonalna. Matrica je normirana ako je
      proizvod svake vrste matrice sa samom sobom jednak jedinici.
      Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
      vrste matrice jednak nuli */
31
33 int ortonormirana(int m[][MAX], int n)

```

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

```

    return 0;
87 }

```

### Rešenje 2.13

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX_V 10
5  #define MAX_K 10
7
   /* Funkcija proverava da li su ispisani svi elementi iz matrice,
      odnosno da li se narušio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
   {
11     return !(top <= bottom && left <= right);
   }
13
   /* Funkcija spiralno ispisuje elemente matrice */
15 void ispis_i_matricu_spiralno(int a[][MAX_K], int n, int m)
   {
17     int i, j, top, bottom, left, right;
19     top = left = 0;
      bottom = n - 1;
21     right = m - 1;
23     while (!krajIspisa(top, bottom, left, right)) {
25         for (j = left; j <= right; j++)
            printf("%d ", a[top][j]);
27
           /* Spusta se prvi red */
29         top++;
31         if (krajIspisa(top, bottom, left, right))
            break;
33
           for (i = top; i <= bottom; i++)
35             printf("%d ", a[i][right]);
37
           /* Pomera se desna kolona za naredni krug ispisa blize levom
              kraju */
39         right--;
41         if (krajIspisa(top, bottom, left, right))
            break;
43
           /* Ispisuje se donja vrsta */
45         for (j = right; j >= left; j--)
            printf("%d ", a[bottom][j]);

```

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

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

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

57      /* Priprema se leva kolona za naredni krug ispisa */
59      left++;
60  }
61  putchar('\n');
62  }

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

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

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

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

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

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

89      printf("Spiralno ispisana matrica: ");
90      ispisi_matricu_spiralno(a, n, m);

93      return 0;
94  }
```

## Rešenje 2.15

```

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

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

7

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

9

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

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

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

27

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

    return 0;
31 }

```

### Rešenje 2.16

```

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

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

8

    /* Velicina alocirane memorije */
    int alocirano;

10

    /* Broj elemenata niza */
    int n;
14

```

```

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

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

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

26
    while (x != 0) {
28        if (n == alocirano) {
            alocirano = alocirano + KORAK;

30
            /* Vrsi se realokacija memorije sa novom velicinom */
32            /* Resenje sa funkcijom malloc() */
            b = (int *) malloc(alocirano * sizeof(int));

34
            if (b == NULL) {
36                fprintf(stderr, "malloc(): ");
                fprintf(stderr, "greska pri alokaciji memorije.\n");
38                free(a);
                exit(EXIT_FAILURE);
40            }

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

46
            free(a);

48
            /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
                bloka koji je prilikom alokacije zapamcen u promenljivoj b
50            */
            a = b;

52
            /******
54            Resenje sa funkcijom realloc()

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

58
            a = (int*) realloc(a,alocirano*sizeof(int));
            if(a == NULL) {
60                fprintf(stderr, "realloc(): ");
                fprintf(stderr, "greska pri alokaciji memorije.\n");
62                exit(EXIT_FAILURE);
64            }
            *****/
66    }

```

```
68     a[n++] = x;
70     scanf("%d", &x);
72 }
74 printf("Niz u obrnutom poretku je: ");
76 for (n--; n >= 0; n--)
78     printf("%d ", a[n]);
80 printf("\n");
82 /* Oslobadja se dinamicki alocirana memorija */
84 free(a);
86 exit(EXIT_SUCCESS);
88 }
```

### Rešenje 2.17

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

```
32     printf("Unesite dve niske karaktera:\n");
34     scanf("%s", s1);
36     scanf("%s", s2);

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

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

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

44     return 0;
45 }
```

### Rešenje 2.18

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

5  int main()
6  {
7      int i, j;

9      /* Pokazivac na dinamicki alociran niz pokazivaca na vrste matrice
10       */
11     double **A = NULL;

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

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

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

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

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

29     /* Dinamicki se alocira prostor za elemente u vrstama */
30
31     /* ... */
```



```

33     for (i = 0; i < n; i++) {
        A[i] = malloc(sizeof(double) * m);

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

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

51     /* Izracunava se trag matrice, odnosno suma elemenata na glavnoj
        dijagonali */
53     trag = 0.0;

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

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

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

63     /* Oslobadja se memorija za niz pokazivaca na vrste */
65     free(A);

67     return 0;
}

```

### Rešenje 2.19

```

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

5  /* Funkcija ucitava matricu sa ulaza */
void ucitaj_matricu(int **M, int n, int m)
7  {
    int i, j;

9

    for (i = 0; i < n; i++)
11         for (j = 0; j < m; j++)

```

```
13     scanf("%d", &M[i][j]);
15 void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
16 {
17     int i, j;
19     for (i = 0; i < n; i++) {
20         for (j = 0; j <= i; j++)
21             printf("%d ", M[i][j]);
22         printf("\n");
23     }
24 }
25
26 int main()
27 {
28     int m, n, i, j;
29     int **matrica = NULL;
31     printf("Unesite broj vrsta i broj kolona matrice:\n ");
32     scanf("%d %d", &n, &m);
33
34     /* Alocira se prostor za niz pokazivaca na vrste matrice */
35     matrica = (int **) malloc(n * sizeof(int *));
36     if (matrica == NULL) {
37         fprintf(stderr, "malloc(): Neuspela alokacija\n");
38         exit(EXIT_FAILURE);
39     }
41     /* Alocira se prostor za svaku vrstu matrice */
42     for (i = 0; i < n; i++) {
43         matrica[i] = (int *) malloc(m * sizeof(int));
45         if (matrica[i] == NULL) {
46             fprintf(stderr, "malloc(): Neuspela alokacija\n");
47             for (j = 0; j < i; j++)
48                 free(matrica[j]);
49             free(matrica);
50             exit(EXIT_FAILURE);
51         }
52     }
53
54     printf("Unesite elemente matrice, vrstu po vrstu:\n");
55     ucitaj_matricu(matrica, n, m);
57     printf("Elementi ispod glavne dijagonale matrice:\n");
58     ispisi_elemente_ispod_dijagonale(matrica, n, m);
59
60     /* Oslobadja se dinamicki alocirana memorija za matricu. Prvo se
61        oslobadja memorija rezervisana za svaku vrstu */
62     for (j = 0; j < n; j++)
63         free(matrica[j]);
```

```
65  /* Zatim se oslobadja memorija za niz pokazivaca na vrste matrice
    */
    free(matrica);
67
    return 0;
69 }
```

### Rešenje 2.21

```

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

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

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

/* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
sporedne dijagonale. Element se nalazi ispod sporedne dijagonale
ukoliko je zbir indeksa vrste i indeksa kolone elementa veci od
n-1 */
float zbir_ispod_sporedne_dijagonale(float **m, int n)
{
    int i, j;
    float zbir = 0;

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

    return zbir;
}

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

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

```
42     for (j = 0; j < n; j++)
43         fscanf(ulaz, "%f", &m[i][j]);
44 }

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

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

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

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

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

73         if (m[i] == NULL) {
74             printf("malloc(): neuspela alokacija memorije!\n");
75             for (j = 0; j < i; j++)
76                 free(m[j]);
77             free(m);
78             exit(EXIT_FAILURE);
79         }
80     }
81     return m;
82 }

84 /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
85    n */
86 void oslobodi_memoriju(float **m, int n)
87 {
88     int i;

90     for (i = 0; i < n; i++)
91         free(m[i]);
92     free(m);
93 }
```

```
94 }
96 int main(int argc, char *argv[])
97 {
98     FILE *ulaz;
99     float **a;
100     int n;
101
102     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska */
103     if (argc < 2) {
104         printf("Greska: ");
105         printf("Nedovoljan broj argumenata komandne linije.\n");
106         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
107         exit(EXIT_FAILURE);
108     }
109
110     /* Otvara se datoteka za citanje */
111     ulaz = fopen(argv[1], "r");
112     if (ulaz == NULL) {
113         fprintf(stderr, "Greska: ");
114         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
115         exit(EXIT_FAILURE);
116     }
117
118     /* Cita se dimenzija matrice */
119     fscanf(ulaz, "%d", &n);
120
121     /* Alocira se memorija */
122     a = alociraj_memoriju(n);
123
124     /* Ucitavaju se elementi matrice */
125     ucitaj_matricu(ulaz, a, n);
126
127     float zbir = zbir_ispod_sporedne_dijagonale(a, n);
128
129     /* Poziva se funkcija za transformaciju matrice */
130     izmeni(a, n);
131
132     /* Ispisuje se rezultat */
133     printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
134     printf("je %.2f.\n", zbir);
135
136     printf("Transformisana matrica je:\n");
137     ispisi_matricu(a, n);
138
139     /* Oslobadja se memorija */
140     oslobodi_memoriju(a, n);
141
142     /* Zatvara se datoteka */
143     fclose(ulaz);
144 }
```

```
    return 0;
146 }
```

### Rešenje 2.26

```
1  #include <stdio.h>
3  #include <stdlib.h>
4  #include <math.h>
5  #include <string.h>
6
7  /* Funkcija tabela() prihvata granice intervala a i b, broj
8     ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
9     funkciju koja prihvata double argument, i vraca double vrednost.
10     Za tako datu funkciju ispisuju se njene vrednosti u intervalu
11     [a,b] u n ekvidistantnih tacaka intervala */
12 void tabela(double a, double b, int n, double (*fp) (double))
13 {
14     int i;
15     double x;
16
17     printf("-----\n");
18     for (i = 0; i < n; i++) {
19         x = a + i * (b - a) / (n - 1);
20         printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
21     }
22     printf("-----\n");
23 }
24
25 double sqr(double a)
26 {
27     return a * a;
28 }
29
30 int main(int argc, char *argv[])
31 {
32     double a, b;
33     int n;
34
35     char ime_fje[6];
36
37     /* Pokazivac na funkciju koja ima jedan argument tipa double i
38        povratnu vrednost istog tipa */
39     double (*fp) (double);
40
41     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
42        */
43     if (argc < 2) {
44         printf("Greska: ");
45         printf("Nedovoljan broj argumenata komandne linije.\n");
46         printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
```

```
47     argv[0]);
48     exit(EXIT_FAILURE);
49 }
50
51 /* Niska ime_fje sadrzi ime trazene funkcije koja je navedena u
52    komandnoj liniji */
53 strcpy(ime_fje, argv[1]);
54
55 /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
56    */
57 if (strcmp(ime_fje, "sin") == 0)
58     fp = &sin;
59 else if (strcmp(ime_fje, "cos") == 0)
60     fp = &cos;
61 else if (strcmp(ime_fje, "tan") == 0)
62     fp = &tan;
63 else if (strcmp(ime_fje, "atan") == 0)
64     fp = &atan;
65 else if (strcmp(ime_fje, "acos") == 0)
66     fp = &acos;
67 else if (strcmp(ime_fje, "asin") == 0)
68     fp = &asin;
69 else if (strcmp(ime_fje, "exp") == 0)
70     fp = &exp;
71 else if (strcmp(ime_fje, "log") == 0)
72     fp = &log;
73 else if (strcmp(ime_fje, "log10") == 0)
74     fp = &log10;
75 else if (strcmp(ime_fje, "sqrt") == 0)
76     fp = &sqrt;
77 else if (strcmp(ime_fje, "floor") == 0)
78     fp = &floor;
79 else if (strcmp(ime_fje, "ceil") == 0)
80     fp = &ceil;
81 else if (strcmp(ime_fje, "sqr") == 0)
82     fp = &sqr;
83 else {
84     printf("Program jos uvek ne podrzava trazenu funkciju!\n");
85     exit(EXIT_SUCCESS);
86 }
87
88 printf("Unesite krajeve intervala:\n");
89 scanf("%lf %lf", &a, &b);
90
91 printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
92 printf("(ukljucujuci krajeve intervala)?\n");
93 scanf("%d", &n);
94
95 /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
96    uneti broj veci od 2 */
97 if (n < 2) {
98     fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
```

```

    exit(EXIT_FAILURE);
99 }

101 /* Ispisuje se ime funkcije */
    printf("      x %10s(x)\n", ime_fje);
103
105 /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
    komandne linije */
    tabela(a, b, n, fp);
107
    exit(EXIT_SUCCESS);
109 }
```



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

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

*Test 1*

<pre> Poziv: ./a.out 1000000 23542 IZLAZ: Linearna pretraga: Element nije u nizu Binarna pretraga: Element nije u nizu Interpolaciona pretraga: Element nije u nizu </pre>	<pre> VREMENA.TXT Dimenzija niza: 1000000 Linearna: 3615091 ns Binarna: 1536 ns Interpolaciona: 558 ns </pre>
--	---

*Test 2*

<pre> Poziv: ./a.out 100000 37842 IZLAZ: Linearna pretraga: Element nije u nizu Binarna pretraga: Element nije u nizu Interpolaciona pretraga: Element nije u nizu </pre>	<pre> VREMENA.TXT Dimenzija niza: 1000000 Linearna: 3615091 ns Binarna: 1536 ns Interpolaciona: 558 ns  Dimenzija niza: 100000 Linearna: 360803 ns Binarna: 1187 ns Interpolaciona: 628 ns </pre>
---	---

[Rešenje 3.1]

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

*Primer 1*

```

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

```

*Primer 2*

```

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

```

[Rešenje 3.2]

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće.

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

### Primer 1

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

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

INTERAKCIJA PROGRAMA:

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

### Primer 2

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

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

INTERAKCIJA PROGRAMA:

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

[Rešenje 3.3]

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

[Rešenje 3.4]

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

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

[Rešenje 3.5]

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

*Test 1*

```
Izlaz:
1.57031
```

[Rešenje 3.6]

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

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

[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.

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

[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>
<pre> ULAZ:  4 IZLAZ:  2 2 </pre>	<pre> ULAZ:  17 IZLAZ:  4 4 </pre>	<pre> ULAZ:  1031 IZLAZ:  10 10 </pre>

[Rešenje 3.9]

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

i same koordinate temena duži. UPUTSTVO: *Vršiti binarnu pretragu intervala  $[0, 90^\circ]$ .*

*Primer 1*

```
INTERAKCIJA PROGRAMA:
Unesi broj tacaka: 2
Unesi koordinate tacaka:
2 0 2 1
1 2 2 2
26.57
```

*Primer 2*

```
INTERAKCIJA PROGRAMA:
Unesi broj tacaka: 2
Unesi koordinate tacaka:
1 0 1 1
0 1 1 1
45
```

*Primer 3*

```
INTERAKCIJA PROGRAMA:
Unesi broj tacaka: 3
Unesi koordinate tacaka:
1 0 1 1
2 0 2 1
1 2 2 2
26.57
```

## 3.2 Sortiranje

**Zadatak 3.11** 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*

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

*Test 2*

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

*Test 3*

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

[Rešenje 3.11]

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

*Primer 1*

```
INTERAKCIJA PROGRAMA:
Unesite prvu nisku anagram
Unesite drugu nisku ramgana
jesu
```

*Primer 2*

```
INTERAKCIJA PROGRAMA:
Unesite prvu nisku anagram
Unesite drugu nisku anagrm
nisu
```

*Primer 3*

```
INTERAKCIJA PROGRAMA:
Unesite prvu nisku test
Unesite drugu nisku tset
jesu
```

[Rešenje 3.12]

**Zadatak 3.13** Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza i neće biti duži od 256 i kraći od jednog elemenata. UPUTSTVO: *Prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<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 3.13]

**Zadatak 3.14** 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.*

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 3</i>
<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 3.14]

**Zadatak 3.15** Napraviti biblioteku 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. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za **merge**, **-q** za **quick**, **-b** za **bubble**, **-i** za **insertion** ili **-s** za **shell sort**. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati **selection sort** algoritmom. Niz koji se sortira generisati neopadajuće ako je prisutna opcija **-r**, nerastuće ako je prisutna opcija **-o** ili potpuno slučajno ako nema nijedne opcije. Vreme meriti programom **time**. Analizirati porast vremena sa porastom dimenzije **n**.

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

[Rešenje 3.15]

**Zadatak 3.16** Napisati funkciju potpisa `int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3, int dim3)` koja prima dva sortirana niza, i na osnovu njih pravi novi sortirani niz koji koji sadrži elemente oba niza. Treća dimenzija predstavlja veličinu niza u koji se smešta rezultat. Ako je ona manja od potrebne dužine, funkcija vraća -1 kao indikator neuspeha, inače vraća 0. Napisati 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.

<p><i>Primer 1</i></p> <pre> 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 </pre>	<p><i>Primer 2</i></p> <pre> 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 </pre>
--	--

[Rešenje 3.16]

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



*Test 1*

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

```
PRVI-DEO.TXT
```

```
Andrija Petrovic
Anja Ilic
Ivana Markovic
Lazar Micic
Nenad Brankovic
Sofija Filipovic
Vladimir Savic
Uros Milic
```

```
DRUGI-DEO.TXT
```

```
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Marija Stankovic
Ognjen Peric
```

```
CEO-TOK.TXT
```

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

[Rešenje 3.17]

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

*Test 1*

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

```
IN.TXT
```

```
3 4
11 6
7 3
2 82
-1 6
```

```
OUT.TXT
```

```
-1 6
2 82
3 4
7 3
11 6
```

*Test 2*

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

```
IN.TXT
```

```
3 4
11 6
7 3
2 82
-1 6
```

```
OUT.TXT
```

```
3 4
-1 6
7 3
11 6
2 82
```

[Rešenje 3.18]

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

*Test 1*

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

```
IZLAZ:
3
```

*Test 2*

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

```
IZLAZ:
1
```

*Test 3*

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

```
IZLAZ:
Problem pri otvaranju
datoteke.
```

[Rešenje 3.19]

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

*Test 1*

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

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

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

*Test 2*

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

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

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

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

*Test 1*

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

[Rešenje 3.21]

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

**Zadatak 3.23** Napisati program koji iz datoteke `aktivnost.txt` čita podatke o aktivnostima studenata na praktikumima i u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt` upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po

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

### Test 1

AKTIVNOSTI.TXT

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

DAT1.TXT

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

DAT2.TXT

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

DAT3.TXT

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

[Rešenje 3.23]

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

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

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

- nema opcija, sortiranje se vrši po broju gledanja;

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

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

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

[Rešenje 3.24]

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

#### Primer 1

```
INTERAKCIJA 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 3.26** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze

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

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

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

#### Test 1

```

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

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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.27** Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime, a zatim se učitava jedan karakter i pronalazi i štampa jedna struktura iz niza osoba čije prezime počinje tim karakterom. Ako takva osoba ne postoji, štampati  $-1$  na standardnom izlazu. Niz struktura ima manje od 100 elemenata i uređen je u rastućem leksikografskom poretku po prezimenima. Pretraživanje niza vršiti bibliotečkom funkcijom `bsearch`. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

```

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: R  IZLAZ: Dusko Radovic </pre>	<pre> ULAZ: E  IZLAZ: Dobrica Eric </pre>	<pre> ULAZ: X  IZLAZ: -1 </pre>

**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.

<i>Primer 1</i>	<i>Primer 2</i>
<pre> 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 </pre>	<pre> 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! </pre>

[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.

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 3</i>
<pre> 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 </pre>	<pre> INTERAKCIJA PROGRAMA: Uneti dimenziju niza: 1 Uneti elemente niza: 234 Sortirani niz u rastucem poretku prema broju delilaca 234 </pre>	<pre> INTERAKCIJA PROGRAMA: Uneti dimenziju niza: 0 Uneti elemente niza: Sortirani niz u rastucem poretku prema broju delilaca: </pre>

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

#### Primer 1

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

INTERAKCIJA 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
Niske sortirane po duzini:
ana matej milos petar milica nikola andjela ljubica aleksandar
Niska "matej"je pronadjena u nizu na poziciji 1
```

[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 3.12, 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. `mr15125`, `mm14001`), ime, prezime i broj poena. Ni ime, ni prezime neće biti duže od 20 karaktera. Napisati

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

#### Test 1

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

#### Primer 1

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

*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
7  -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
10 njemu element x. Pretraga se vrši prostom iteracijom kroz niz. Ako
11 se element pronadje funkcija vraca indeks pozicije na kojoj je
12 pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
13 pronadjen u nizu, funkcija vraca -1, kao indikator neuspesne
14 pretrage. */
15 int linearna_pretraga(int a[], int n, int x)
16 {
17     int i;
18     for (i = 0; i < n; i++)
19         if (a[i] == x)
20             return i;
21     return -1;
22 }
23
24 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
25 pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
26 int binarna_pretraga(int a[], int n, int x)
27 {
28     int levi = 0;
29     int desni = n - 1;
30     int srednji;
31     /* Dokle god je indeks levi levo od indeksa desni */
32     while (levi <= desni) {
33         /* Srednji indeks je njihova aritmeticka sredina */
34         srednji = (levi + desni) / 2;
35         /* Ako je element sa sredisnjim indeksom veci od x, tada se x
36 mora nalaziti u levoj polovini niza */
37         if (x < a[srednji])
38             desni = srednji - 1;
39         /* Ako je element sa sredisnjim indeksom manji od x, tada se x
40 mora nalaziti u desnoj polovini niza */
41         else if (x > a[srednji])
42             levi = srednji + 1;
43         else
44             /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
45 x pronadjen na poziciji srednji */
46             return srednji;
47     }
48     /* Ako element x nije pronadjen, vraca se -1 */
49     return -1;
50 }
51
```

```
/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
53 pozicije nadjenog elementa ili -1, ako element nije pronadjen */
int interpolaciona_pretraga(int a[], int n, int x)
55 {
    int levi = 0;
57     int desni = n - 1;
    int srednji;
59     /* Dokle god je indeks levi levo od indeksa desni... */
    while (levi <= desni) {
61         /* Ako je trazeni element manji od pocetnog ili veci od
           poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
63 nije u tom delu niza. Ova provera je neophodna, da se ne bi
           dogodilo da se prilikom izracunavanja indeksa srednji izadje
65 izvan opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
67             return -1;
        /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
69 a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
           jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
71 desni). Ova provera je neophodna, jer bi se u suprotnom
           prilikom izracunavanja indeksa srednji pojavilo deljenje
73 nulom. */
        else if (a[levi] == a[desni])
75             return levi;
        /* Racunanje srednjeg indeksa */
77         srednji =
            levi +
79             ((double) (x - a[levi]) / (a[desni] - a[levi])) *
            (desni - levi);
81         /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
           verovatno biti blize trazenoj vrednosti nego da je prosto uvek
83 uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
           porediti sa pretragom recnika: ako neko trazi rec na slovo 'B
           ',
85 sigurno nece da otvori recnik na polovini, vec verovatno negde
           blize pocetku. */
87         /* Ako je element sa indeksom srednji veci od trazenog, tada se
           trazeni element mora nalaziti u levoj polovini niza */
89         if (x < a[srednji])
            desni = srednji - 1;
91         /* Ako je element sa indeksom srednji manji od trazenog, tada se
           trazeni element mora nalaziti u desnoj polovini niza */
93         else if (x > a[srednji])
            levi = srednji + 1;
95         else
            /* Ako je element sa indeksom srednji jednak trazenom, onda se
97 pretraga zavrшава na poziciji srednji */
            return srednji;
99     }
    /* U slucaju neuspesne pretrage vraca se -1 */
101     return -1;
}
```

```
103 int main(int argc, char **argv)
105 {
    int a[MAX];
107     int n, i, x;
    struct timespec time1, time2, time3, time4, time5, time6;
109     FILE *f;

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

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

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

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

137     /* Linearna pretraga */
    printf("Linearna pretraga:\n");
139     /* Vreme proteklo od Nove godine 1970 */
    clock_gettime(CLOCK_REALTIME, &time1);
    i = linearna_pretraga(a, n, x);
    /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
143     linearnu pretragu */
    clock_gettime(CLOCK_REALTIME, &time2);
145     if (i == -1)
        printf("Element nije u nizu\n");
    else
147         printf("Element je u nizu na poziciji %d\n", i);

149     /* Binarna pretraga */
151     printf("Binarna pretraga:\n");
    clock_gettime(CLOCK_REALTIME, &time3);
153     i = binarna_pretraga(a, n, x);
```

```

155 clock_gettime(CLOCK_REALTIME, &time4);
156 if (i == -1)
157     printf("Element nije u nizu\n");
158 else
159     printf("Element je u nizu na poziciji %d\n", i);
160
161 /* Interpolaciona pretraga */
162 printf("Interpolaciona pretraga:\n");
163 clock_gettime(CLOCK_REALTIME, &time5);
164 i = interpolaciona_pretraga(a, n, x);
165 clock_gettime(CLOCK_REALTIME, &time6);
166 if (i == -1)
167     printf("Element nije u nizu\n");
168 else
169     printf("Element je u nizu na poziciji %d\n", i);
170
171 /* Podaci o izvršavanju programa bivaju upisani u log fajl */
172 if ((f = fopen("vremena.txt", "a")) == NULL) {
173     fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
174     exit(EXIT_FAILURE);
175 }
176
177 fprintf(f, "Dimenzija niza: %d\n", n);
178 fprintf(f, "\tLinearna: %10ld ns\n",
179         (time2.tv_sec - time1.tv_sec) * 1000000000 +
180         time2.tv_nsec - time1.tv_nsec);
181 fprintf(f, "\tBinarna: %19ld ns\n",
182         (time4.tv_sec - time3.tv_sec) * 1000000000 +
183         time4.tv_nsec - time3.tv_nsec);
184 fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
185         (time6.tv_sec - time5.tv_sec) * 1000000000 +
186         time6.tv_nsec - time5.tv_nsec);
187
188 /* Zatvaranje datoteke */
189 fclose(f);
190
191 return 0;
192 }

```

### 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;
8     /* Izlaz iz rekurzije */
    if (n <= 0)
10         return -1;

```



```
12  /* Ako je prvi element trazeni */
13  if (a[0] == x)
14      return 0;
15  /* Pretraga ostatka niza */
16  tmp = lin_pretraga_rek_sufiks(a + 1, n - 1, x);
17  return tmp < 0 ? tmp : tmp + 1;
18  }
19
20  int lin_pretraga_rek_prefiks(int a[], int n, int x)
21  {
22      /* Izlaz iz rekurzije */
23      if (n <= 0)
24          return -1;
25      /* Ako je poslednji element trazeni */
26      if (a[n - 1] == x)
27          return n - 1;
28      /* Pretraga ostatka niza */
29      return lin_pretraga_rek_prefiks(a, n - 1, x);
30  }
31
32  int bin_pretraga_rek(int a[], int l, int d, int x)
33  {
34      int srednji;
35      if (l > d)
36          return -1;
37      /* Sredisnja pozicija na kojoj se trazi vrednost x */
38      srednji = (l + d) / 2;
39      /* Ako je element na sredisnjoj poziciji trazeni */
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  }
51
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;
```

```
64     if (a[p] < x)
65         return interp_pretraga_rek(a, p + 1, d, x);
66     else
67         return interp_pretraga_rek(a, l, p - 1, x);
68 }
69
70 int main()
71 {
72     int a[MAX];
73     int x;
74     int i, indeks;
75
76     /* Ucitavanje trazenog broja */
77     printf("Unesite trazeni broj: ");
78     scanf("%d", &x);
79
80     /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
81        korisnik pritisne CTRL+D za naznaku kraja */
82     i = 0;
83     printf("Unesite sortiran niz elemenata: ");
84     while (scanf("%d", &a[i]) == 1) {
85         i++;
86     }
87
88     /* Linearna pretraga */
89     printf("Linearna pretraga\n");
90     indeks = lin_pretraga_rek_sufiks(a, i, x);
91     if (indeks == -1)
92         printf("Element se ne nalazi u nizu.\n");
93     else
94         printf("Pozicija elementa je %d.\n", indeks);
95
96     /* Binarna pretraga */
97     printf("Binarna pretraga\n");
98     indeks = bin_pretraga_rek(a, 0, i - 1, x);
99     if (indeks == -1)
100         printf("Element se ne nalazi u nizu.\n");
101     else
102         printf("Pozicija elementa je %d.\n", indeks);
103
104     /* Interpolaciona pretraga */
105     printf("Interpolaciona pretraga\n");
106     indeks = interp_pretraga_rek(a, 0, i - 1, x);
107     if (indeks == -1)
108         printf("Element se ne nalazi u nizu.\n");
109     else
110         printf("Pozicija elementa je %d.\n", indeks);
111
112     return 0;
113 }
```

## Rešenje 3.3

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
7
8 /* 0 svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
9
10 typedef struct {
11     /* Indeks mora biti tipa long jer su podaci u datoteci preveliki za
12      int, npr. 20140123 */
13     long indeks;
14     char ime[MAX_DUZINA];
15     char prezime[MAX_DUZINA];
16 } Student;
17
18 /* Ucitani niz studenata ce biti sortiran rastuce prema indeksu, jer
   su studenti u datoteci vec sortirani. Iz tog razloga pretraga po
19 indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
   jer nema garancije da postoji uredjenje po prezimenu. */
20
21 /* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
   sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
   ako element nije pronadjen. */
22
23 int binarna_pretraga(Student a[], int n, long x)
24 {
25     int levi = 0;
26     int desni = n - 1;
27     int srednji;
28     /* Dokle god je indeks levi levo od indeksa desni */
29     while (levi <= desni) {
30         /* Racuna se srednja pozicija */
31         srednji = (levi + desni) / 2;
32         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
           se trazeni indeks mora nalaziti u levoj polovini niza */
33         if (x < a[srednji].indeks)
34             desni = srednji - 1;
35         /* Ako je pak manji od trazenog, tada se on mora nalaziti u
           desnoj polovini niza */
36         else if (x > a[srednji].indeks)
37             levi = srednji + 1;
38         else
39             /* Ako je jednak trazenom indeksu x, tada je pronadjen student
               sa trazenom indeksom na poziciji srednji */
40             return srednji;
41     }
42     /* Ako nije pronadjen, vraca se -1 */
43     return -1;
44 }
45
46 }

```

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

62 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
64    prosledjuju kao argumenti komandne linije */
int main(int argc, char *argv[])
66 {
    Student dosije[MAX_STUDENATA];
68     FILE *fin = NULL;
    int i;
70     int br_studenata = 0;
    long trazen_indeks = 0;
72     char trazeno_prezime[MAX_DUZINA];
    int bin_pretraga;
74
    /* Provera da li je korisnik prilikom poziva programa prosledio ime
76     datoteke sa informacijama o studentima i opciju pretrage */
    if (argc != 3) {
78         fprintf(stderr,
            "Greska: Program se poziva sa %s ime_datoteke opcija\n",
80             argv[0]);
        exit(EXIT_FAILURE);
82     }

84     /* Provera prosledjene opcije */
    if (strcmp(argv[2], "-indeks") == 0)
        bin_pretraga = 1;
86     else if (strcmp(argv[2], "-prezime") == 0)
        bin_pretraga = 0;
88     else {
90         fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
        exit(EXIT_FAILURE);
92     }

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

102     /* Citanje se vrši sve dok postoji red sa informacijama o studentu
```

```

104     */
105     i = 0;
106     while (1) {
107         if (i == MAX_STUDENATA)
108             break;
109         if (fscanf
110             (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
111              dosije[i].prezime) != 3)
112             break;
113         i++;
114     }
115     br_studenata = i;
116
117     /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
118     fclose(fin);
119
120     /* Pretraga po indeksu */
121     if (bin_pretraga) {
122         /* Unos indeksa koji se binarno trazi u nizu */
123         printf("Unesite indeks studenta cije informacije zelite: ");
124         scanf("%ld", &trazen_indeks);
125         i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
126         /* Rezultat binarne pretrage */
127         if (i == -1)
128             printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
129         else
130             printf("Indeks: %ld, Ime i prezime: %s %s\n",
131                    dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132     }
133     /* Pretraga po prezimenu */
134     else {
135         /* Unos prezimena koje se linearno trazi u nizu */
136         printf("Unesite prezime studenta cije informacije zelite: ");
137         scanf("%s", trazeno_prezime);
138         i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
139         /* Rezultat linearne pretrage */
140         if (i == -1)
141             printf("Ne postoji student sa prezimenom %s\n",
142                    trazeno_prezime);
143         else
144             printf("Indeks: %ld, Ime i prezime: %s %s\n",
145                    dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
146     }
147     return 0;
148 }

```

### Rešenje 3.4

```

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

```

```
4
#define MAX_STUDENATA 128
6
#define MAX_DUZINA 16

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

14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
16 {
    /* Ako je pozicija elementa na levom kraju veca od pozicije
18    elementa na desnom kraju dela niza koji se pretrazuje, onda se
    zapravo pretrazuje prazan deo niza. U praznom delu niza nema
20    trazenog elementa pa se vraca -1 */
    if (levi > desni)
22        return -1;
    /* Racunanje pozicije srednjeg elementa */
    int srednji = (levi + desni) / 2;
24    /* Da li je srednji bas onaj trazeni */
    if (a[srednji].indeks == x) {
26        return srednji;
    }
28    /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
    poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
30    je poznato da je niz sortiran po indeksu u rastucem poretку. */
    if (x < a[srednji].indeks)
32        return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
    /* Inace ga treba traziti u desnoj polovini */
34    else
    return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
36 }

38
int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
    /* Ako je niz prazan, vraca se -1 */
42    if (n == 0)
        return -1;
    /* Kako se trazi prvi student sa trazenim prezimenom, pocinje se sa
44    prvim studentom u nizu. */
    if (strcmp(a[0].prezime, x) == 0)
46        return 0;
    int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
48    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;
57     /* Ako se trazi poslednji student sa trazanim prezimenom, pocinje
58        se sa poslednjim studentom u nizu. */
59     if (strcmp(a[n - 1].prezime, x) == 0)
60         return n - 1;
61     return linearna_pretraga_rekurzivna(a, n - 1, x);
62 }

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

75     /* Provera da li je korisnik prilikom poziva programa prosledio ime
76        datoteke sa informacijama o studentima i opciju pretrage */
77     if (argc != 3) {
78         fprintf(stderr,
79             "Greska: Program se poziva sa %s ime_datoteke opcija\n",
80             argv[0]);
81         exit(EXIT_FAILURE);
82     }

83
84     /* Provera prosledjene opcije */
85     if (strcmp(argv[2], "-indeks") == 0)
86         bin_pretraga = 1;
87     else if (strcmp(argv[2], "-prezime") == 0)
88         bin_pretraga = 0;
89     else {
90         fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
91         exit(EXIT_FAILURE);
92     }

93
94     /* Otvaranje datoteke */
95     fin = fopen(argv[1], "r");
96     if (fin == NULL) {
97         fprintf(stderr,
98             "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
99         exit(EXIT_FAILURE);
100     }

101
102     /* Citanje se vrši sve dok postoji red sa informacijama o studentu
103        */
104     i = 0;
105     while (1) {
106         if (i == MAX_STUDENATA)

```

```

108     break;
109     if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
110         dosije[i].prezime) != 3)
        break;
111     i++;
112 }
113 br_studenata = i;
114
115 /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
116 fclose(fin);
117
118 /* Pretraga po indeksu */
119 if (bin_pretraga) {
120     /* Unos indeksa koji se binarno trazi u nizu */
121     printf("Unesite indeks studenta cijje informacije zelite: ");
122     scanf("%ld", &trazen_indeks);
123     i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata,
124                                     trazen_indeks);
125
126     /* Rezultat binarne pretrage */
127     if (i == -1)
128         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
129     else
130         printf("Indeks: %ld, Ime i prezime: %s %s\n",
131               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132 }
133 /* Pretraga po prezimenu */
134 else {
135     /* Unos prezimena koje se linearno trazi u nizu */
136     printf("Unesite prezime studenta cijje informacije zelite: ");
137     scanf("%s", trazeno_prezime);
138     i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
139                                         trazeno_prezime);
140
141     /* Rezultat linearne pretrage */
142     if (i == -1)
143         printf("Ne postoji student sa prezimenom %s\n",
144               trazeno_prezime);
145     else
146         printf("Indeks: %ld, Ime i prezime: %s %s\n",
147               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
148 }
149 return 0;
150 }

```

### Rešenje 3.5

```

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

```



```
/* Struktura koja opisuje tacku u ravni */
7 typedef struct Tacka {
    float x;
9    float y;
} Tacka;

11
/* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13 pocetka (0,0) */
float rastojanje(Tacka A)
15 {
    return sqrt(A.x * A.x + A.y * A.y);
17 }

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

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

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

```

59     if (fabs(t[i].y) < fabs(najbliza.y)) {
        najbliza = t[i];
    }
61 }
    return najbliza;
63 }

65 #define MAX 1024

67 int main(int argc, char *argv[])
{
69     FILE *ulaz;
    Tacka tacke[MAX];
71     Tacka najbliza;
    int i, n;

73     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75     ime datoteke sa tackama */
    if (argc < 2) {
77         fprintf(stderr,
79             "koriscenje programa: %s ime_datoteke\n", argv[0]);
        return EXIT_FAILURE;
    }

81     /* Otvaranje datoteke za citanje */
    ulaz = fopen(argv[1], "r");
83     if (ulaz == NULL) {
85         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[1]);
87         return EXIT_FAILURE;
    }

89     /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
91     tackama; i predstavlja indeks tekuce tacke */
    i = 0;
93     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
        i++;
95     }
    n = i;

97     /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
99     dodatnih argumenata */
    if (argc == 2)
101        /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
        najbliza = najbliza_koordinatnom(tacke, n);
103    /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
        pitanju opcija -x */
105    else if (strcmp(argv[2], "-x") == 0)
        /* Racuna se rastojanje u odnosu na x osu */
107        najbliza = najbliza_x_osi(tacke, n);
    /* Ako je u pitanju opcija -y */
109    else if (strcmp(argv[2], "-y") == 0)

```

```

111     /* Racuna se rastojanje u odnosu na y osu */
    najbliza = najbliza_y_osi(tacke, n);
113 else {
    /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
    korisnika i prekida se izvršavanje programa */
115     fprintf(stderr, "Pogresna opcija\n");
    return EXIT_FAILURE;
117 }

119 /* Stapanje 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
int main()
8  {
    double l, d, s;

10
    /* Kako je u pitanju interval [0, 2] leva granica je 0, a desna 2
    */
12     l = 0;
    d = 2;

14
    /* Sve dok se ne pronadje trazena vrednost argumenta */
16     while (1) {
        /* Polovi se interval */
18         s = (l + d) / 2;
        /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
        tacnosti, prekida se pretraga */
20         if (fabs(cos(s)) < EPS) {
            break;
22         }
        /* Ako je nula u levom delu intervala, nastavlja se pretraga na
        [l, s] */
24         if (cos(l) * cos(s) < 0)
            d = s;
        else
26             /* Inace, na intervalu [s, d] */
            l = s;
28
30

```

```
    }  
32  
    /* Stampanje vrednost trazene tacke */  
34    printf("%g\n", s);  
  
36    return 0;  
}
```

### Rešenje 3.7

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

```
42     return 0;
    }
```

### Rešenje 3.8

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

## Rešenje 3.9

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

```

51     printf("%u %u\n", logaritam_a(x), logaritam_b(x));
53     return 0;
}

```

### Rešenje 3.11

```

1  #include<stdio.h>
   #define MAX 256
3
   /* Iterativna verzija funkcije koja sortira niz celih brojeva,
   primenom algoritma Selection Sort */
5  void selectionSort(int a[], int n)
7  {
   int i, j;
   int min;
   int pom;
11
   /* U svakoj iteraciji ove petlje se pronalazi najmanji element
   medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
13   poziciju i, dok se element na poziciji i premesta na poziciju
   min, na kojoj se nalazio najmanji od gore navedenih elemenata.
15   */
   for (i = 0; i < n - 1; i++) {
17       /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
       najmanji od elemenata a[i],...,a[n-1]. */
19       min = i;
       for (j = i + 1; j < n; j++)
21           if (a[j] < a[min])
               min = j;
23       /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
       su (i) i min razliciti, inace je nepotrebno. */
25       if (min != i) {
           pom = a[i];
           a[i] = a[min];
           a[min] = pom;
27       }
29   }
31 }

33 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
   sortiranom nizu celih brojeva */
35 int najmanje_rastojanje(int a[], int n)
   {
37     int i, min;
     min = a[1] - a[0];
39     for (i = 2; i < n; i++)
         if (a[i] - a[i - 1] < min)
41             min = a[i] - a[i - 1];
     return min;
43 }

```

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

### Rešenje 3.12

```
1 #include<stdio.h>
2 #include<string.h>
3
4 #define MAX_DIM 128
5
6 /* Funkcija za sortiranje niza karaktera */
7 void selectionSort(char s[])
8 {
9     int i, j, min;
10    char pom;
11    for (i = 0; s[i] != '\0'; i++) {
12        min = i;
13        for (j = i + 1; s[j] != '\0'; j++)
14            if (s[j] < s[min])
15                min = j;
16        if (min != i) {
17            pom = s[i];
18            s[i] = s[min];
19            s[min] = pom;
20        }
21    }
22 }
23
24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
25 int anagrami(char s[], char t[])
26 {
27     int i;
28
29     /* Ako dve niske imaju razlicit broj karaktera onda one nisu
```



```

30     anagrami */
31     if (strlen(s) != strlen(t))
32         return 0;

34     /* Sortiramo niske */
35     selectionSort(s);
36     selectionSort(t);

38     /* Dve sortirane niske su anagrami ako i samo ako su jednake */
39     for (i = 0; s[i] != '\0'; i++)
40         if (s[i] != t[i])
41             return 0;
42     return 1;
43 }

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

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

54
55     /* Poziv funkcije */
56     if (anagrami(s, t))
57         printf("jesu\n");
58     else
59         printf("nisu\n");

60
61     return 0;
62 }

```

### Rešenje 3.13

```

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

3
4  /* Funkcija za sortiranje niza */
5  void selectionSort(int s[], int n)
6  {
7      int i, j, min;
8      char pom;
9      for (i = 0; i < n; i++) {
10         min = i;
11         for (j = i + 1; j < n; j++)
12             if (s[j] < s[min])
13                 min = j;
14         if (min != i) {
15             pom = s[i];

```

```

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

```

### Rešenje 3.14

```

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

```

### Rešenje 3.15

```

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

```

```

/* Datoteka sort.c */
2
#include "sort.h"
4
#define MAX 1000000
6
/* Funkcija sortira niz celih brojeva metodom sortiranja izborom.
   Ideja algoritma je sledeca: U svakoj iteraciji pronalazi se
   najmanji element i premesta se na pocetak niza. Dakle, u prvoj
10 iteraciji, pronalazi se najmanji element, i dovodi na nulto mesto
   u nizu. U i-toj iteraciji najmanjih i elemenata su vec na svojim
   pozicijama, pa se od i+1 do n-1 elementa trazi najmanji, koji se
12 dovodi na i+1 poziciju. */
14 void selectionsort(int a[], int n)
{

```

```
16  int i, j;
17  int min;
18  int pom;

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
23     min, na kojoj se nalazio najmanji od gore navedenih elemenata.
24     */
25  for (i = 0; i < n - 1; i++) {
26      /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
27         najmanji od elemenata a[i],...,a[n-1]. */
28      min = i;
29      for (j = i + 1; j < n; j++)
30          if (a[j] < a[min])
31              min = j;

32      /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
33         su (i) i min razliciti, inace je nepotrebno. */
34      if (min != i) {
35          pom = a[i];
36          a[i] = a[min];
37          a[min] = pom;
38      }
39  }
40 }

42 /* Funkcija sortira niz celih brojeva metodom sortiranja umetanjem.
43    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
45    iteraciji treba element a[i] umetnuti na pravu poziciju medju
46    prvih i elemenata tako da se dobije niz duzine i+1 koji je
47    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
49    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
51    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
53    levim susedom. Ovim uzastopnim premestanjem se a[i] umece na pravo
54    mesto u nizu. */
55 void insertionsort(int a[], int n)
56 {
57     int i, j;

58     /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
59        sortiran */
60     for (i = 1; i < n; i++) {
61
62         /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
63            potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...,a[i]
64            bude sortiran. Indeks j je trenutna pozicija na kojoj se
65            element koji se umece nalazi. Petlja se zavrшава ili kada
```

```

        element dodje do levog kraja (j==0) ili kada se naidje na
68     element a[j-1] koji je manji od a[j]. */
    for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
70         int temp = a[j];
        a[j] = a[j - 1];
72         a[j - 1] = temp;
    }
74 }
}

76 /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
78 algoritma je sledeca: prolazi se kroz niz redom poredeci susedne
elemente, i pri tom ih zamenjujuci ako su u pogresnom poretku.
80 Ovim se najveći element poput mehurica istiskuje na "povrsinu",
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
elemenata niza bez poslednjeg koji je postavljen na pravu
84 poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
kracim prefiksima niza, cime se jedan po jedan istiskuju
86 elementi na svoje prave pozicije. */
void bubblesort(int a[], int n)
88 {
    int i, j;
90     int ind;

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

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

102                /* Promenljiva ind registruje da je bilo premestanja. Samo u
104                tom slucaju ima smisla ici na sledecu iteraciju, jer ako
nije bilo premestanja, znaci da su svi elementi vec u
106                dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
108                ispravno, ali manje efikasano, jer bi se cesto nepotrebno
vrsila mnoga upoređivanja, kada je vec jasno da je
110                sortiranje završeno. */
                ind = 1;
112            }
    }
114 }

116 /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
118 sastoji u tome da se kroz algoritam umetanja prolazi vise puta; u
prvom prolazu, umesto koraka 1 uzima se neki korak h koji je manji

```

```

120     od n (sto omogućuje razmenu udaljenih elemenata) i tako se dobija
122     h-sortiran niz, tj. niz u kome su elementi na rastojanju h
124     sortirani, mada susedni elementi to ne moraju biti. U drugom
126     prolazu kroz isti algoritam sprovodi se isti postupak ali za manji
128     korak h. Sa prolazima se nastavlja sve do koraka h = 1, u kome se
130     dobija potpuno sortirani niz. Izbor pocetne vrednosti za h, i
132     nacina njegovog smanjivanja menja u nekim slucajevima brzinu
134     algoritma, ali bilo koja vrednost ce rezultovati ispravnim
136     sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
138     vrednost 1. */
140 void shellsort(int a[], int n)
142 {
144     int h = n / 2, i, j;
146     while (h > 0) {
148         /* Insertion sort sa korakom h */
150         for (i = h; i < n; i++) {
152             int temp = a[i];
154             j = i;
156             while (j >= h && a[j - h] > temp) {
158                 a[j] = a[j - h];
160                 j -= h;
162             }
164             a[j] = temp;
166         }
168         h = h / 2;
170     }
172 }

174 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem. Sortiranje
176 se vrši od elementa na poziciji l do onog na poziciji d. Na
178 pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a d je
180 jednako poslednjem validnom indeksu u nizu. Funkcija niz podeli na
182 dve polovine, levu i desnu, koje zatim rekursivno sortira. Od ova
184 dva sortirana podniza, sortiran niz se dobija ucesljavanjem, tj.
186 istovremenim prolaskom kroz oba niza i izborom trenutnog manjeg
188 elementa koji se smesta u pomocni niz. Na kraju algoritma,
190 sortirani elementi su u pomocnom nizu, koji se kopira u originalni
192 niz. */
194 void mergesort(int a[], int l, int d)
196 {
198     int s;
199     static int b[MAX];           /* Pomocni niz */
200     int i, j, k;
202     /* Izlaz iz rekurziije */
203     if (l >= d)
204         return;
206     /* Odredjivanje sredisnjeg indeksa */
207     s = (l + d) / 2;
209     /* Rekursivni pozivi */

```

```

172 mergesort(a, l, s);
    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;
178 j = s + 1;
    k = 0;

180 /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
182 while (i <= s && j <= d) {
    if (a[i] < a[j])
184         b[k++] = a[i++];
    else
186         b[k++] = a[j++];
}

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

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

200 /* Prepisuje se "ucesljani" niz u originalni niz */
202 for (k = 0, i = l; i <= d; i++, k++)
    a[i] = b[k];
204 }

206 /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
208 {
    int tmp = a[i];
210 a[i] = a[j];
    a[j] = tmp;
212 }

214 /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r. Njena
    ideja sortiranja je izbor jednog elementa niza, koji se naziva
216 pivot, i koji se dovodi na svoje mesto. Posle ovog koraka, svi
    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
    sortiran, potrebno je sortirati elemente levo (manje) od njega, i
220 elemente desno (vece). Kako su dimenzije ova dva podniza manje od
    dimenzije pocetnog niza koji je trebalo sortirati, ovaj deo moze
222

```



```

224     se uraditi rekurzivno. */
void quicksort(int a[], int l, int r)
{
226     int i, pivot_position;

228     /* Izlaz iz rekurzije -- prazan niz */
    if (l >= r)
230         return;

232     /* Partitionisanje niza. Svi elementi na pozicijama levo od
234     pivot_position (izuzev same pozicije l) su strogo manji od
    pivota. Kada se pronadje neki element manji od pivota, uvecava
236     se promenljiva pivot_position i na tu poziciju se premesta
    nadjeni element. Na kraju ce pivot_position zaista biti pozicija
238     na koju treba smestiti pivot, jer ce svi elementi levo od te
    pozicije biti manji a desno biti veci ili jednaki od pivota. */
240     pivot_position = l;
    for (i = l + 1; i <= r; i++)
242         if (a[i] < a[l])
            swap(a, ++pivot_position, i);

244     /* Postavljanje pivota na svoje mesto */
    swap(a, l, pivot_position);

246     /* Rekurzivno sortiranje elementa manjih od pivota */
    quicksort(a, l, pivot_position - 1);
    /* Rekurzivno sortiranje elementa vecih od pivota */
250     quicksort(a, pivot_position + 1, r);
252 }

```

```

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

6  /* Maksimalna duzina niza */
#define MAX 1000000

8
int main(int argc, char *argv[])
10 {
    /******
12     tip_sortiranja == 0 => selectionsort, (podrazumevano)
    tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
14     tip_sortiranja == 2 => bubblesort,    -b opcija komandne linije
    tip_sortiranja == 3 => shellsort,        -s opcija komandne linije
16     tip_sortiranja == 4 => mergesort,      -m opcija komandne linije
    tip_sortiranja == 5 => quicksort,        -q opcija komandne linije
18     *****/
    int tip_sortiranja = 0;
20    /******
    tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)

```

```

22     tip_niza == 1 => rastuce sortirani nizovi,   -r opcija
23     tip_niza == 2 => opadajuće sortirani nizovi, -o opcija
24     *****/
25     int tip_niza = 0;
26
27     /* Dimenzija niza koji se sortira */
28     int dimenzija;
29     int i;
30     int niz[MAX];
31
32     /* Provera argumenata komandne linije */
33     if (argc < 2) {
34         fprintf(stderr,
35             "Program zahteva bar 2 argumenta komandne linije!\n");
36         exit(EXIT_FAILURE);
37     }
38
39     /* Ocitanje opcija i argumenata prilikom poziva programa */
40     for (i = 1; i < argc; i++) {
41         /* Ako je u pitanju opcija... */
42         if (argv[i][0] == '-') {
43             switch (argv[i][1]) {
44                 case 'i':
45                     tip_sortiranja = 1;
46                     break;
47                 case 'b':
48                     tip_sortiranja = 2;
49                     break;
50                 case 's':
51                     tip_sortiranja = 3;
52                     break;
53                 case 'm':
54                     tip_sortiranja = 4;
55                     break;
56                 case 'q':
57                     tip_sortiranja = 5;
58                     break;
59                 case 'r':
60                     tip_niza = 1;
61                     break;
62                 case 'o':
63                     tip_niza = 2;
64                     break;
65                 default:
66                     printf("Pogresna opcija -%c\n", argv[i][1]);
67                     return 1;
68                     break;
69             }
70         }
71         /* Ako je u pitanju argument, onda je to duzina niza koji treba
72            da se sortira */
73         else {

```

```

74     dimenzija = atoi(argv[i]);
75     if (dimenzija <= 0 || dimenzija > MAX) {
76         fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
77         exit(EXIT_FAILURE);
78     }
79 }
80 }

82 /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
83 niza dobijenom iz komandne linije. srandom() funkcija
84 obezbedjuje novi seed za pozivanje random funkcije, i kako
85 generisani niz ne bi uvek bio isti seed je postavljen na tekuce
86 vreme u sekundama od Nove godine 1970. random()%100 daje brojeve
87 izmedju 0 i 99 */
88 srandom(time(NULL));
89 if (tip_niza == 0)
90     for (i = 0; i < dimenzija; i++)
91         niz[i] = random();
92 else if (tip_niza == 1)
93     for (i = 0; i < dimenzija; i++)
94         niz[i] = i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
95 else
96     for (i = 0; i < dimenzija; i++)
97         niz[i] = i == 0 ? random() % 100 : niz[i - 1] - random() % 100;
98
99 /* Ispisivanje elemenata niza */
100 /******
101 Ovaj deo je iskomentarisano 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 */

```

```

126  /*****
128      Ovaj deo je iskomentarisan jer vreme potrebno za njegovo
130      izvršavanje ne bi trebalo da bude ukljuceno u vreme izmereno
132      programom time. Takodje, kako je svrha ovog programa da prikaze
134      vremena razlicitih algoritama sortiranja, dimenzije nizova ce
136      biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
138      od toliko elemenata. Ovaj deo je koriscen u razvoju programa
140      zarad testiranja korektnosti.

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

140  return 0;
    }

```

### Rešenje 3.16

```

#include <stdio.h>
#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;
      /* U slucaju da je dimenzija treceg niza manja od neophodne,
         funkcija vraca -1 */
10     if (dim3 < dim1 + dim2)
12         return -1;

      /* 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])
18             niz3[k++] = niz1[i++];
            else
                niz3[k++] = niz2[j++];
    }
    /* Ostatak prvog niza prepisujemo u treci */
22     while (i < dim1)
24         niz3[k++] = niz1[i++];

    /* Ostatak drugog niza prepisujemo u treci */
26     while (j < dim2)
28         niz3[k++] = niz2[j++];
    return dim1 + dim2;
}

30 int main()
32 {

```

```

34     int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
35     int i = 0, j = 0, k, dim3;
36
37     /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
38        Pretpostavka je da na ulazu nece biti vise od MAX_DIM elemenata
39        */
38     printf("Unesite elemente prvog niza: ");
39     while (1) {
40         scanf("%d", &niz1[i]);
41         if (niz1[i] == 0)
42             break;
43         i++;
44     }
45     printf("Unesite elemente drugog niza: ");
46     while (1) {
47         scanf("%d", &niz2[j]);
48         if (niz2[j] == 0)
49             break;
50         j++;
51     }
52
53     /* Poziv trazene funkcije */
54     dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);
55
56     /* Ispis niza */
57     for (k = 0; k < dim3; k++)
58         printf("%d ", niz3[k]);
59     printf("\n");
60
61     return 0;
62 }

```

### Rešenje 3.17

```

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 }

```

```
18 }
19
20 /* Otvaranje datoteke zadate prvim argumentom komandne linije */
21 fin1 = fopen(argv[1], "r");
22 if (fin1 == NULL) {
23     fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
24     exit(EXIT_FAILURE);
25 }
26
27 /* Otvaranje datoteke zadate drugim argumentom komandne linije */
28 fin2 = fopen(argv[2], "r");
29 if (fin2 == NULL) {
30     fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
31     exit(EXIT_FAILURE);
32 }
33
34 /* Otvaranje datoteke za upis rezultata */
35 fout = fopen("ceo-tok.txt", "w");
36 if (fout == NULL) {
37     fprintf(stderr,
38         "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
39     exit(EXIT_FAILURE);
40 }
41
42 /* Citanje narednog studenta iz prve datoteke */
43 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
44     kraj1 = 1;
45
46 /* Citanje narednog studenta iz druge datoteke */
47 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
48     kraj2 = 1;
49
50 /* Sve dok nije dostignut kraj neke datoteke */
51 while (!kraj1 && !kraj2) {
52     if (strcmp(ime1, ime2) < 0) {
53         /* Ime i prezime iz prve datoteke je leksikografski ranije, i
54            biva upisano u izlaznu datoteku */
55         fprintf(fout, "%s %s\n", ime1, prezime1);
56         /* Citanje narednog studenta iz prve datoteke */
57         if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
58             kraj1 = 1;
59     } else {
60         /* Ime i prezime iz druge datoteke je leksikografski ranije, i
61            biva upisano u izlaznu datoteku */
62         fprintf(fout, "%s %s\n", ime2, prezime2);
63         /* Citanje narednog studenta iz druge datoteke */
64         if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
65             kraj2 = 1;
66     }
67 }
68
69 /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
```

```

70     druge datoteke, onda ima jos studenata u prvoj datoteci, koje
       treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
72     */
       while (!kraj1) {
           fprintf(fout, "%s %s\n", ime1, prezime1);
74           if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
               kraj1 = 1;
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 3.18

```

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++) {
31             if (rastojanje(t[j]) < rastojanje(t[min])) {
                min = j;
33             }
        }
35         if (min != i) {
            tmp = t[i];
37             t[i] = t[min];
            t[min] = tmp;
39         }
    }
41 }

43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
void sortiraj_po_x(Tacka t[], int n)
45 {
    int min, i, j;
    Tacka tmp;
47
    for (i = 0; i < n - 1; i++) {
49         min = i;
        for (j = i + 1; j < n; j++) {
51             if (abs(t[j].x) < abs(t[min].x)) {
                min = j;
53             }
        }
55         if (min != i) {
            tmp = t[i];
57             t[i] = t[min];
            t[min] = tmp;
59         }
    }
61 }

63
65 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
void sortiraj_po_y(Tacka t[], int n)
67 {
    int min, i, j;
    Tacka tmp;
69
    for (i = 0; i < n - 1; i++) {
71         min = i;
        for (j = i + 1; j < n; j++) {
73             if (abs(t[j].y) < abs(t[min].y)) {
                min = j;
75             }
        }
    }
}
```



```
    }
77     if (min != i) {
79         tmp = t[i];
81         t[i] = t[min];
83         t[min] = tmp;
    }
}

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        brojem 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 }
```

```

129  /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
      "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
      (karakter -), a karakter argv[1][1] odredjuje kriterijum
131  sortiranja */
      switch (argv[1][1]) {
133  case 'x':
          /* Sortiranje po vrednosti x koordinate */
135  sortiraj_po_x(tacke, n);
          break;
137  case 'y':
          /* Sortiranje po vrednosti y koordinate */
139  sortiraj_po_y(tacke, n);
          break;
141  case 'o':
          /* Sortiranje po udaljenosti od koordinatnog pocetka */
143  sortiraj_po_rastojanju(tacke, n);
          break;
145  }

147  /* Upisivanje dobijenog niza u izlaznu datoteku */
      for (i = 0; i < n; i++) {
149  fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
      }

151
      /* Zatvaranje otvorenih datoteka */
153  fclose(ulaz);
      fclose(izlaz);
155
      return 0;
157 }

```

### Rešenje 3.19

```

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

4
#define MAX 1000
6  #define MAX_DUZINA 16

8  /* Struktura koja reprezentuje jednog gradjanina */
typedef struct gr {
10  char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
12 } Gradjanin;

14 /* Funkcija sortira niz gradjana rastuce po imenima */
void sort_ime(Gradjanin a[], int n)
16 {
    int i, j;
18  int min;

```

```
20   Gradjanin pom;
21
22   for (i = 0; i < n - 1; i++) {
23       /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
24          najmanji od elemenata a[i].ime,...,a[n-1].ime. */
25       min = i;
26       for (j = i + 1; j < n; j++)
27           if (strcmp(a[j].ime, a[min].ime) < 0)
28               min = j;
29       /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
30          su (i) i min razliciti, inace je nepotrebno. */
31       if (min != i) {
32           pom = a[i];
33           a[i] = a[min];
34           a[min] = pom;
35       }
36   }
37
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   }
61
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;
```

```

72 }
73
74 int main()
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;
78     int i, n;
79     FILE *fp = NULL;
80
81     /* Otvaranje datoteke */
82     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
83         fprintf(stderr,
84             "Neuspesno otvaranje datoteke biracki-spisak.txt.\n");
85         exit(EXIT_FAILURE);
86     }
87
88     /* Citanje sadrzaja */
89     for (i = 0;
90         fscanf(fp, "%s %s", spisak1[i].ime,
91             spisak1[i].prezime) != EOF; i++)
92         spisak2[i] = spisak1[i];
93     n = i;
94
95     /* Zatvaranje datoteke */
96     fclose(fp);
97
98     sort_ime(spisak1, n);
99
100     /******
101     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
102     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
103
104     printf("Biracki spisak [uredjen prema imenima]:\n");
105     for(i=0; i<n; i++)
106         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107     *****/
108
109     sort_prezime(spisak2, n);
110
111     /******
112     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
113     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
114
115     printf("Biracki spisak [uredjen prema prezimenima]:\n");
116     for(i=0; i<n; i++)
117         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
118     *****/
119
120     /* Linearno pretrazivanje nizova */
121     for (i = 0; i < n; i++)
122         if (i == linearna_pretraga(spisak2, n, &spisak1[i]))

```

```

124         isti_rbr++;

126     /* Alternativno (efikasnije) resenje */
126     /*****
128         for(i=0; i<n ;i++)
128             if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
129                 strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
130                 isti_rbr++;
130     *****/

132     /* Ispis rezultata */
134     printf("%d\n", isti_rbr);

136     exit(EXIT_SUCCESS);
}

```

### Rešenje 3.21

```

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

5  #define MAX_BR_RECII 128
   #define MAX_DUZINA_RECII 32

7  /* Funkcija koja izracunava broj suglasnika u reci */
9  int broj_suglasnika(char s[])
10 {
11     char c;
12     int i;
13     int suglasnici = 0;
14     /* Prolaz 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            */
21             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
22                 suglasnici++;
23         }
24     }
25     /* Vraca se izracunata vrednost */
26     return suglasnici;
27 }

29 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
   duzini reci se mora proslediti zbog pravilnog upravljanja
31     memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)

```

```
33 {
34     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35         duzina_j, duzina_min;
36     char tmp[MAX_DUZINA_RECII];
37     for (i = 0; i < n - 1; i++) {
38         min = i;
39         for (j = i; j < n; j++) {
40             /* Prvo se uporedjuje broj suglasnika */
41             broj_suglasnika_j = broj_suglasnika(reci[j]);
42             broj_suglasnika_min = broj_suglasnika(reci[min]);
43             if (broj_suglasnika_j < broj_suglasnika_min)
44                 min = j;
45             else if (broj_suglasnika_j == broj_suglasnika_min) {
46                 /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
47                    se duzine */
48                 duzina_j = strlen(reci[j]);
49                 duzina_min = strlen(reci[min]);
50
51                 if (duzina_j < duzina_min)
52                     min = j;
53                 else
54                     /* Ako reci imaju i isti broj suglasnika i iste duzine,
55                        uporedjuju se leksikografski */
56                     if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
57                         min = j;
58             }
59         }
60         if (min != i) {
61             strcpy(tmp, reci[min]);
62             strcpy(reci[min], reci[i]);
63             strcpy(reci[i], tmp);
64         }
65     }
66 }
67
68 int main()
69 {
70     FILE *ulaz;
71     int i = 0, n;
72
73     /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
74        a drugi maksimalnu duzinu pojedinačne reci */
75     char reci[MAX_BR_RECII][MAX_DUZINA_RECII];
76
77     /* Otvaranje datoteke niske.txt za citanje */
78     ulaz = fopen("niske.txt", "r");
79     if (ulaz == NULL) {
80         fprintf(stderr,
81             "Greska prilikom otvaranja datoteke niske.txt!\n");
82         return 0;
83     }
84 }
```

```

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_REC)
          break;
91      /* Priprema brojac za narednu iteraciju */
      i++;
93  }

95  /* n je duzina niza reci i predstavlja poslednju vrednost
     koriscenog brojac */
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 3.22

```

#include <stdio.h>
2  #include <stdlib.h>
   #include <string.h>
4
   #define MAX_ARTIKALA 100000
6
   /* Struktura koja predstavlja jedan artikal */
8  typedef struct art {
    long kod;
10     char naziv[20];
    char proizvođač[20];
12     float cena;
    } Artikal;
14

   /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
      traženim bar kodom */
16  int binarna_pretraga(Artikal a[], int n, long x)
18  {
    int levi = 0;
20     int desni = n - 1;

```

```
22  /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
24      /* Racuna se sredisnji indeks */
      int srednji = (levi + desni) / 2;
26      /* Ako je sredisnji element veci od trazenog, tada se trazeni
        mora nalaziti u levoj polovini niza */
28      if (x < a[srednji].kod)
        desni = srednji - 1;
30      /* Ako je sredisnji element manji od trazenog, tada se trazeni
        mora nalaziti u desnoj polovini niza */
32      else if (x > a[srednji].kod)
        levi = srednji + 1;
34      else
        /* Ako je sredisnji element jednak trazenom, tada je artikal sa
        bar kodom x pronadjen na poziciji srednji */
36      return srednji;
38    }
    /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
40    return -1;
  }

42  /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44  void selection_sort(Artikal a[], int n)
  {
46    int i, j;
    int min;
48    Artikal pom;

50    for (i = 0; i < n - 1; i++) {
      min = i;
52      for (j = i + 1; j < n; j++)
        if (a[j].kod < a[min].kod)
54          min = j;
      if (min != i) {
56        pom = a[i];
        a[i] = a[min];
58        a[min] = pom;
      }
60    }
  }

62  int main()
  {
64    Artikal asortiman[MAX_ARTIKALA];
    long kod;
    int i, n;
66    float racun;

70    FILE *fp = NULL;

72    /* Otvaranje datoteke */
    if ((fp = fopen("artikli.txt", "r")) == NULL) {
```



```

74     fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
       exit(EXIT_FAILURE);
76 }

78 /* Ucitavanje artikala */
       i = 0;
80     while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
                    asortiman[i].naziv, asortiman[i].proizvodjac,
82                    &asortiman[i].cena) == 4)
           i++;
84
       /* Zatvaranje datoteke */
86     fclose(fp);

88     n = i;

90     /* Sortira se celokupan asortiman prodavnice prema kodovima jer ce
       pri kucanju racuna prodavac unositi kod artikla. Prilikom
92     kucanja svakog racuna pretrazuje se asortiman, da bi se utvrdila
       cena artikla. Kucanje racuna obuhvata vise pretraga asortimana i
94     cilj je da ta operacija bude sto efikasnija. Zato se koristi
       algoritam binarne pretrage prilikom pretrazivanja po kodu
96     artikla. Iz tog razloga, potrebno je da asortiman bude sortiran
       po kodovima i to ce biti uradjeno primenom selection sort
98     algoritma. Sortiranje se vrsi samo jednom na pocetku, ali se
       zato posle artikli mogu brzo pretrazivati prilikom kucanja
100    proizvodljno puno racuna. Vreme koje se utrosi na sortiranje na
       pocetku izvorsavanja programa, kasnije se isplati jer se za
102    brojna trazanja artikla umesto linearne moze koristiti
       efikasnija binarna pretraga. */
104    selection_sort(asortiman, n);

106    /* Ispis stanja u prodavnici */
       printf
108     ("Asortiman:\nKOD           Naziv artikla       Ime
       proizvodjaca      Cena\n");
       for (i = 0; i < n; i++)
110         printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
                asortiman[i].naziv, asortiman[i].proizvodjac,
112                asortiman[i].cena);

114    kod = 0;
       while (1) {
116         printf("-----\n");
         printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
118         printf("- Za nov racun unesite kod artikla!\n");
         /* Unos bar koda provog artikla sledeceg kupca */
120         if (scanf("%ld", &kod) == EOF)
             break;
122         /* Trenutni racun novog kupca */
         racun = 0;
124         /* Za sve artikle trenutnog kupca */

```

```

126     while (1) {
127         /* Vrsi se njihov pronalazak u nizu */
128         if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
129             printf("\tGRESKA: Ne postoji proizvod sa traženim kodom!\n");
130         } else {
131             printf("\tTrazili ste:\t%s %s %12.2f\n",
132                 asortiman[i].naziv, asortiman[i].proizvodjac,
133                 asortiman[i].cena);
134             /* I dodavanje na ukupan racun */
135             racun += asortiman[i].cena;
136         }
137         /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
138            nema vise artikla */
139         printf("Unesite kod artikla [ili 0 za prekid]: \t");
140         scanf("%ld", &kod);
141         if (kod == 0)
142             break;
143     }
144     /* Stampanje ukupnog racuna trenutnog kupca */
145     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
146 }
147
148 printf("Kraj rada kase!\n");
149
150 exit(EXIT_SUCCESS);
151 }

```

### Rešenje 3.23

```

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;

```

```

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) {
30             pom = niz[min];
31             niz[min] = niz[i];
32             niz[i] = pom;
33         }
34     }
35 }
36
37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
38    zadataka opadajuće, a ukoliko neki studenti imaju isti broj
39    uradjenih zadataka sortiraju se po dužini imena rastuće. */
40 void sort_zadatke_pa_imena(Student niz[], int n)
41 {
42     int i, j;
43     int max;
44     Student pom;
45     for (i = 0; i < n - 1; i++) {
46         max = i;
47         for (j = i + 1; j < n; j++)
48             if (niz[j].zadaci > niz[max].zadaci)
49                 max = j;
50             else if (niz[j].zadaci == niz[max].zadaci
51                     && strlen(niz[j].ime) < strlen(niz[max].ime))
52                 max = j;
53         if (max != i) {
54             pom = niz[max];
55             niz[max] = niz[i];
56             niz[i] = pom;
57         }
58     }
59 }
60
61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
62    su bili opadajuće. Ukoliko neki studenti imaju isti broj casova,
63    sortiraju se opadajuće po broju uradjenih zadataka, a ukoliko se
64    i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65    prezimenu opadajuće. */
66 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
67 {
68     int i, j;
69     int max;
70     Student pom;
71     for (i = 0; i < n - 1; i++) {
72         max = i;
73         for (j = i + 1; j < n; j++)
74             if (niz[j].prisustvo > niz[max].prisustvo)

```

```
75         max = j;
76     else if (niz[j].prisustvo == niz[max].prisustvo
77             && niz[j].zadaci > niz[max].zadaci)
78         max = j;
79     else if (niz[j].prisustvo == niz[max].prisustvo
80             && niz[j].zadaci == niz[max].zadaci
81             && strcmp(niz[j].prezime, niz[max].prezime) > 0)
82         max = j;
83     if (max != i) {
84         pom = niz[max];
85         niz[max] = niz[i];
86         niz[i] = pom;
87     }
88 }
89 }

91 int main(int argc, char *argv[])
92 {
93     Student praktikum[MAX];
94     int i, br_studenata = 0;
95
96     FILE *fp = NULL;
97
98     /* Otvaranje datoteke za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
100         fprintf(stderr, "Neuspesno otvaranje datoteke aktivnost.txt.\n");
101         exit(EXIT_FAILURE);
102     }
103
104     /* Ucitavanje sadrzaja */
105     for (i = 0;
106         fscanf(fp, "%s%s%d%d", praktikum[i].ime,
107               praktikum[i].prezime, &praktikum[i].prisustvo,
108               &praktikum[i].zadaci) != EOF; i++);
109
110     /* Zatvaranje datoteke */
111     fclose(fp);
112     br_studenata = i;
113
114     /* Kreiranje prvog spiska studenata po prvom kriterijumu */
115     sort_ime_leksikografski(praktikum, br_studenata);
116     /* Otvaranje datoteke za pisanje */
117     if ((fp = fopen("dat1.txt", "w")) == NULL) {
118         fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
119         exit(EXIT_FAILURE);
120     }
121     /* Upis niza u datoteku */
122     fprintf
123     (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
124     for (i = 0; i < br_studenata; i++)
125         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
126               praktikum[i].prezime, praktikum[i].prisustvo,
127               praktikum[i].zadaci);
```

```

127  /* Zatvaranje datoteke */
    fclose(fp);

129

131  /* Kreiranje drugog spiska studenata po drugom kriterijumu */
    sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
133  if ((fp = fopen("dat2.txt", "w")) == NULL) {
        fprintf(stderr, "Neuspješno otvaranje datoteke dat2.txt.\n");
135  exit(EXIT_FAILURE);
    }

137  /* Upis niza u datoteku */
    fprintf(fp, "Studenti sortirani po broju zadataka opadajuće,\n");
139  fprintf(fp, "pa po dužini imena raste:\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, "Neuspješno 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 3.24

```

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

4
  #define KORAK 10

6
  /* Struktura koja opisuje jednu pesmu */
8  typedef struct {

```

```
10 char *izvodjac;
11 char *naslov;
12 int broj_gledanja;
13 } Pesma;

14 /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
   rad qsort funkcije) */
15 int uporedi_gledanost(const void *pp1, const void *pp2)
16 {
17     Pesma *p1 = (Pesma *) pp1;
18     Pesma *p2 = (Pesma *) pp2;
19
20     return p2->broj_gledanja - p1->broj_gledanja;
21 }

22
23 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
   funkcije) */
24 int uporedi_naslove(const void *pp1, const void *pp2)
25 {
26     Pesma *p1 = (Pesma *) pp1;
27     Pesma *p2 = (Pesma *) pp2;
28
29     return strcmp(p1->naslov, p2->naslov);
30 }

31
32 /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
   qsort funkcije) */
33 int uporedi_izvodjace(const void *pp1, const void *pp2)
34 {
35     Pesma *p1 = (Pesma *) pp1;
36     Pesma *p2 = (Pesma *) pp2;
37
38     return strcmp(p1->izvodjac, p2->izvodjac);
39 }

40
41
42
43
44 int main(int argc, char *argv[])
45 {
46     FILE *ulaz;
47     Pesma *pesme;
48     /* Pokazivac na deo memorije za
       cuvanje pesama */
49     int alocirano_za_pesme;
50     /* Broj mesta alociranih za pesme */
51     int i;
52     /* Redni broj pesme cije se
       informacije citaju */
53     int n;
54     /* Ukupan broj pesama */
55     int j, k;
56     char c;
57     int alocirano;
58     /* Broj mesta alociranih za propratne
       informacije o pesmama */
59     int broj_gledanja;
60
61     /* Priprema datoteke za citanje */
62     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
```

```
62     if (ulaz == NULL) {
63         printf("Greska pri otvaranju ulazne datoteke!\n");
64         return 0;
65     }
66
67     /* Citanje informacija o pesmama */
68     pesme = NULL;
69     alocirano_za_pesme = 0;
70     i = 0;
71
72     while (1) {
73
74         /* Proverava da li je dostignut kraj datoteke */
75         c = fgetc(ulaz);
76         if (c == EOF) {
77             /* Nema vise sadrzaja za citanje */
78             break;
79         } else {
80             /* Inace, vracamo procitani karakter nazad */
81             ungetc(c, ulaz);
82         }
83
84         /* Provera da li postoji dovoljno memorije za citanje nove pesme */
85         if (alocirano_za_pesme == i) {
86
87             /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
88              * novih KORAK mesta */
89             alocirano_za_pesme += KORAK;
90             pesme =
91                 (Pesma *) realloc(pesme,
92                                   alocirano_za_pesme * sizeof(Pesma));
93
94             /* Proverava da li je nova memorija uspesno realocirana */
95             if (pesme == NULL) {
96                 /* Ako nije ispisuje se obavestenje */
97                 printf("Problem sa alokacijom memorije!\n");
98                 /* I oslobadja sva memorija zauzeta do ovog koraka */
99                 for (k = 0; k < i; k++) {
100                     free(pesme[k].izvodjac);
101                     free(pesme[k].naslov);
102                 }
103                 free(pesme);
104                 return 0;
105             }
106
107             /* Ako jeste, nastavlja se sa citanjem pesama ... */
108             /* Cita se ime izvodjaca */
109             j = 0;
110             alocirano = 0;
111
112             /* Pozicija na koju treba smestiti
113              * procitani karakter */
114             /* Broj alociranih mesta */
```

```
112     pesme[i].izvodjac = NULL;    /* Memorija za smestanje procitanih
                                     karaktera */
114
116     /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
        izvodjaca) citaju se karakteri iz datoteke */
118     while ((c = fgetc(ulaz)) != ' ') {
120         /* Proverav da li postoji dovoljno memorije za smestanje
            procitanog karaktera */
122         if (j == alocirano) {
124             /* Ako ne, ako je potrosena sva alocirana memorija, alocira
                se novih KORAK mesta */
126             alocirano += KORAK;
128             pesme[i].izvodjac =
                (char *) realloc(pesme[i].izvodjac,
                                alocirano * sizeof(char));
130
132             /* Provera da li je nova alokacija uspesna */
134             if (pesme[i].izvodjac == NULL) {
136                 /* Ako nije oslobadja se sva memorija zauzeta do ovog
                    koraka */
138                 for (k = 0; k < i; k++) {
140                     free(pesme[k].izvodjac);
142                     free(pesme[k].naslov);
144                 }
146                 free(pesme);
148                 /* I prekida sa izvršavanjem programa */
150                 return 0;
152             }
154
156             /* Ako postoji dovoljno memorije, smestamo procitani karakter
            */
158             pesme[i].izvodjac[j] = c;
160             j++;
162             /* I nastavlja se sa citanjem */
164         }
166
168         /* Upis terminirajuće nule na kraj reci */
169         pesme[i].izvodjac[j] = '\\0';
171
172         /* Preskace se karakter - */
173         fgetc(ulaz);
175
176         /* Preskace se razmak */
177         fgetc(ulaz);
179
180         /* Cita se naslov pesme */
181         j = 0;
182         /* Pozicija na koju treba smestiti
            procitani karakter */
183         alocirano = 0;
184         /* Broj alociranih mesta */
185         pesme[i].naslov = NULL;
186         /* Memorija za smestanje procitanih
```



```
164                                     karaktera */
166 /* Sve do zareza (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 */
171     if (j == alocirano) {
172         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
173            se novih KORAK mesta */
174         alocirano += KORAK;
175         pesme[i].naslov =
176             (char *) realloc(pesme[i].naslov,
177                             alocirano * sizeof(char));
178
179         /* Provera da li je nova alokacija uspesna */
180         if (pesme[i].naslov == NULL) {
181             /* Ako nije, oslobadja se sva memorija zauzeta do ovog
182                koraka */
183             for (k = 0; k < i; k++) {
184                 free(pesme[k].izvodjac);
185                 free(pesme[k].naslov);
186             }
187             free(pesme[i].izvodjac);
188             free(pesme);
189
190             /* I prekida izvršavanje programa */
191             return 0;
192         }
193     }
194     /* Ako postoji dovoljno memorije, smesta se procitani karakter
195     */
196     pesme[i].naslov[j] = c;
197     j++;
198     /* I nastavlja dalje sa citanjem */
199 }
200 /* Upisuje se terminirajuca nula na kraj reci */
201 pesme[i].naslov[j] = '\0';
202
203 /* Preskace se razmak */
204 fgetc(ulaz);
205
206 /* Cita se broj gledanja */
207 broj_gledanja = 0;
208
209 /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
210    datoteke */
211 while ((c = fgetc(ulaz)) != '\n') {
212     broj_gledanja = broj_gledanja * 10 + (c - '0');
213 }
214 pesme[i].broj_gledanja = broj_gledanja;
```

```

214     /* Prelazi se na citanje sledece pesme */
215     i++;
216 }

218 /* Informacija o broju procitanih pesama */
219 n = i;
220 /* Zatvaranje nepotrebne datoteke */
221 fclose(ulaz);

222 /* Analiza argumenta komandne linije */
223 if (argc == 1) {
224     /* Nema dodatnih opcija => sortiranje po broju gledanja */
225     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
226 } else {
227     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
228         /* Sortiranje po naslovu */
229         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
230     } else {
231         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
232             /* Sortiranje po izvodjacu */
233             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
234         } else {
235             printf("Nedozvoljeni argumenti!\n");
236             free(pesme);
237             return 0;
238         }
239     }
240 }

242 /* Ispis rezultata */
243 for (i = 0; i < n; i++) {
244     printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
245           pesme[i].broj_gledanja);
246 }

248 /* Oslobadjanje memorije */
249 for (i = 0; i < n; i++) {
250     free(pesme[i].izvodjac);
251     free(pesme[i].naslov);
252 }
253 free(pesme);

254 return 0;
255 }

```

### Rešenje 3.28

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #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
        se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
13
14     /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
        oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */
15
16     int b1 = *((int *) a);
17     int b2 = *((int *) b);
18
19     if (b1 > b2)
20         return 1;
21     else if (b1 < b2)
22         return -1;
23     else
24         return 0;
25 }
26
27 int compare_int_desc(const void *a, const void *b)
28 {
29     /* Za obrnuti poredak treba samo oduzimati a od b */
30     /* return *((int *)b) - *((int *)a); */
31
32     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
        funkcija */
33     return -compare_int(a, b);
34 }
35
36
37 int main()
38 {
39     size_t n;
40     int i, x;
41     int a[MAX], *p = NULL;
42
43     /* Unos dimenzije */
44     printf("Uneti dimenziju niza: ");
45     scanf("%ld", &n);
46     if (n > MAX)
47         n = MAX;
48
49     /* Unos elementa niza */
50     printf("Uneti elemente niza:\n");
51     for (i = 0; i < n; i++)
52         scanf("%d", &a[i]);
53
54     /* Sortiranje niza celih brojeva */
55     qsort(a, n, sizeof(int), &compare_int);
```

```

57  /* Prikaz sortiranog niz */
59  printf("Sortirani niz u rastucem poretku:\n");
   for (i = 0; i < n; i++)
61      printf("%d ", a[i]);
   putchar('\n');
63
   /* 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");
   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)
20         return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22     br = 2;
23     for (i = 2; i < sqrt(x); i++)
24         if (x % i == 0)
25             /* Ako i deli x onda su delioci: i, x/i */
26             br += 2;
27     /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
28        promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29        jos jednog delioca */
30     if (i * i == x)
31         br++;
32
33     return br;
34 }
35
36 /* Funkcija poredjenja dva cela broja po broju delilaca */
37 int compare_no_deviders(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = no_of_deviders(ak);
42     int n_d_b = no_of_deviders(bk);
43
44     if (n_d_a > n_d_b)
45         return 1;
46     else if (n_d_a < n_d_b)
47         return -1;
48     else
49         return 0;
50 }
51
52 int main()
53 {
54     size_t n;
55     int i;
56     int a[MAX];
57
58     /* Unos dimenzije */
59     printf("Uneti dimenziju niza: ");
60     scanf("%ld", &n);
61     if (n > MAX)
62         n = MAX;
63
64     /* Unos elementa niza */
65     printf("Uneti elemente niza:\n");
66     for (i = 0; i < n; i++)
67         scanf("%d", &a[i]);
68
69     /* Sortiranje niza celih brojeva prema broju delilaca */
70     qsort(a, n, sizeof(int), &compare_no_deviders);
```

```

71      /* Prikaz sortiranog niza */
73      printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
75      for (i = 0; i < n; i++)
77          printf("%d ", a[i]);
79      putchar('\n');

      return 0;
}

```

### 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 veca 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 uporeduje
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

```

```

41 int main()
42 {
43     int i;
44     size_t n;
45     FILE *fp = NULL;
46     char niske[MAX_NISKI][MAX_DUZINA];
47     char *p = NULL;
48     char x[MAX_DUZINA];
49
50     /* Otvaranje datoteke */
51     if ((fp = fopen("niske.txt", "r")) == NULL) {
52         fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
53         exit(EXIT_FAILURE);
54     }
55
56     /* Citanje sadrzaja datoteke */
57     for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
58
59     /* Zatvaranje datoteke */
60     fclose(fp);
61     n = i;
62
63     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
64        prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
65        niske po duzini */
66     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);
67
68     printf("Leksikografski sortirane niske:\n");
69     for (i = 0; i < n; i++)
70         printf("%s ", niske[i]);
71     printf("\n");
72
73     /* Unos trazene niske */
74     printf("Uneti trazenu nisku: ");
75     scanf("%s", x);
76
77     /* Binarna pretraga */
78     /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
79        je niz vec sortiran leksikografski. */
80     p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
81                &poredi_leksikografski);
82
83     if (p != NULL)
84         printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
85                p, (p - (char *) niske) / MAX_DUZINA);
86     else
87         printf("Niska nije pronadjena u nizu\n");
88
89     /* Sortiranje po duzini */
90     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
91
92     printf("Niske sortirane po duzini:\n");

```

```

93     for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
        printf("\n");
95
96     /* Linearna pretraga */
97     p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
        &poredi_leksikografski);
99
100    if (p != NULL)
101        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
        p, (p - (char *) niske) / MAX_DUZINA);
103    else
104        printf("Niska nije pronadjena u nizu\n");
105
106    exit(EXIT_SUCCESS);
107 }

```

### Rešenje 3.31

```

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 pokazivaca na karaktere ovog potpisa
11   char *niske[3];
12   posle alokacije u main-u se moze graficki predstaviti ovako:
13
14   | X | -----> | a | b | c | \0|
15   -----
16   | Y | -----> | d | e | \0|
17   -----
18   | Z | -----> | f | g | h | \0|
19   -----
20
21   Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
22   njegov element pokazivac koji pokazuje na alocirane karaktere i-te
23   reci. Njegov tip je char*.
24
25   Kako pokazivaci a i b u sledecoj funkciji sadrže adrese elemenata
26   koji trebaju biti upoređeni (recimo adresu od X i adresu od Z), i
27   kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
28   moraju i kastovati. Da bi se leksikografski uporedili elementi X i
29   Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
30   u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
31   kastovani na odgovarajući tip.
32   *****/
33 int poredi_leksikografski(const void *a, const void *b)

```



```
33 {
34     return strcmp(*(char **) a, *(char **) b);
35 }

37 /* Funkcija slicna prethodnoj, osim sto elemente ne uporeduje
   leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
40 {
41     return strlen(*(char **) a) - strlen(*(char **) b);
42 }

43
44 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
45 element u nizu sa kojim se poredi, pa njega treba kastovati na
46 char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
47 ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
48 main funkciji je to x, koji je tipa char*, tako da pokazivac a
49 ovde samo treba kastovati i ne dereferencirati. */
50 int poredi_leksikografski_b(const void *a, const void *b)
51 {
52     return strcmp((char *) a, *(char **) b);
53 }

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

63     /* Otvaranje datoteke */
64     if ((fp = fopen("niske.txt", "r")) == NULL) {
65         fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
66         exit(EXIT_FAILURE);
67     }

68
69     /* Citanje sadrzaja datoteke */
70     i = 0;
71     while (fscanf(fp, "%s", x) != EOF) {
72         /* Alociranje dovoljne memorije za i-tu nisku */
73         if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
74             fprintf(stderr, "Greska pri alociranju niske\n");
75             exit(EXIT_FAILURE);
76         }
77         /* Kopiranje procitane niske na svoje mesto */
78         strcpy(niske[i], x);
79         i++;
80     }

81
82     /* Zatvaranje datoteke */
83     fclose(fp);
```

```

85     n = i;

87     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
88        prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
89        niske po duzini */
    qsort(niske, n, sizeof(char *), &poredi_leksikografski);

91     printf("Leksikografski sortirane niske:\n");
93     for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
95     printf("\n");

97     /* Unos trazene niske */
    printf("Uneti trazenu nisku: ");
99     scanf("%s", x);

101    /* Binarna pretraga */
    p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
103    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
104              *p, p - niske);
    else
107        printf("Niska nije pronadjena u nizu\n");

109    /* Linearna pretraga */
    p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
111    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
112              *p, p - niske);
    else
115        printf("Niska nije pronadjena u nizu\n");

117    /* Sortiramo po duzini */
    qsort(niske, n, sizeof(char *), &poredi_duzine);

119    printf("Niske sortirane po duzini:\n");
121    for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
123    printf("\n");

125    /* Oslobadjanje zauzete memorije */
    for (i = 0; i < n; i++)
127        free(niske[i]);

129    exit(EXIT_SUCCESS);
}

```

### Rešenje 3.32

```

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

```

```
4 #include <string.h>
5 #include <search.h>
6
7 #define MAX 500
8
9 /* Struktura sa svim informacijama o pojedinacnom studentu */
10 typedef struct {
11     char ime[21];
12     char prezime[21];
13     int bodovi;
14 } Student;
15
16 /* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
17 istim brojem bodova se dodatno sortiraju leksikografski po
18 prezimenu */
19 int poredi1(const void *a, const void *b)
20 {
21     Student *prvi = (Student *) a;
22     Student *drugi = (Student *) b;
23
24     if (prvi->bodovi > drugi->bodovi)
25         return -1;
26     else if (prvi->bodovi < drugi->bodovi)
27         return 1;
28     else
29         /* Ako su jednaki po broju bodova, treba ih uporediti po
30         prezimenu */
31         return strcmp(prvi->prezime, drugi->prezime);
32 }
33
34 /* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
35 Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
36 parametar je element niza ciji se bodovi porede. */
37 int poredi2(const void *a, const void *b)
38 {
39     int bodovi = *(int *) a;
40     Student *s = (Student *) b;
41     return s->bodovi - bodovi;
42 }
43
44 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
45 Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
46 parametar je element niza cije se prezime porede. */
47 int poredi3(const void *a, const void *b)
48 {
49     char *prezime = (char *) a;
50     Student *s = (Student *) b;
51     return strcmp(prezime, s->prezime);
52 }
53
54 int main(int argc, char *argv[])
55 {
```

```
Student kolokvijum[MAX];
56 int i;
size_t br_studenata = 0;
58 Student *nadjen = NULL;
FILE *fp = NULL;
60 int bodovi;
char prezime[21];
62

/* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64 informacija korisniku kako se program koristi i prekida se
izvrsavanje. */
66 if (argc < 2) {
    fprintf(stderr,
68         "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
        argv[0]);
70    exit(EXIT_FAILURE);
}
72

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

/* Ucitavanje sadrzaja */
80 for (i = 0;
        fscanf(fp, "%s%s%d", kolokvijum[i].ime,
82             kolokvijum[i].prezime,
            &kolokvijum[i].bodovi) != EOF; i++);
84

/* Zatvaranje datoteke */
86 fclose(fp);
br_studenata = i;
88

/* Sortiranje niza studenata po broju bodova, gde se unutar grupe
90 studenata sa istim brojem bodova sortiranje vrsi po prezimenu */
qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
92

printf("Studenti sortirani po broju poena opadajuće, ");
94 printf("pa po prezimenu rastuće:\n");
for (i = 0; i < br_studenata; i++)
96     printf("%s %s %d\n", kolokvijum[i].ime,
        kolokvijum[i].prezime, kolokvijum[i].bodovi);
98

/* Pretrazivanje studenata po broju bodova se vrsi binarnom
100 pretragom jer je niz sortiran po broju bodova. */
printf("Unesite broj bodova: ");
102 scanf("%d", &bodovi);

104 nadjen =
        bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
106             &poredi2);
```

```

108  if (nadjen != NULL)
    printf
110      ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
        nadjen->ime, nadjen->prezime, nadjen->bodovi);
112  else
    printf("Nema studenta sa unetim brojem bodova\n");
114
    /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116       sortiran po bodovima. */
    printf("Unesite prezime: ");
118    scanf("%s", prezime);

120    nadjen =
        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
122              &poredi3);

124    if (nadjen != NULL)
        printf
126          ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
            nadjen->ime, nadjen->prezime, nadjen->bodovi);
128    else
        printf("Nema studenta sa unetim prezimenom\n");
130
132    return 0;
}

```

### 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 */
    if (strlen(s) != strlen(t))
18         return 0;

20     /* Sortiranje niski */
    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 */
26  return !strcmp(s, t);
    }
28
    int main()
30  {
    char s[MAX], t[MAX];
32
    /* Unos niski */
34  printf("Unesite prvu nisku: ");
    scanf("%s", s);
36  printf("Unesite drugu nisku: ");
    scanf("%s", t);
38
    /* Ispituje se da li su niske anagrami */
40  if (anagrami(s, t))
        printf("jesu\n");
42  else
        printf("nisu\n");
44
    return 0;
46  }

```

### Rešenje 3.34

```

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

5  #define MAX 10
    #define MAX_DUZINA 32
7
    /* Funkcija porenjenja */
9  int uporedi_niske(const void *pa, const void *pb)
    {
11     return strcmp((char *) pa, (char *) pb);
    }
13
    int main()
15  {
    int i, n;
17     char S[MAX][MAX_DUZINA];

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

23     /* Unos niza niski */
    printf("Unesite niske:\n");

```

```

25     for (i = 0; i < n; i++)
26         scanf("%s", S[i]);
27
28     /* Sortiranje niza niski */
29     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
30
31     /******
32     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
33     sortiranih niski. Koriscen je samo u fazi testiranja programa.
34
35     printf("Sortirane niske su:\n");
36     for(i = 0; i < n; i++)
37         printf("%s ", S[i]);
38     *****/
39
40     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
41     niza biti jedna do druge */
42     for (i = 0; i < n - 1; i++)
43         if (strcmp(S[i], S[i + 1]) == 0) {
44             printf("ima\n");
45             return 0;
46         }
47
48     printf("nema\n");
49     return 0;
50 }

```

### Rešenje 3.35

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4
5  #define MAX 21
6
7  /* Struktura koja predstavlja jednog studenta */
8  typedef struct student {
9      char nalog[8];
10     char ime[MAX];
11     char prezime[MAX];
12     int poeni;
13 } Student;
14
15 /* Funkcija poredi studente prema broju poena, rastuce */
16 int uporedi_poeni(const void *a, const void *b)
17 {
18     Student s = *(Student *) a;
19     Student t = *(Student *) b;
20     return s.poeni - t.poeni;
21 }

```

```

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;
76     char nalog_trazeni[8];
77     Student niz_studenata[100];
78     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) {
84         fprintf(stderr,
85             "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
86         );
87         exit(EXIT_FAILURE);
88     }

89     /* Otvaranje datoteke za citanje */
90     in = fopen("studenti.txt", "r");
91     if (in == NULL) {
92         fprintf(stderr,
93             "Greska prilikom otvaranja datoteke studenti.txt!\n");
94         exit(EXIT_FAILURE);
95     }

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

105     /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
106     while (fscanf
107         (in, "%s %s %s %d", niz_studenata[i].nalog,
108          niz_studenata[i].ime, niz_studenata[i].prezime,
109          &niz_studenata[i].poeni) != EOF)
110         i++;

111     br_studenata = i;

113     /* Ako je prisutna opcija -p, vrši se sortiranje po poenima */
114     if (strcmp(argv[1], "-p") == 0)
115         qsort(niz_studenata, br_studenata, sizeof(Student),
116             &uporedi_poeni);
117     /* A ako je prisutna opcija -n, vrši se sortiranje po nalogu */
118     else if (strcmp(argv[1], "-n") == 0)
119         qsort(niz_studenata, br_studenata, sizeof(Student),
120             &uporedi_nalog);
121

123     /* Sortirani studenti se ispisuju u izlaznu datoteku */
}
```

```

125     for (i = 0; i < br_studenata; i++)
        fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
            niz_studenata[i].ime, niz_studenata[i].prezime,
127            niz_studenata[i].poeni);

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

133
        /* ... pronalazi se student sa tim nalogom... */
135        nadjen =
            (Student *) bsearch(nalog_trazeni, niz_studenata,
137                               br_studenata, sizeof(Student),
                               &uporedi_bsearch);

139
        if (nadjen == NULL)
141            printf("Nije nadjen!\n");
        else
143            printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
                    nadjen->prezime, nadjen->poeni);
145    }

147    /* Zatvaranje datoteka */
    fclose(in);
149    fclose(out);

151    return 0;
}

```

### Rešenje 3.37

```

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

4  /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
    standardnog ulaza */
6  void ucitaj_matricu(int **a, int n, int m)
    {
8      printf("Unesite elemente matrice po vrstama:\n");
        int i, j;

10
        for (i = 0; i < n; i++) {
12            for (j = 0; j < m; j++) {
                scanf("%d", &a[i][j]);
14            }
        }
16    }

18    /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
        kolona */

```

```
20 int zbir_vrste(int **a, int v, int m)
21 {
22     int i, zbir = 0;
23
24     for (i = 0; i < m; i++) {
25         zbir += a[v][i];
26     }
27     return zbir;
28 }
29
30 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
31    osnovu zbira koriscenjem selection sort algoritma */
32 void sortiraj_vrste(int **a, int n, int m)
33 {
34     int i, j, min;
35
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 }
51
52 /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
53    standardni izlaz */
54 void ispis_matricu(int **a, int n, int m)
55 {
56     int i, j;
57
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 }
65
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 *));
    if (a == NULL) {
74      fprintf(stderr, "Problem sa alokacijom memorije!\n");
        exit(EXIT_FAILURE);
76  }
    /* Za svaku vrstu ponaosob */
78  for (i = 0; i < n; i++) {
        /* Alocira se memorija */
80      a[i] = (int *) malloc(m * sizeof(int));
        /* Proverava se da li je doslo do greske prilikom alokacije */
82      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 */

```

```
124     sortiraj_vrstu(a, n, m);  
126     /* Ispis rezultujuće matrice */  
128     printf("Sortirana matrica je:\n");  
130     ispisi_matricu(a, n, m);  
132     /* Oslobađanje memorije */  
134     oslobodi_memoriju(a, n, m);  
136     return 0;  
138 }
```

## Glava 4

# Dinamičke strukture podataka

### 4.1 Liste

**Zadatak 4.1** Napisati biblioteku za rad sa jednostruko povezanom listom čiji čvorovi sadrže cele brojeve.

- (a) Definirati strukturu `Cvor` kojom se predstavlja čvor liste. Čvor treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.
- (b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje mu polja i vraća njegovu adresu.
- (c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.
- (d) Napisati funkciju `Cvor *pronadji_poslednji(Cvor * glava)` koja pronalazi poslednji čvor u listi.
- (e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste.
- (f) Napisati funkciju `Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.

- (g) Napisati funkciju `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 elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene zagradaama `[, ]` i međusobno razdvojene zaptetama.
- (j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.
- (k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.
- (l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.
- (n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. NAPOMENA: *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

*Primer 1*

```

INTERAKCIJA 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*

```

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [35, 14, 23]

Unesite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!

```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

*Primer 1*

```

INTERAKCIJA 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]

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 14, 17]

```

*Primer 2*

```

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [23, 14]
35
Lista: [23, 14, 35]

Unesite 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 se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*



*Primer 1*

```

INTERAKCIJA 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 trazi: 14
  Trazeni broj 14 je u listi!

  Unesite broj koji se brise: 3
  Lista nakon brisanja: [2, 5, 14]

```

*Primer 2*

```

INTERAKCIJA PROGRAMA:
  Unesite brojeve: (za kraj CTRL+D)
  23
  Lista: [23]
  14
  Lista: [14, 23]
  35
  Lista: [14, 23, 35]

  Unesite broj koji se trazi: 8
  Broj 8 se ne nalazi u listi!

  Unesite broj koji se brise: 3
  Lista nakon brisanja: [14, 23, 35]

```

[Rešenje 4.1]

**Zadatak 4.2** Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekursivno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati 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 briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave`.
- 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*

```
IZRAZ.TXT
{[23 + 5344] * (24 - 234)} - 23

IZLAZ:
Zagrade su ispravno uparene.
```

*Test 2*

```
IZRAZ.TXT
{[23 + 5] * (9 * 2)} - {23}

IZLAZ:
Zagrade su ispravno uparene.
```

*Test 3*

```
IZRAZ.TXT
{[2 + 54] / (24 * 87)} + (234 + 23)

IZLAZ:
Zagrade nisu ispravno uparene.
```

*Test 4*

```
IZRAZ.TXT
{(2 - 14) / (23 + 11)} * (2 + 13)

IZLAZ:
Zagrade nisu ispravno uparene.
```

*Test 5*

```
DATOTEKA IZRAZ.TXT JE PRAZNA

IZLAZ:
Zagrade su ispravno uparene.
```

*Test 6*

```
DATOTEKA IZRAZ.TXT NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja
datoteke izraz.txt!
```

[Rešenje 4.4]

**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>
</html>

IZLAZ:
  Etikete nisu pravilno uparene
  (nadjena je etiketa </html>, a poslednja
  otvorena je <body>)

```

## Test 5

```

Poziv: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ:
  Greska prilikom otvaranja
  datoteke datoteka.html.

```

## Test 6

```

Poziv: ./a.out datoteka.html

DATOTEKA.HTML JE PRAZNA

IZLAZ:
  Etikete su pravilno uparene!

```

[Rešenje 4.5]

**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. Postavlja mu se 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, nezvano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

*Primer 1*

```

INTERAKCIJA PROGRAMA:
Sluzbenik evidentira korisnicke zahteve:
Novi zahtev [CTRL+D za kraj]
JMBG: 1234567890123
Opis problema: Otvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 2345678901234
Opis problema: Podizanje novca

Novi zahtev [CTRL+D za kraj]
JMBG: 3456789012345
Opis problema: Reklamacija

Novi zahtev [CTRL+D za kraj]
JMBG:

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 2345678901234
i zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

IZVESTAJ.TXT
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 3456789012345 Zahtev: Reklamacija
JMBG: 1234567890123 Zahtev: Otvaranje racuna

```

[Rešenje 4.6]

**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 DATOTEKA.HTML 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, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku *da* ili *ne*, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

*Primer 1*

```

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

DATOTEKA STUDENTI.TXT 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, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

*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

DATOTEKA DAT2.TXT NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja datoteke
dat2.txt.

```

*Test 3*

```

Poziv: ./a.out dat1.txt dat2.txt

DATOTEKA DAT1.TXT JE PRAZNA

DAT2.TXT
5 6 11 12 14 16

IZLAZ:
[5, 6, 11, 12, 14, 16]

```

*Test 4*

```

Poziv: ./a.out dat1.txt

IZLAZ:
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 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

```
|| BROJEVI.TXT
||   43 12 15 16 4 2 8
||
|| IZLAZ:
|| REZULTAT.TXT
||   12 15 16
```

#### Test 2

```
|| DATOTEKA BROJEVI.TXT
|| NE POSTOJI.
||
|| IZLAZ:
|| REZULTAT.TXT
||   Greska prilikom otvaranja
||   datoteke brojevi.txt.
```

#### Test 3

```
|| DATOTEKA BROJEVI.TXT 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 sada počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se

nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

<i>Test 1</i> <pre>    ULAZ:      5 3       IZLAZ:     3 1 5 2 4 </pre>	<i>Test 2</i> <pre>    ULAZ:      8 4       IZLAZ:     4 8 5 2 1 3 7 6 </pre>	<i>Test 3</i> <pre>    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 sada 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.*

<i>Test 1</i> <pre>    ULAZ:      5 3       IZLAZ:     3 5 4 2 1 </pre>	<i>Test 2</i> <pre>    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 alokira 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 standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku **Nedostaje ime ulazne datoteke!**. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

*Test 1*

```

Poziv: ./a.out test.txt

TEST.TXT
Sunce utorak raCunar SUNCE programiranje
jabuka PROGramiranje sunCE JABUKa

IZLAZ:
jabuka: 2
programiranje: 2
racunar: 1
sunce: 3
utorak: 1

Najcesca rec: sunce (pojavljuje se 3 puta)

```

*Test 2*

```

Poziv: ./a.out suma.txt

SUMA.TXT
lipa zova hrast ZOVA breza LIPA

IZLAZ:
breza: 1
hrast: 1
lipa: 2
zova: 2

Najcesca rec: lipa
(pojavljuje se 2 puta)

```

*Test 3*

```

Poziv: ./a.out ulaz.txt

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
  Greska: Neuspesno otvaranje datoteke ulaz.txt.
\end{upotreba}
\end{miditest}
\begin{miditest}
\begin{test}{4}
Poziv: ./a.out

IZLAZ:
  Nedostaje ime ulazne datoteke!

```

[Rešenje 4.15]

**Zadatak 4.16** U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. **Pera Peric** 064/123-4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči **KRAJ**, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

*Primer 1*

```

IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858

INTERAKCIJA 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*

```

DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
  Unesite ime datoteke: imenik1.txt
  Greska: Neuspesno otvaranje datoteke
    imenik1.txt.
  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*

```

DATOTEKA PRIJEMNI.TXT NE POSTOJI

IZLAZ:
Greska: Neuspesno otvaranje datoteke
prijemni.txt.

```

[Rešenje 4.17]

\* **Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije. Svaka linija datoteke je formata **Ime Prezime DD.MM.** i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj unosa. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu DD.MM.. Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

*Primer 1*

```

Poziv: ./a.out rodjendani.txt

RODJENDANI.TXT
Marko Markovic 12.12.
Milan Jevremovic 04.06.
Maja Agic 23.04.
Nadica Zec 01.01.
Jovana Milic 05.05.

INTERAKCIJA 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: 20.12.
Slavljenik: Nadica Zec 01.01.
Unesite datum:

```

*Primer 2*

```

Poziv: ./a.out rodjendani.txt

DATOTEKA RODJENDANI.TXT NE POSTOJI

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

```

[Rešenje 4.18]

**Zadatak 4.19** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. 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 napisan u prvom zadatku se može iskoristiti kao mala biblioteka 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 standardni izlaz.

*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.

- Napisati funkciju koja izračunava broj čvorova stabla.
- Napisati funkciju koja izračunava broj listova stabla.
- Napisati funkciju koja štampa pozitivne vrednosti listova stabla.
- Napisati funkciju koja izračunava zbir čvorova stabla.
- Napisati funkciju koja izračunava najveći element stabla.
- Napisati funkciju koja izračunava dubinu stabla.
- Napisati funkciju koja izračunava broj čvorova na  $i$ -tom nivou stabla.
- Napisati funkciju koja ispisuje sve elemente na  $i$ -tom nivou stabla.

- (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.

<i>Test 1</i>	<i>Test 2</i>
<pre> Poziv: ./a.out 2 15  ULAZ: 10 5 15 3 2 4 30 12 14 13  IZLAZ: Broj cvorova: 10 Broj listova: 4 Pozitivni listovi: 2 4 13 30 Zbir cvorova: 108 Najveci element: 30 Dubina stabla: 5 Broj cvorova na 2. nivou: 3 Elementi na 2. nivou: 3 12 30 Maksimalni element na 2. nivou: 30 Zbir elemenata na 2. nivou: 45 Zbir elemenata manjih ili jednakih od 15: 78 </pre>	<pre> Poziv: ./a.out 3 31  ULAZ: 24 53 61 9 7 55 20 16  IZLAZ: Broj cvorova: 8 Broj listova: 3 Pozitivni listovi: 7 16 55 Zbir cvorova: 245 Najveci element: 61 Dubina stabla: 4 Broj cvorova na 3. nivou: 2 Elementi na 3. nivou: 16 55 Maksimalni element na 3. nivou: 55 Zbir elemenata na 3. nivou: 71 Zbir elemenata manjih ili jednakih od 31: 76 </pre>

[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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 10 5 15 3 2 4 30 12 14 13  IZLAZ: 0.nivo: 10 1.nivo: 5 15 2.nivo: 3 12 30 3.nivo: 2 4 14 4.nivo: 13 </pre>	<pre> ULAZ: 6 11 8 3 -2  IZLAZ: 0.nivo: 6 1.nivo: 3 11 2.nivo: -2 8 </pre>	<pre> ULAZ: 24 53 61 9 7 55 20 16  IZLAZ: 0.nivo: 24 1.nivo: 9 53 2.nivo: 7 20 61 3.nivo: 16 55 </pre>

[Rešenje 4.23]

\* **Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

*Primer 1*

```
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]

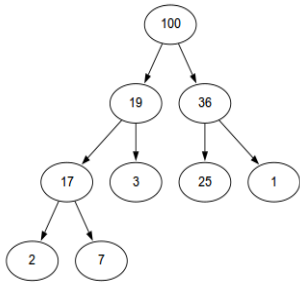
**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva *hip*. Napisati zatim i glavni program koji kreira stablo zadato slikom 4.1, poziva funkciju `heap` i ispisuje rezultat na standardni izlaz.

*Test 1*

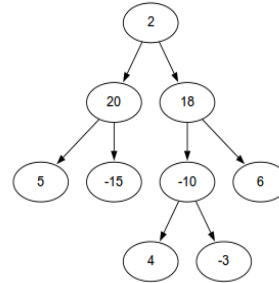
```
IZLAZ:
Zadato stablo je hip!
```

[Rešenje 4.26]





Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.
- Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

*Test 1*

```

|| IZLAZ:
|| Vrednost u cvoru sa maksimalnim desnim zbirom: 18
|| Vrednost u cvoru sa minimalnim levim zbirom: 18
|| 2 18 -10 4
|| 2 20 -15
  
```

## 4.3 Rešenja

Rešenje 4.1

```
2  #ifndef _LISTA_H
3  #define _LISTA_H
4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
5  typedef struct cvor {
6      int vrednost;
7      struct cvor *sledeci;
8  } Cvor;
9
10 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
11 dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
12 na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
13 Cvor *napravi_cvor(int broj);
14
15 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
16 ciji se pokazivac glava nalazi na adresi adresa_glave. */
17 void oslobodi_listu(Cvor ** adresa_glave);
18
19 /* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
20 greske pri alokaciji memorije, inace vraca 0. */
21 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
22
23 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
24 NULL ukoliko je lista prazna. */
25 Cvor *pronadji_poslednji(Cvor * glava);
26
27 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
28 pri alokaciji memorije, inace vraca 0. */
29 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
30
31 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
32 nov cvor sa vrednoscu broj. */
33 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
34
35 /* Funkcija uvezuje cvor novi iza postojeceg cvora tekuci. */
36 void dodaj_iza(Cvor * tekuci, Cvor * novi);
37
38 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
39 sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
40 inace vraca 0. */
41 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
42
43 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
44 Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
45 NULL u slucaju da takav cvor ne postoji u listi. */
46 Cvor *pretrazi_listu(Cvor * glava, int broj);
47
48 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
49 U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
50 neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
```

```

52     sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
    */
    Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
54
    /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivac na glavu liste, koji može biti promenjen u slucaju da se
    56     obrise stara glava. */
    void obrisi_cvor(Cvor ** adresa_glave, int broj);
58
    /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na činjenicu da je prosledjena lista sortirana
    60     neopadajuće. Azurira pokazivac na glavu liste, koji može biti
    62     promenjen ukoliko se obrise stara glava liste. */
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
64
    /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
    liste, razdvojene zarezima i uokvirene zagradama. */
    66     void ispisi_listu(Cvor * glava);
    68
    #endif
    70

```

```

#include <stdio.h>
2  #include <stdlib.h>
#include "lista.h"
4
Cvor *napravi_cvor(int broj)
6  {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    8     if (novi == NULL)
        return NULL;

    10     novi->vrednost = broj;
    novi->sledeci = NULL;
    12     return novi;
    14 }

void oslobodi_listu(Cvor ** adresa_glave)
16 {
    18     Cvor *pomocni = NULL;

    /* Ako lista nije prazna, onda treba osloboditi memoriju. */
    20     while (*adresa_glave != NULL) {
        /* Potrebno je prvo zapamtiti adresu sledećeg cvora i onda
        22         osloboditi cvor koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
        24         free(*adresa_glave);
        /* Sledeći cvor je nova glava liste. */
        26         *adresa_glave = pomocni;
    }
    28 }
}
30

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)

```

```
32 {
33     /* Kreira se nov cvor i proverava se da li je bilo greske pri
34        alokaciji. */
35     Cvor *novi = napravi_cvor(broj);
36     if (novi == NULL)
37         return 1;
38
39     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste. */
40     novi->sledeci = *adresa_glave;
41     *adresa_glave = novi;
42
43     return 0;
44 }
45
46 Cvor *pronadji_poslednji(Cvor * glava)
47 {
48     /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
49     if (glava == NULL)
50         return NULL;
51
52     /* Sve dok glava pokazuje na cvor koji ima sledeceg, pokazivac
53        glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
54        ce pokazivati na cvor liste koji nema sledeceg, tj. na poslednji
55        cvor liste i vraca se vrednost pokazivaca glava.
56
57        Pokazivac glava je argument funkcije i njegove promene nece se
58        odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
59     while (glava->sledeci != NULL)
60         glava = glava->sledeci;
61
62     return glava;
63 }
64
65 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
66 {
67     Cvor *novi = napravi_cvor(broj);
68     if (novi == NULL)
69         return 1;
70
71     /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
72        ujedno i cela lista. Azurira se vrednost na koju pokazuje
73        adresa_glave i tako se azurira i pokazivacka promenljiva u
74        pozivajucoj funkciji. */
75     if (*adresa_glave == NULL) {
76         *adresa_glave = novi;
77         return 0;
78     }
79
80     /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
81        se dodaje na kraj liste kao sledbenik poslednjeg. */
82     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
83     poslednji->sledeci = novi;
```

```
84     return 0;
85 }
86
87 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
88 {
89     /* U praznoj listi nema takvog mesta i vraca se NULL. */
90     if (glava == NULL)
91         return NULL;
92
93     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
94        pokazivala na cvor ciji je sledeci ili ne postoji ili ima
95        vrednost vecu ili jednaku vrednosti novog cvora.
96
97        Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
98        provera da li se doslo do poslednjeg cvora liste pre nego sto se
99        proveru vrednost u sledecem cvoru, jer u slucaju poslednjeg,
100        sledeci ne postoji, pa ni njegova vrednost. */
101     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
102         glava = glava->sledeci;
103
104     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
105        poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
106        vecu od broj. */
107     return glava;
108 }
109
110 void dodaj_iza(Cvor * tekuci, Cvor * novi)
111 {
112     /* Novi cvor se dodaje iza tekuceg cvora. */
113     novi->sledeci = tekuci->sledeci;
114     tekuci->sledeci = novi;
115 }
116
117 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
118 {
119     /* U slucaju prazne liste glava nove liste je novi cvor. Ukoliko je
120        doslo do greske pri alokaciji memorije cvraa se 1. */
121     if (*adresa_glave == NULL) {
122         Cvor *novi = napravi_cvor(broj);
123         if (novi == NULL)
124             return 1;
125         *adresa_glave = novi;
126         return 0;
127     }
128
129     /* Lista nije prazna. */
130     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
131        treba dodati na pocetak liste. */
132     if ((*adresa_glave)->vrednost >= broj) {
133         return dodaj_na_pocetak_liste(adresa_glave, broj);
134     }
135 }
```

```
136      /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
138      tada se pronalazi cvor liste iza koga treba uvezati nov cvor. */
Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
140      Cvor *novi = napravi_cvor(broj);
      if (novi == NULL)
142          return 1;

      /* Uvezuje se novi cvor iza pomocnog. */
      dodaj_iza(pomocni, novi);
146      return 0;
  }

148      Cvor *pretrazi_listu(Cvor * glava, int broj)
  {
150      for (; glava != NULL; glava = glava->sledeci)
152          if (glava->vrednost == broj)
              return glava;

154      /* Nema trazenog broja u listi i vraca se NULL. */
156      return NULL;
  }

158      Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
  {
160      /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
162      for (; glava != NULL && glava->vrednost <= broj;
              glava = glava->sledeci)
164          if (glava->vrednost == broj)
              return glava;

166      return NULL;
168  }

170      void obrisi_cvor(Cvor ** adresa_glave, int broj)
  {
172      Cvor *tekuci = NULL;
      Cvor *pomocni = NULL;
174

      /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
176      broju, i azurira se pokazivac na glavu liste. */
      while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
      {
178          /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
              adresi adresa_glave. */
180          pomocni = (*adresa_glave)->sledeci;
              free(*adresa_glave);
182          *adresa_glave = pomocni;
      }

184

      /* Ako je nakon toga lista ostala prazna, izlazi se iz funkcije. */
186      if (*adresa_glave == NULL)
```

```

188     return;
190     /* Od ovog trenutka, u svakoj iteraciji petlje tekuci pokazuje na
192     cvor cija vrednost je razlicita od trazenog broja. Isto vazi i
194     za sve cvorove levo od tekućeg. Poredi se vrednost sledećeg
    cvora (ako postoji) sa trazenim brojem. Cvor se brise ako je
    jednak, ili, ako je razlicit, prelazi se na sledeći cvor. Ovaj
    postupak se ponavlja dok se ne dodje do poslednjeg cvora. */
    tekuci = *adresa_glave;
196     while (tekuci->sledeci != NULL)
198         if (tekuci->sledeci->vrednost == broj) {
200             /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
                prvo cuva u pomocni. */
202             pomocni = tekuci->sledeci;
                /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
                njegovog trenutnog sledećeg. Njegov novi sledeci ce biti
                sledeci od cvora koji se brise. */
204             tekuci->sledeci = pomocni->sledeci;
                /* Sada treba osloboditi cvor sa vrednoscu broj. */
206             free(pomocni);
            } else {
208                /* Inace, ne treba brisati sledećeg od tekućeg i pokazivac se
                pomera na sledeci. */
210                tekuci = tekuci->sledeci;
            }
212     return;
214 }
216 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
218 {
    Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;

220     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
        broju i azurira se pokazivac na glavu liste. */
222     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
        {
224         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
            adresi adresa_glave. */
            pomocni = (*adresa_glave)->sledeci;
226             free(*adresa_glave);
            *adresa_glave = pomocni;
228         }

230     /* Ako je nakon toga lista ostala prazna, funkcija se prekida. Isto
        se radi i ukoliko glava liste sadrzi vrednost koja je veca od
232     broja, jer kako je lista sortirana rastuce nema potrebe broj
        traziti u repu liste. */
234     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;
236     /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje

```

```

238     na cvor cija vrednost je manja od trazenog broja, kao i svim
240     cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
242     je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
    ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
    cija vrednost je veca od trazenog broja. */
    tekuci = *adresa_glave;
244     while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
    )
        if (tekuci->sledeci->vrednost == broj) {
246            pomocni = tekuci->sledeci;
            tekuci->sledeci = tekuci->sledeci->sledeci;
248            free(pomocni);
        } else {
250            /* Ne treba brisati sledeceg od tekuceg jer je manji od
            trazenog i tekuci se pomera na sledeci cvor. */
252            tekuci = tekuci->sledeci;
        }
254     return;
    }
256
258 void ispisi_listu(Cvor * glava)
    {
260     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
        na glavu liste iz pozivajuce funkcije. */
262     putchar('[');
        for (; glava != NULL; glava = glava->sledeci) {
264         printf("%d", glava->vrednost);
            if (glava->sledeci != NULL)
266                 printf(", ");
        }
268     printf("]\n");
270 }

```

```

#include <stdio.h>
2  #include <stdlib.h>
#include "lista.h"
4
/* 1) Glavni program */
6 int main()
    {
8     /* Lista je prazna na pocetku. */
        Cvor *glava = NULL;
10     Cvor *trazeni = NULL;
        int broj;
12
        /* Testiranje dodavanja novog broja na pocetak liste */
14     printf("Unesite brojeve: (za kraj CTRL+D)\n");
        while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
            memorije za nov cvor. Memoriju alociranu za cvorove liste

```



```

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 }

28 printf("\nUnesite broj koji se trazi: ");
29 scanf("%d", &broj);

30 trazen = pretrazi_listu(glava, broj);
31 if (trazen == NULL)
32     printf("Broj %d se ne nalazi u listi!\n", broj);
33 else
34     printf("Trazeni broj %d je u listi!\n", trazen->vrednost);
35
36 oslobodi_listu(&glava);

38 return 0;
40 }

```

```

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

/* 2) Glavni program */
int main()
{
    Cvor *glava = NULL;
    int broj;

    /* Testiranje dodavanja novog broja na kraj liste. */
    printf("Unesite brojeve: (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
           memorije za nov cvor. Memoriju alociranu za cvorove liste
           treba osloboditi pre napustanja programa. */
        if (dodaj_na_kraj_liste(&glava, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
        }
        printf("\tLista: ");
        ispisi_listu(glava);
    }

    printf("\nUnesite broj koji se brise: ");
    scanf("%d", &broj);
}

```

```

30     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
        procitanom sa ulaza */
    obrisi_cvor(&glava, broj);
32
    printf("Lista nakon brisanja: ");
34    ispisi_listu(glava);
36    oslobodi_listu(&glava);
38
    return 0;
}

```

```

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

5  /* 3) Glavni program */
   int main()
7  {
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
    int broj;

11     /* Testira se dodavanje u listu tako da ona bude neopadajuće
        uredjena */
13     printf("Unosite brojeve (za kraj CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
15         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
            memorije za nov cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa. */
17         if (dodaj_sortirano(&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: ");
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("\nUnesite broj koji se brise: ");
    scanf("%d", &broj);
39
    /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju

```

```

41     procitanom sa ulaza */
    obrisi_cvor_sortirane_liste(&glava, broj);
43
    printf("Lista nakon brisanja: ");
45     ispisi_listu(glava);
47     oslobodi_listu(&glava);
49     return 0;
}

```

## Rešenje 4.2

```

#define _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

12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
14 Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);

20 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   bilo greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
26 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

28 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
   ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
   memorije, inace vraca 0. */
30 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
32

34 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
   NULL u slucaju da takav cvor ne postoji u listi. */
36 Cvor *pretrazi_listu(Cvor * glava, int broj);

38 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.

```

```

40     U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
    neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
    */
42     Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

44     /* Funkcija briše iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivac na glavu liste, koji može biti promjenjen u slučaju da se
    obriše stara glava liste. */
46     void obriši_cvor(Cvor ** adresa_glave, int broj);

48     /* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na cinjenicu da je prosledjena lista sortirana
    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
    promjenjen ukoliko se obriše stara glava liste. */
50     void obriši_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

52     /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
    zarezima. */
54     void ispisi_vrednosti(Cvor * glava);

56     /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
    liste, razdvojene zarezima i uokvirene zagradama. */
58     void ispisi_listu(Cvor * glava);

60     #endif
62

```

```

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

5  Cvor *napravi_cvor(int broj)
   {
7      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
      if (novi == NULL)
9          return NULL;

      novi->vrednost = broj;
      novi->sledeci = NULL;
11     return novi;
   }

13
15     void oslobodi_listu(Cvor ** adresa_glave)
17     {
        /* Lista je već prazna */
19         if (*adresa_glave == NULL)
            return;

21         /* Ako lista nije prazna, treba osloboditi memoriju. Pre
        oslobađanja memorije za glavu liste, treba osloboditi rep
        liste. */
23         oslobodi_listu(&(*adresa_glave)->sledeci);
25

```

```
27  /* Nakon oslobodjenog repa, oslobadja se glava liste, i azurira se
    glava u pozivajucoj funkciji tako da odgovara praznoj listi */
29  free(*adresa_glave);
    *adresa_glave = NULL;
31  }

33  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
34  {
    /* Kreira se nov cvor i proverava se da li je bilo greske pri
    alokaciji */
35  Cvor *novi = napravi_cvor(broj);
37  if (novi == NULL)
    return 1;

39  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
41  novi->sledeci = *adresa_glave;
    *adresa_glave = novi;
43  return 0;
45  }

47  int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
48  {
    if (*adresa_glave == NULL) {
49      /* Glava liste je upravo novi cvor i ujedno i cela lista. */
        Cvor *novi = napravi_cvor(broj);
51      /* Ukoliko je bilo greske pri alokaciji vraca se 1. */
        if (novi == NULL)
53          return 1;

55      /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
        azurira i pokazivacka promenljiva u pozivajucoj funkciji. */
57      *adresa_glave = novi;
        return 0;
59    }

61    /* Ako lista nije prazna, broj se dodaje u rep liste. */
    /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
63    nov broj se dodaje u rep liste u rekurzivnom pozivu. Informacija
    o uspesnosti alokacije u rekurzivnom pozivu funkcija prosledjuje
65    visem rekurzivnom pozivu koji tu informaciju vraca u rekurzivni
    poziv iznad, sve dok se ne vrati u main. Tek je iz main funkcije
67    moguće pristupiti pravom pocetku liste i osloboditi je celu, ako
    ima potrebe. Ako je funkcija vratila 0, onda nije bilo greske.
    */
69    return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
71  }

73  int dodaj_sortirano(Cvor ** adresa_glave, int broj)
74  {
    /* U slucaju prazne liste, glava nove liste je upravo novi cvor. */
75    if (*adresa_glave == NULL) {
        Cvor *novi = napravi_cvor(broj);
```

```
77     if (novi == NULL)
78         return 1;
79
80     *adresa_glave = novi;
81     return 0;
82 }
83
84 /* Lista nije prazna. Ako je broj manji ili jednak vrednosti u
85    glavi liste, onda se dodaje na pocetak liste i vraca se
86    informacija o uspesnosti alokacije. */
87 if ((*adresa_glave)->vrednost >= broj)
88     return dodaj_na_pocetak_liste(adresa_glave, broj);
89
90 /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
91    bude sortirana lista. */
92 return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
93 }
94
95 Cvor *pretrazi_listu(Cvor * glava, int broj)
96 {
97     /* U praznoj listi ga sigurno nema */
98     if (glava == NULL)
99         return NULL;
100
101     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava.
102        */
103     if (glava->vrednost == broj)
104         return glava;
105
106     /* Inace, pretraga se nastavlja u repu liste. */
107     return pretrazi_listu(glava->sledeci, broj);
108 }
109
110 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
111 {
112     /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
113        vrednosti u glavi liste, jer je lista neopadajuce sortirana. */
114     if (glava == NULL || glava->vrednost > broj)
115         return NULL;
116
117     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava.
118        */
119     if (glava->vrednost == broj)
120         return glava;
121
122     /* Inace, pretraga se nastavlja u repu. */
123     return pretrazi_listu(glava->sledeci, broj);
124 }
125
126 void obrisi_cvor(Cvor ** adresa_glave, int broj)
127 {
128     /* U praznoj listi, nema cvorova za brisanje. */
129 }
```

```

127     if (*adresa_glave == NULL)
128         return;
129
130     /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj. */
131     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
132
133     /* Preostaje provera da li glavu liste treba obrisati. */
134     if ((*adresa_glave)->vrednost == broj) {
135         /* pomocni pokazuje na cvor koji treba da se obrise. */
136         Cvor *pomocni = *adresa_glave;
137         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
138            brise se cvor koji je bio glava liste. */
139         *adresa_glave = (*adresa_glave)->sledeci;
140         free(pomocni);
141     }
142 }
143
144 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
145 {
146     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
147        od broja, kako je lista sortirana rastuce nema potrebe broj
148        traziti u repu liste i zato se funkcija prekida. */
149     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
150         return;
151
152     /* Brisu se cvorovi iz repa koji imaju vrednost broj. */
153     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
154
155     /* Preostaje provera da li glavu liste treba obrisati. */
156     if ((*adresa_glave)->vrednost == broj) {
157         /* pomocni pokazuje na cvor koji treba da se obrise. */
158         Cvor *pomocni = *adresa_glave;
159         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
160            brise se cvor koji je bio glava liste. */
161         *adresa_glave = (*adresa_glave)->sledeci;
162         free(pomocni);
163     }
164 }
165
166 void ispisi_vrednosti(Cvor * glava)
167 {
168     /* Prazna lista */
169     if (glava == NULL)
170         return;
171
172     /* Ispisuje se vrednost u glavi liste. */
173     printf("%d", glava->vrednost);
174
175     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
176        se poziva ista funkcija za ispis ostalih. */
177     if (glava->sledeci != NULL) {
178         printf(", ");
179     }
180 }

```

```

179     ispisi_vrednosti(glava->sledeci);
180 }
181 }
182
183 void ispisi_listu(Cvor * glava)
184 {
185     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
186        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
187        na glavu liste iz pozivajuće funkcije. Ona ispisuje samo
188        zgrade, a rekurzivno ispisivanje vrednosti u listi prepusta
189        rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
190        elemente razdvojene zapetom i razmakom. */
191     putchar('[');
192     ispisi_vrednosti(glava);
193     printf("]\n");
194 }

```

### Rešenje 4.3

```

1 #ifndef _LISTA_H
2 #define _LISTA_H
3
4 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
5    vrednost i pokazivace na sledeci i prethodni cvor liste. */
6 typedef struct cvor {
7     int vrednost;
8     struct cvor *sledeci;
9     struct cvor *prethodni;
10 } Cvor;
11
12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
13    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
15 Cvor *napravi_cvor(int broj);
16
17 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
18    ciji se pocetni cvor nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave);
20
21 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
22    bilo greske pri alokaciji memorije, inace vraca 0. */
23 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
24
25 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
26    NULL ukoliko je lista prazna. */
27 Cvor *pronadji_poslednji(Cvor * glava);
28
29 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
30    pri alokaciji memorije, inace vraca 0. */
31 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
32

```



```

34  /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
    nov cvor sa vrednoscu broj. */
    Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
36
    /* Funkcija uvezuje cvor novi iza postojeceg cvora tekuci. */
38  void dodaj_iza(Cvor * tekuci, Cvor * novi);

40  /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
    inace vraca 0. */
42  int dodaj_sortirano(Cvor ** adresa_glave, int broj);
44
    /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
    NULL u slucaju da takav cvor ne postoji u listi. */
46  Cvor *pretrazi_listu(Cvor * glava, int broj);
48

    /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
    neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
    sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
    */
50  Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
52

    /* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
    pokazivac glava se nalazi na adresi adresa_glave. */
54  void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci);
56

    /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivac na glavu liste, koji može biti promenjen u slucaju da se
    obrise stara glava. */
60  void obrisi_cvor(Cvor ** adresa_glave, int broj);
62

    /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na cinjenicu da je prosledjena lista neopadajuce
    sortirana. Azurira pokazivac na glavu liste, koji može biti
    promenjen ukoliko se obrise stara glava liste. */
64  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
66

    /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
    liste, razdvojene zapetama i uokvirene zagradama. */
68  void ispisi_listu(Cvor * glava);
70

    /* Funkcija prikazuje cvorove liste pocev od kraja ka glavi liste. */
72  void ispisi_listu_unazad(Cvor * glava);
74

76  #endif

#include <stdio.h>
2  #include <stdlib.h>
#include "lista.h"
4

```

```

6  Cvor *napravi_cvor(int broj)
{
8      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
      if (novi == NULL)
          return NULL;

10     novi->vrednost = broj;
12     novi->sledeci = NULL;
      return novi;
14 }

16 void oslobodi_listu(Cvor ** adresa_glave)
{
18     Cvor *pomocni = NULL;

20     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
      while (*adresa_glave != NULL) {
22         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
           osloboditi memoriju cvora koji predstavlja glavu liste. */
24         pomocni = (*adresa_glave)->sledeci;
           free(*adresa_glave);
26         /* Sledeci cvor je nova glava liste. */
           *adresa_glave = pomocni;
28     }
}

30
32 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
      Cvor *novi = napravi_cvor(broj);
34     if (novi == NULL)
          return 1;

36     /* Sledbenik novog cvora je glava stare liste */
38     novi->sledeci = *adresa_glave;
      /* Ako stara lista nije bila prazna, onda prethodni od glave treba
         da bude nov cvor. */
40     if (*adresa_glave != NULL)
42         (*adresa_glave)->prethodni = novi;
      /* Novi cvor je nova glava liste. */
44     *adresa_glave = novi;

46     return 0;
}

48
Cvor *pronadji_poslednji(Cvor * glava)
50 {
      /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
52     if (glava == NULL)
          return NULL;

54     /* Sve dok glava pokazuje na cvor koji ima sledeceg, pokazivac
56     glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava

```

```

58     ce pokazivati na cvor liste koji nema sledeceg, tj. na poslednji
    cvor liste i vraća se vrednost pokazivaca glava.

60     Pokazivac glava je argument funkcije i njegove promene neće se
    odraziti na vrednost pokazivaca glava u pozivajućoj funkciji. */
62     while (glava->sledeci != NULL)
        glava = glava->sledeci;
64
66     return glava;
67 }

68 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
69 {
70     Cvor *novi = napravi_cvor(broj);
71     if (novi == NULL)
72         return 1;
73
74     /* U slučaju prazne liste, glava nove liste je upravo novi cvor i
    ujedno i cela lista. Azurira se vrednost na koju pokazuje
76     adresa_glave i tako se azurira i pokazivačka promenljiva u
    pozivajućoj funkciji. */
78     if (*adresa_glave == NULL) {
        *adresa_glave = novi;
80         return 0;
81     }

82     /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
    se dodaje na kraj liste kao sledbenik poslednjeg. */
84     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
86     novi->prethodni = poslednji;
88
89     return 0;
90 }

92 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
93 {
94     /* U praznoj listi nema takvog mesta i vraća se NULL. */
95     if (glava == NULL)
96         return NULL;
97
98     /* Pokazivac glava se pomera na sledeći cvor sve dok ne bude
    pokazivala na cvor čiji je sledeći ili ne postoji ili ima
100     vrednost veću ili jednaku vrednosti novog cvora.

102     Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
    provera da li se doslo do poslednjeg cvora liste pre nego što se
104     proverí vrednost u sledecem cvoru, jer u slučaju poslednjeg,
    sledeći ne postoji, pa ni njegova vrednost. */
106     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
        glava = glava->sledeci;
108

```

```
110  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
    poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
    vecu od broj. */
112  return glava;
    }
114
void dodaj_iza(Cvor * tekuci, Cvor * novi)
116 {
    novi->sledeci = tekuci->sledeci;
118    novi->prethodni = tekuci;

120    /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
    a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
    na ispravne adrese. */
122    if (tekuci->sledeci != NULL)
124        tekuci->sledeci->prethodni = novi;
    tekuci->sledeci = novi;
126 }

128 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
    {
130    /* Ako je lista prazna, glava nove liste je novi cvor. */
    if (*adresa_glave == NULL) {
132        Cvor *novi = napravi_cvor(broj);
        if (novi == NULL)
134            return 1;
        *adresa_glave = novi;
136        return 0;
    }

138
140    /* Ukoliko je vrednost glave liste veka ili jednaka od nove
    vrednosti onda nov cvor treba staviti na pocetak liste. */
    if ((*adresa_glave)->vrednost >= broj) {
142        dodaj_na_pocetak_liste(adresa_glave, broj);
        return 0;
144    }

146    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
148        return 1;

150    /* Pronazi se cvor iza koga treba uvezati nov cvor. */
    Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
152    dodaj_iza(pomocni, novi);

154    return 0;
    }
156

Cvor *pretrazi_listu(Cvor * glava, int broj)
158 {
    for (; glava != NULL; glava = glava->sledeci)
160        if (glava->vrednost == broj)
```

```
        return glava;
162
    /* Nema trazenog broja u listi i vraca se NULL. */
164    return NULL;
    }
166
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
168 {
    /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
170    for (; glava != NULL && glava->vrednost <= broj;
        glava = glava->sledeci)
172        if (glava->vrednost == broj)
            return glava;
174
    /* Nema trazenog broja u listi i bice vrateno NULL. */
176    return NULL;
    }
178
/* Kod dvostruko povezane liste brisanje cvora na koji pokazuje
180 tekuci moze se lako uraditi jer sadrzi pokazivace na svog
sledbenika i prethodnika u listi. */
182 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
    {
184        /* Ako je tekuci NULL pokazivac, nema sta da se brise. */
        if (tekuci == NULL)
186            return;

188        /* Ako postoji prethodnik od tekuceg, onda se postavlja da njegov
sledeci bude sledeci od tekuceg. */
190        if (tekuci->prethodni != NULL)
            tekuci->prethodni->sledeci = tekuci->sledeci;
192

194        /* Ako postoji sledbenik tekuceg, onda njegov prethodnik treba da
bude prethodnik tekuceg. */
        if (tekuci->sledeci != NULL)
196            tekuci->sledeci->prethodni = tekuci->prethodni;

198        /* Ako je glava cvor koji se brise, nova glava liste bice sledbenik
stare glave. */
200        if (tekuci == *adresa_glave)
            *adresa_glave = tekuci->sledeci;
202

204        /* Oslobadja se dinamicki alociran prostor za cvor tekuci. */
        free(tekuci);
    }
206

void obrisi_cvor(Cvor ** adresa_glave, int broj)
208 {
    Cvor *tekuci = *adresa_glave;
210

    while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
212        obrisi_tekuci(adresa_glave, tekuci);
}
```

```

214 }
215
216 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
217 {
218     Cvor *tekuci = *adresa_glave;
219
220     while ((tekuci =
221             pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
222         obrisi_tekuci(adresa_glave, tekuci);
223 }
224
225 void ispisi_listu(Cvor * glava)
226 {
227     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
228        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
229        na glavu liste iz pozivajuće funkcije. */
230     putchar('[');
231     for (; glava != NULL; glava = glava->sledeci) {
232         printf("%d", glava->vrednost);
233         if (glava->sledeci != NULL)
234             printf(", ");
235     }
236     printf("]\n");
237 }
238
239 void ispisi_listu_unazad(Cvor * glava)
240 {
241     putchar('[');
242     if (glava == NULL) {
243         printf("]\n");
244         return;
245     }
246
247     glava = pronadji_poslednji(glava);
248
249     for (; glava != NULL; glava = glava->prethodni) {
250         printf("%d", glava->vrednost);
251         if (glava->prethodni != NULL)
252             printf(", ");
253     }
254     printf("]\n");
255 }
256
257 #include <stdio.h>
258 #include <stdlib.h>
259 #include "lista.h"
260
261 /* 1) Glavni program */
262 int main()
263 {
264     /* 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) {
   /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
15     memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */

```

```

17     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
18         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
19         oslobodi_listu(&glava);
20         exit(EXIT_FAILURE);
21     }
22     printf("\tLista: ");
23     ispisi_listu(glava);
24 }
25
26 printf("\nUnesite broj koji se brise iz liste: ");
27 scanf("%d", &broj);
28
29 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
30    procitanom sa ulaza. */
31 obrisi_cvor(&glava, broj);
32
33 printf("Lista nakon brisanja: ");
34 ispisi_listu(glava);
35
36 printf("\nLista ispisana u nazad: ");
37 ispisi_listu_unazad(glava);
38
39 oslobodi_listu(&glava);
40
41 return 0;
42 }

```

```

#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"
4
/* 3) Glavni program */
6 int main()
{
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 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);
23        }
24        printf("\tLista: ");
25        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

```

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

3

5  typedef struct cvor {
        char zagrada;
        struct cvor *sledeci;
7  } Cvor;

9  int main()
    {
11     Cvor *stek = NULL;
        FILE *ulaz = NULL;
13     char c;
        Cvor *pomocni = NULL;

15     ulaz = fopen("izraz.txt", "r");
17     if (ulaz == NULL) {
        fprintf(stderr,
19         "Greska prilikom otvaranja datoteke izraz.txt!\n");
        exit(EXIT_FAILURE);
    }

```

```

21     }
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     /* Procitana je cela datoteka. Zatvaramo je. */
56     fclose(ulaz);
57
58     /* Ako je stek prazan i procitana je cela datoteka, zagrade su
59        ispravno uparene, u suprotnom, nisu. */
60     if (stek == NULL && c == EOF)
61         printf("Zagrade su ispravno uparene.\n");
62     else {
63         printf("Zagrade nisu ispravno uparene.\n");
64         /* U slucaju neispravnog uparivanja treba osloboditi memoriju
65            koja je ostala zauzeta stekom. */
66         while (stek != NULL) {
67             pomocni = stek->sledeci;
68             free(stek);
69             stek = pomocni;
70         }
71     }

```

```

    return 0;
73 }

```

### Rešenje 4.5

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 #define MAX 100
7
8 #define OTVORENA 1
9 #define ZATVORENA 2
10
11 #define VAN_ETIKETE 0
12 #define PROCITANO_MANJE 1
13 #define U_ETIKETI 2
14
15 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
16    pokazivac na sledeci cvor. */
17 typedef struct cvor {
18     char etiketa[MAX];
19     struct cvor *sledeci;
20 } Cvor;
21
22 /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
23    adresu ili NULL ako alokacija nije bila uspesna. */
24 Cvor *napravi_cvor(char *etiketa)
25 {
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;
29
30     /* Inicijalizacija polja u novom cvoru */
31     if (strlen(etiketa) >= MAX) {
32         fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
33         etiketa[MAX - 1] = '\0';
34     }
35     strcpy(novi->etiketa, etiketa);
36     novi->sledeci = NULL;
37     return novi;
38 }
39
40 /* Funkcija oslobadja memoriju zauzetu stekom. */
41 void oslobodi_stek(Cvor ** adresa_vrha)
42 {
43     Cvor *pomocni;
44     while (*adresa_vrha != NULL) {
45         pomocni = *adresa_vrha;
46         *adresa_vrha = (*adresa_vrha)->sledeci;

```

```

    free(pomocni);
48 }
}
50
/* Funkcija proverava uspesnost alokacije memorije za cvor novi i
52 ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
zauzeta memorija za listu ciji se pokazivac vrh nalazi na adresi
54 adresa_vrha. */
void prover_i_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
56 {
    if (novi == NULL) {
58         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_stek(adresa_vrha);
60         exit(EXIT_FAILURE);
    }
62 }

/* Funkcija postavlja na vrh steka novu etiketu. */
void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
64 {
    Cvor *novi = napravi_cvor(etiketa);
66     prover_i_alokaciju(adresa_vrha, novi);
    novi->sledeci = *adresa_vrha;
70     *adresa_vrha = novi;
}

/* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
72 pokazivac razlicit od NULL, tada u niz karaktera na koji on
74 pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
76 samim tim nije bilo moguće skinuti vrednost sa steka) ili 1 u
suprotnom. */
78 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
80 {
    Cvor *pomocni;
82
    /* Pokusaj skidanja vrednost sa vrha praznog steka rezultuje
84 greskom i vraca se 0. */
    if (*adresa_vrha == NULL)
86         return 0;

    /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
88 adresu kopira etiketa sa vrha steka. */
    if (etiketa != NULL)
90         strcpy(etiketa, (*adresa_vrha)->etiketa);

    /* Element sa vrha steka se uklanja. */
92     pomocni = *adresa_vrha;
    *adresa_vrha = (*adresa_vrha)->sledeci;
94     free(pomocni);
96
98     return 1;

```

```
100 }
101
102 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
103    steka. Ukoliko je stek prazan, vraca NULL. */
104 char *vrh_steka(Cvor * vrh)
105 {
106     if (vrh == NULL)
107         return NULL;
108     return vrh->etiketa;
109 }
110
111 /* Funkcija prikazuje stek pocev od vrha prema dnu. */
112 void prikazi_stek(Cvor * vrh)
113 {
114     for (; vrh != NULL; vrh = vrh->sledeci)
115         printf("<%s>\n", vrh->etiketa);
116 }
117
118 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
119    etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
120    Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
121    procita etiketa. Vraca OTVORENA, ako je procitana otvorena
122    etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa. */
123 int uzmi_etiketu(FILE * f, char *etiketa)
124 {
125     int c;
126     int i = 0;
127     /* Stanje predstavlja informaciju dokle se stalo sa citanjem
128        etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
129        nije zapoceto citanje. */
130     /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
131        OTVORENA ili ZATVORENA. */
132     int stanje = VAN_ETIKETE;
133     int tip;
134
135     /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
136        to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
137        samo malim slovima. */
138     while ((c = fgetc(f)) != EOF) {
139         switch (stanje) {
140             case VAN_ETIKETE:
141                 if (c == '<')
142                     stanje = PROCITANO_MANJE;
143                 break;
144             case PROCITANO_MANJE:
145                 if (c == '/') {
146                     /* Cita se zatvorena etiketa. */
147                     tip = ZATVORENA;
148                 } else {
149                     if (isalpha(c)) {
150                         /* Cita se otvorena etiketa */
151                         tip = OTVORENA;
```

```

152     etiketa[i++] = tolower(c);
153 }
154 /* Od sada se cita etiketa i zato se menja stanje. */
155 stanje = U_ETIKETI;
156 break;
157 case U_ETIKETI:
158     if (isalpha(c) && i < MAX - 1) {
159         /* Ako je procitani karakter slovo i nije premasena
160            dozvoljena duzina etikete, procitani karakter se smanjuje
161            i smesta u etiketu. */
162         etiketa[i++] = tolower(c);
163     } else {
164         /* Inace, staje se sa citanjem etikete. Korektno se završava
165            niska koja sadrzi procitanu etiketu i vraća se njen tip.
166         */
167         etiketa[i] = '\0';
168         return tip;
169     }
170     break;
171 }
172 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
173    etiketa i vraća se EOF. */
174 return EOF;
175 }
176
177 int main(int argc, char **argv)
178 {
179     /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
180        nije procitana. */
181     Cvor *vrh = NULL;
182     char etiketa[MAX];
183     int tip;
184     int uparene = 1;
185     FILE *f = NULL;
186
187     /* Ime datoteke se preuzima iz komandne linije. */
188     if (argc < 2) {
189         fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
190         exit(0);
191     }
192
193     /* Datoteka se otvara za citanje. */
194     if ((f = fopen(argv[1], "r")) == NULL) {
195         fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
196             argv[1]);
197         exit(1);
198     }
199
200     /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
201     while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {

```

```

202  /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
204  etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
206  potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
208  koje nemaju sadrzaj (npr link). Zbog jednostavnosti
210  pretpostavlja se da njih nema u HTML dokumentu. */
212  if (tip == OTVORENA) {
214      if (strcmp(etiketa, "br") != 0
216          && strcmp(etiketa, "hr") != 0
218          && strcmp(etiketa, "meta") != 0)
220          potisni_na_stek(&vrh, etiketa);
222  }
224  /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
226  u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
228  uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
230  je taj uslov ispunjen, skida se sa steka, jer je upravo
232  zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
234  nisu pravilno uparene. */
236  else if (tip == ZATVORENA) {
238      if (vrh_steka(vrh) != NULL
240          && strcmp(vrh_steka(vrh), etiketa) == 0)
242          skini_sa_steka(&vrh, NULL);
244      else {
246          printf("Etikete nisu pravilno uparene\n");
248          printf("(nadjena je etiketa </%s>", etiketa);
250          if (vrh_steka(vrh) != NULL)
252              printf(", a poslednja otvorena je </%s>)\n", vrh_steka(vrh));
254          ;
256          else
258              printf(" koja nije otvorena)\n");
260          uparene = 0;
262          break;
264      }
266  }
268  }
270  /* Zavrsono je citanje datoteke i zatvara se. */
272  fclose(f);
274
276  /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
278  da bude prazan. Ukoliko nije, tada postoje etikete koje su
280  ostale otvorene. */
282  if (uparene) {
284      if (vrh_steka(vrh) == NULL)
286          printf("Etikete su pravilno uparene!\n");
288      else {
290          printf("Etikete nisu pravilno uparene\n");
292          printf("(etiketa </%s> nije zatvorena)\n", vrh_steka(vrh));
294          /* Oslobadja se memorija zauzeta stekom. */
296          oslobodi_stek(&vrh);
298      }
300  }
302  return 0;
304 }

```

## Rešenje 4.6

```

1  #ifndef _RED_H
2  #define _RED_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define MAX 1000
8  #define JMBG_DUZINA 14
9
10 /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11    opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
18    korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
20     Zahtev nalog;
21     struct cvor *sledeci;
22 } Cvor;
23
24 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
25    poslate adrese i vraća adresu novog cvora ili NULL ako je doslo do
26    greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
27    treba smestiti u nov cvor zbog smestanja manjeg podatka na
28    sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
29    bajtovima(B) u odnosu na strukturu Zahtev. */
30 Cvor *napravi_cvor(Zahtev * zahtev);
31
32 /* Funkcija prazni red, oslobadjajuci memoriju koji je red zauzeo. */
33 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
34
35 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
36    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
37    zauzeta memorija za listu ciji se pokazivac pocetak se nalazi na
38    adresi adresa_pocetka i prekida program. */
39 void proveri_alokaciju(Cvor ** adresa_pocetka,
40                        Cvor ** adresa_kraja, Cvor * novi);
41
42 /* Funkcija dodaje na kraj reda novi zahtev. */
43 void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
44                 Zahtev * zahtev);
45
46 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
47    pokazivac razlicit od NULL, tada se u strukturu na koju on
48    pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
49    suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
    suprotnom. */

```



```

51 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
    Zahtev * zahtev);
53
54 /* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev korisnika
55    na pocetku reda. Ukoliko je red prazan, vraca NULL. */
56 Zahtev *pocetak_reda(Cvor * pocetak);
57
58 /* Funkcija prikazuje sadrzaj reda. */
59 void prikazi_red(Cvor * pocetak);
61 #endif

```

```

1  #include "red.h"
3  Cvor *napravi_cvor(Zahtev * zahtev)
4  {
5      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
6      if (novi == NULL)
7          return NULL;
8
9      novi->nalog = *zahtev;
10     novi->sledeci = NULL;
11     return novi;
12 }
13
14 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
15 {
16     Cvor *pomocni = NULL;
17
18     while (*pocetak != NULL) {
19         pomocni = *pocetak;
20         *pocetak = (*pocetak)->sledeci;
21         free(pomocni);
22     }
23     *kraj = NULL;
24 }
25
26 void prover_i_alokaciju(Cvor ** adresa_pocetka,
27                        Cvor ** adresa_kraja, Cvor * novi)
28 {
29     if (novi == NULL) {
30         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
31         oslobodi_red(adresa_pocetka, adresa_kraja);
32         exit(EXIT_FAILURE);
33     }
34 }
35
36 void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
37                Zahtev * zahtev)
38 {
39     Cvor *novi = napravi_cvor(zahtev);
40     prover_i_alokaciju(adresa_pocetka, adresa_kraja, novi);

```

```

41  /* U red se uvek dodaje na kraj, ali zbog postojanja pokazivaca na
43  kraj, dodavanje na kraj je podjednako efikasno kao dodavanje na
    pocetak. */
45  if (*adresa_kraja != NULL) {
47      (*adresa_kraja)->sledeci = novi;
      *adresa_kraja = novi;
    } else {
49      /* Ako je red bio ranije prazan */
      *adresa_pocetka = novi;
51      *adresa_kraja = novi;
    }
53 }

55 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
    Zahtev * zahtev)
57 {
    Cvor *pomocni = NULL;
59
61     if (*adresa_pocetka == NULL)
        return 0;

63     if (zahtev != NULL)
        *zahtev = (*adresa_pocetka)->nalog;

65     pomocni = *adresa_pocetka;
67     *adresa_pocetka = (*adresa_pocetka)->sledeci;
    free(pomocni);

69     if (*adresa_pocetka == NULL)
71         *adresa_kraja = NULL;

73     return 1;
    }

75 Zahtev *pocetak_reda(Cvor * pocetak)
77 {
79     if (pocetak == NULL)
        return NULL;

81     return &(pocetak->nalog);
    }

83 void prikazi_red(Cvor * pocetak)
85 {
87     for (; pocetak != NULL; pocetak = pocetak->sledeci)
        printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);

89     printf("\n");
    }

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "red.h"
5
6 #define VREME_ZA_PAUZU 5
7
8 int main(int argc, char **argv)
9 {
10     /* Red je prazan. */
11     Cvor *pocetak = NULL, *kraj = NULL;
12     Zahtev nov_zahtev;
13     Zahtev *sledeci = NULL;
14     char odgovor[3];
15     int broj_usluzenih = 0;
16     FILE *izlaz = fopen("izvestaj.txt", "a");
17
18     if (izlaz == NULL) {
19         fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
20         exit(EXIT_FAILURE);
21     }
22
23     /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog JMBG
24        broja i opisa potrebne usluge. */
25     printf("Sluzbenik evidentira korisnicke zahteve:\n");
26
27     /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
28        JMBG broja procitao i da bi fgets nakon toga procitala ispravan
29        red sa opisom zahteva. */
30     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
31     while (scanf("%s", nov_zahtev.jmbg) != EOF) {
32         getchar();
33         printf("\tOpis problema: ");
34         fgets(nov_zahtev.opis, MAX - 1, stdin);
35         /* Ako je poslednji karakter nov red, eliminise se. */
36         if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
37             nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
38         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
39         printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
40     }
41
42     /* Datoteka vise nije potrebna i treba je zatvoriti. */
43     fclose(izlaz);
44
45     /* Dokle god ima korisnika u redu, treba ih usluziti. */
46     while (1) {
47         sledeci = pocetak_reda(pocetak);
48         /* Ako nema nikog vise u redu, prekida se petlja. */
49         if (sledeci == NULL)
50             break;
51
52         printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);

```

```

54     printf("i zahtevom: %s\n", sledeci->opis);
56     skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
58     broj_usluzenih++;
60     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
62     scanf("%s", odgovor);
64     if (strcmp(odgovor, "Da") == 0)
        dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
66     else
        fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
            nov_zahtev.opis);
68     if (broj_usluzenih == VREME_ZA_PAUZU) {
        printf("\nDa li je kraj smene? [Da/Ne] ");
70         scanf("%s", odgovor);
72         if (strcmp(odgovor, "Da") == 0)
            break;
74         else
            broj_usluzenih = 0;
76     }
78 }
79
80 /*****
81  Usluzivanje korisnika moze da se izvrši i na sledeći način:
82  *****/
83 while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
84     printf("\nSledeći je korisnik sa JMBG brojem: %s\n",
85         nov_zahtev.jmbg);
86     printf("sa zahtevom: %s\n", nov_zahtev.opis);
87     broj_usluzenih++;
88
89     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
90     scanf("%s", odgovor);
91     if (strcmp(odgovor, "Da") == 0)
92         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
93     else
94         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
95             nov_zahtev.jmbg, nov_zahtev.opis);
96
97     if (broj_usluzenih == VREME_ZA_PAUZU) {
98         printf("\nDa li je kraj smene? [Da/Ne] ");
99         scanf("%s", odgovor);
100        if (strcmp(odgovor, "Da") == 0)
101            break;
102        else
103            broj_usluzenih = 0;
104    }
105 }

```

```

106  *****/
108  /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
      neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
      koju zauzima red sa neobradjenim zahtevima korisnika. */
110  oslobodi_red(&pocetak, &kraj);
112  return 0;
    }

```

### Rešenje 4.7

```

1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  #define MAX_DUZINA 20
5
6  typedef struct _Cvor {
7      unsigned broj_pojavljivanja;
8      char etiketa[20];
9      struct _Cvor *sledeci;
10 } Cvor;
11
12 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
13    ili NULL ako alokacija nije uspesno izvsena. */
14 Cvor *napravi_cvor(unsigned br, char *etiketa)
15 {
16     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
17     if (novi == NULL)
18         return NULL;
19
20     novi->broj_pojavljivanja = br;
21     strcpy(novi->etiketa, etiketa);
22     novi->sledeci = NULL;
23     return novi;
24 }
25
26 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste. */
27 void oslobodi_listu(Cvor ** adresa_glave)
28 {
29     Cvor *pomocni = NULL;
30
31     while (*adresa_glave != NULL) {
32         pomocni = (*adresa_glave)->sledeci;
33         free(*adresa_glave);
34         *adresa_glave = pomocni;
35     }
36 }
37
38 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
39    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno

```

```

41     zauzeta memorija za listu ciji pokazivac glava se nalazi na adresi
    adresa_glave i prekida program. */
42 void provera_alokacije(Cvor * novi, Cvor ** adresa_glave)
43 {
44     if (novi == NULL) {
45         fprintf(stderr, "malloc() greska u funkciji napravi_cvor()!\n");
46         oslobodi_listu(adresa_glave);
47         exit(EXIT_FAILURE);
48     }
49 }

51 /* Funkcija dodaje novi cvor na pocetak liste. */
52 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
53                             char *etiketa)
54 {
55     Cvor *novi = napravi_cvor(br, etiketa);
56     provera_alokacije(novi, adresa_glave);
57     novi->sledeci = *adresa_glave;
58     *adresa_glave = novi;
59 }

61 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
    NULL ako takav cvor ne postoji. */
62 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
63 {
64     Cvor *tekuci;
65     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
66         if (strcmp(tekuci->etiketa, etiketa) == 0)
67             return tekuci;
68     return NULL;
69 }

71 /* Funkcija ispisuje sadrzaj liste */
72 void ispisi_listu(Cvor * glava)
73 {
74     for (; glava != NULL; glava = glava->sledeci)
75         printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
76 }

77 /* Glavni program */
78 int main(int argc, char **argv)
79 {
80     if (argc != 2) {
81         fprintf(stderr,
82                 "Greska! Program se poziva sa: ./a.out datoteka.html!\n");
83         ;
84         exit(EXIT_FAILURE);
85     }
86 }

87 /* Otvaramo datoteku za citanje */
88 FILE *in = NULL;
89 in = fopen(argv[1], "r");

```

```

91     if (in == NULL) {
92         fprintf(stderr,
93             "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
94         exit(EXIT_FAILURE);
95     }

97     char c;
98     int i = 0;
99     char procitana[MAX_DUZINA];
100     Cvor *glava = NULL;
101     Cvor *trazeni = NULL;

103     while ((c = fgetc(in)) != EOF) {

105         if (c == '<') {
106             /* Cita se zatvorena etiketa. */
107             if ((c = fgetc(in)) == '/') {
108                 i = 0;
109                 while ((c = fgetc(in)) != '>')
110                     procitana[i++] = c;
111             }
112             /* Cita se otvorena etiketa. */
113             else {
114                 i = 0;
115                 procitana[i++] = c;
116                 while ((c = fgetc(in)) != ' ' && c != '>')
117                     procitana[i++] = c;
118             }
119             procitana[i] = '\0';

121             /* Trazi se ucitana etiketa medju postojećim cvorovima liste.
122              * Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
123              * sa brojem pojavljivanja 1, inace uvecava se broj
124              * pojavljivanja etikete. */
125             trazeni = pretrazi_listu(glava, procitana);
126             if (trazeni == NULL)
127                 dodaj_na_pocetak_liste(&glava, 1, procitana);
128             else
129                 trazeni->broj_pojavljivanja++;
130         }
131     }

133     fclose(in);

135     ispisi_listu(glava);
136     oslobodi_listu(&glava);
137
138     return 0;
139 }

```

## Rešenje 4.8

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

4
#define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21

8 /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
   studentu. */
10 typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
12    char ime[MAX_IME_PREZIME];
    char prezime[MAX_IME_PREZIME];
14    struct _Cvor *sledeci;
} Cvor;

16
/* Funkcija kreira, inicijalizuje cvor liste i vraća pokazivac na nov
   cvor ili NULL ukoliko alokacija nije prosla. */
18 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
22    if (novi == NULL)
        return NULL;
24    /* Inicijalizacija polja novog cvora */
    strcpy(novi->broj_indeksa, broj_indeksa);
26    strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
28    novi->sledeci = NULL;
    return novi;
30 }

32 /* Funkcija oslobadja memoriju zauzetu za cvorove liste. */
void oslobodi_listu(Cvor ** adresa_glave)
34 {
    if (*adresa_glave == NULL)
36        return;
    /* Rep liste se oslobadja rekurzivnim pozivom. */
38    oslobodi_listu(&(*adresa_glave)->sledeci);
    /* Potom se oslobadja i glava liste. */
40    free(*adresa_glave);
    *adresa_glave = NULL;
42 }

44 /* Funkcija proverava da li je novi NULL pokazivac, i ukoliko jeste
   oslobadja celu listu ciji se pokazivac glava nalazi na adresi
46 adresa_glave i prekida program. */
void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
48 {
    if (novi == NULL) {
50        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_listu(adresa_glave);
    }
}
```



```
52     exit(EXIT_FAILURE);
53 }
54 }
55
56 /* Funkcija dodaje novi cvor na pocetak liste. */
57 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
58                             char *ime, char *prezime)
59 {
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
61     prover_i_alokaciju(adresa_glave, novi);
62     novi->sledeci = *adresa_glave;
63     *adresa_glave = novi;
64 }
65
66 /* Funkcija ispisuje sadrzaj cvorova liste. */
67 void ispisi_listu(Cvor * glava)
68 {
69     for (; glava != NULL; glava = glava->sledeci)
70         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
71                glava->prezime);
72 }
73
74 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu, u
75    suprotnom vraca NULL. */
76 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
77 {
78     if (glava == NULL)
79         return NULL;
80     if (!strcmp(glava->broj_indeksa, broj_indeksa))
81         return glava;
82     return pretrazi_listu(glava->sledeci, broj_indeksa);
83 }
84
85 int main(int argc, char **argv)
86 {
87     /* Argumenti komandne linije su neophodni jer se iz komandne linije
88        dobija ime datoteke sa informacijama o studentima. */
89     if (argc != 2) {
90         fprintf(stderr,
91                "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
92         exit(EXIT_FAILURE);
93     }
94
95     /* Otvaranje datoteke za citanje */
96     FILE *in = NULL;
97     in = fopen(argv[1], "r");
98     if (in == NULL) {
99         fprintf(stderr,
100                "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
101         exit(EXIT_FAILURE);
102     }
103 }
```

```

104  /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
      listu */
106  char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
      char broj_indeksa[MAX_INDEKS];
108  Cvor *glava = NULL;
      Cvor *trazeni = NULL;
110
      /* Ucitavanje vrednosti u listu */
112  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
      dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);
114
      /* Datoteka vise nije potrebna i zatvara se. */
116  fclose(in);

      /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
118  while (scanf("%s", broj_indeksa) != EOF) {
120      trazeni = pretrazi_listu(glava, broj_indeksa);
      if (trazeni == NULL)
122          printf("ne\n");
      else
124          printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
  }
126
      /* Oslobadja se memorija zauzeta za cvorove liste. */
128  oslobodi_listu(&glava);

130  return 0;
  }

```

### Rešenje 4.9

```

1  #include<stdio.h>
      #include<stdlib.h>
3  #include "601/lista.h"

5  /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
      na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7  pokazivaca postojećih cvorova listi. */
      Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9  {
      /* Pokazivac na glavu rezultujuće liste. */
11  Cvor *rezultujuca = NULL;
      /* Tekuci je pokazivac na pokazivac kome sledecem treba promeniti
13  vrednosti. Inicijalizuje se na adresu pokazivaca rezultujuca jer
      prvo treba odrediti glavu rezultujuće liste. */
15  Cvor **tekuci = &rezultujuca;

17  /* Ako su obe liste prazne, rezultat je isto prazna lista. */
      if (*adresa_glave_1 == NULL && *adresa_glave_2 == NULL)
19      return NULL;

```

```

21  /* Ako je prva lista prazna, rezultat je druga lista. */
    if (*adresa_glave_1 == NULL)
23      return *adresa_glave_2;

25  /* Ako je druga lista prazna, rezultat je prva lista. */
    if (*adresa_glave_2 == NULL)
27      return *adresa_glave_1;

29  /* Sve dok u obe liste ima cvorova, azurira se vrednost pokazivaca
    na koji tekuci pokazuje. U prvoj iteraciji tekuci pokazuje na
31  pokazivac rezultujuca i ovako se pokazivac rezultujuca usmerava
    da pokazuje na pocetak nove liste, tj. na cvor sa vrednoscu
33  manjeg od brojeva sadrzanih u cvorovima na koje vode pokazivaci
    na adresama adresa_glave_1 i adresa_glave_2. U svim ostalim
35  iteracijama to isto se dogadja samo pokazivacu na koji tekuci u
    tom trenutku pokazuje. */
37  while (*adresa_glave_1 != NULL && *adresa_glave_2 != NULL) {
    if ((*adresa_glave_1)->vrednost < (*adresa_glave_2)->vrednost) {
39      /* Pokazivac na koji tekuci pokazuje dobija vrednosti
        pokazivaca koji se nalazi na adresa_glave_1. Time sledbenik
        poslednjeg uvezanog cvora postaje cvor koji je aktuelna
        glava prve liste. */
41      *tekuci = *adresa_glave_1;
        /* Pomera se glava prve liste na sledeci cvor prve liste.

43      Ova promena bice vidljiva i van funkcije jer se direktno
        menja promenljiva koja se nalazi na adresi adresa_glave_1.
45      */
        *adresa_glave_1 = (*adresa_glave_1)->sledeci;
49    } else {
        /* Sledbenik poslednjeg uvezanog cvora bice cvor koji je
        aktuelna glava druge liste. */
51      *tekuci = *adresa_glave_2;
        /* Pomera se glava druge liste na sledeci cvor druge liste */
53      *adresa_glave_2 = (*adresa_glave_2)->sledeci;
55    }
    /* Tekuci se pomera na pokazivac sledeci od poslednjeg uvezanog,
    jer je upravo to pokazivac koji treba da bude azuriran u
57    sledecoj iteraciji petlje. */
    tekuci = &((*tekuci)->sledeci);
59  }

61  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
    rezultujucu listu treba nadovezati ostatak druge liste. Tako
63    sledbenik poslednjeg uvezanog cvora treba da bude ostatak druge
    liste. */
65  if (*adresa_glave_1 == NULL)
    *tekuci = *adresa_glave_2;
67  else {
    if (*adresa_glave_2 == NULL)
69      *tekuci = *adresa_glave_1;
71  }

```

```
73     return rezultujuca;
74 }
75
76 /* Druga verzija prethodne funkcije koja ne pristupa pokazivacima
77 preko adresa vec direktno. Ne salju joj se adrese, vec vrednosti
78 pokazivaca na glave listi. */
79 Cvor *objedini_v2(Cvor * lista1, Cvor * lista2)
80 {
81     Cvor *rezultujuca = NULL;
82     Cvor *tekuci = NULL;
83
84     /* Ako su obe liste prazne i rezultat je prazna lista. */
85     if (lista1 == NULL && lista2 == NULL)
86         return NULL;
87
88     /* Ako je prva lista prazna, rezultat je druga lista. */
89     if (lista1 == NULL)
90         return lista2;
91
92     /* Ako je druga lista prazna, rezultat je prva lista. */
93     if (lista2 == NULL)
94         return lista1;
95
96     /* Rezultujuca pokazuje na pocetak nove liste, tj. na cvor sa
97 vrednoscu manjeg od brojeva sadrzanih u cvorovima na koje
98 pokazuju lista1 i lista2. */
99     if (lista1->vrednost < lista2->vrednost) {
100         rezultujuca = lista1;
101         lista1 = lista1->sledeci;
102     } else {
103         rezultujuca = lista2;
104         lista2 = lista2->sledeci;
105     }
106     tekuci = rezultujuca;
107
108     /* Kako rezultujuca pokazuje na pocetak nove liste i ne sme joj se
109 menjati vrednost, koristi se pokazivac tekuci koji trenutno
110 sadrzi adresu promenljive rezultujuca. U svakoj iteraciji
111 petlje, dobijace adekvatnog sledbenika tako da i nova lista bude
112 uredjena neopadajuce i pomerace se na adresu sledeceg. */
113     while (lista1 != NULL && lista2 != NULL) {
114         if (lista1->vrednost < lista2->vrednost) {
115             tekuci->sledeci = lista1;
116             lista1 = lista1->sledeci;
117         } else {
118             tekuci->sledeci = lista2;
119             lista2 = lista2->sledeci;
120         }
121         tekuci = tekuci->sledeci;
122     }
123 }
```

```
125  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
    rezultujuću listu treba nadovezati ostatak druge liste. */
127  if (lista1 == NULL)
    tekuci->sledeci = lista2;
129  else
    tekuci->sledeci = lista1;

131  return rezultujuća;
}

133
134 /* Glavni program */
135 int main(int argc, char **argv)
{
137     /* Argumenti komandne linije su neophodni. */
    if (argc != 3) {
139         fprintf(stderr,
            "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
141         exit(EXIT_FAILURE);
    }

143
    /* Otvaramo datoteke sa elementima obe liste. */
145     FILE *in1 = NULL;
    in1 = fopen(argv[1], "r");
147     if (in1 == NULL) {
        fprintf(stderr,
149             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
151     }

153     FILE *in2 = NULL;
    in2 = fopen(argv[2], "r");
155     if (in2 == NULL) {
        fprintf(stderr,
157             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
        exit(EXIT_FAILURE);
159     }

161     int broj;
    Cvor *lista1 = NULL;
163     Cvor *lista2 = NULL;
    Cvor *rezultat = NULL;

165
    /* Ucitavanje listi */
167     while (fscanf(in1, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&lista1, broj);
169     while (fscanf(in2, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&lista2, broj);

171
    /* Pokazivac rezultat ce pokazivati na glavu liste koja se dobila
    objedinjavanjem listi */
173     rezultat = objedini(&lista1, &lista2);
175 }
```

```

177  /*****
179  Poziv druge verzije prethodne funkcije
181
183  rezultat = objedini_v2(lista1, lista2);
185  *****/
187  /* Ispis rezultujuće liste. */
189  ispisi_listu(rezultat);
191
193  /* Kako je lista rezultat dobijena prevezivanjem cvorova polaznih
   listi, njenim oslobadjanjem bice oslobodjena sva zauzeta
   memorija. */
   oslobodi_listu(&rezultat);

   fclose(in1);
   fclose(in2);
   return 0;
}

```

#### Rešenje 4.14

```

2  #include <stdio.h>
   #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 vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj)
   {
16     /* Alokira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
18     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
20         return NULL;

22     /* Inicijalizuju se polja novog cvora. */
       novi->broj = broj;
       novi->levo = NULL;
       novi->desno = NULL;

26     /* Vraca se adresa novog cvora. */
       return novi;
28 }
30

```

```
32 /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
33 void prover_i_alokaciju(Cvor * novi_cvor)
34 {
35     /* Ukoliko je cvor neuspesno kreiran */
36     if (novi_cvor == NULL) {
37
38         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa */
39         fprintf(stderr, "Malloc greska za novi cvor!\n");
40         exit(EXIT_FAILURE);
41     }
42 }
43
44 /* c) Funkcija koja dodaje zadati broj u stablo */
45 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
46 {
47     /* Ako je stablo prazno */
48     if (*adresa_korena == NULL) {
49
50         /* Kreira se novi cvor */
51         Cvor *novi = napravi_cvor(broj);
52         prover_i_alokaciju(novi);
53
54         /* I proglašava se korenom stabla */
55         *adresa_korena = novi;
56         return;
57     }
58
59     /* U suprotnom trazi se odgovarajuca pozicija za zadati broj: */
60
61     /* Ako je zadata vrednost manja od vrednosti korena */
62     if (broj < (*adresa_korena)->broj)
63
64         /* Broj se dodaje u levo podstablo */
65         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
66
67     else
68         /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
69         dodaje u desno podstablo */
70         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
71 }
72
73 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
74 Cvor *pretrazi_stablo(Cvor * koren, int broj)
75 {
76
77     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
78     if (koren == NULL)
79         return NULL;
80
81     /* Ako je trazena vrednost sadrzana u korenu */
```

```
84     if (koren->broj == broj) {
86         /* Prekidamo pretragu */
87         return koren;
88     }
89
90     /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
91     if (broj < koren->broj)
92     {
93         /* Pretraga se nastavlja u levom podstablu */
94         return pretrazi_stablo(koren->levo, broj);
95     }
96     else
97     {
98         /* U suprotnom, pretraga se nastavlja u desnom podstablu */
99         return pretrazi_stablo(koren->desno, broj);
100     }
101
102     /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost u stablu */
103     Cvor *pronadji_najmanji(Cvor * koren)
104     {
105         /* Ako je stablo prazno, prekida se pretraga */
106         if (koren == NULL)
107             return NULL;
108
109         /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
110            levo od njega */
111
112         /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
113            najmanju vrednost */
114         if (koren->levo == NULL)
115             return koren;
116
117         /* Inace, pretragu treba nastaviti u levom podstablu */
118         return pronadji_najmanji(koren->levo);
119     }
120
121     /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u stablu */
122     Cvor *pronadji_najveci(Cvor * koren)
123     {
124         /* Ako je stablo prazno, prekida se pretraga */
125         if (koren == NULL)
126             return NULL;
127
128         /* Vrednosti koje su veće od vrednosti u korenu stabla nalaze se
129            desno od njega */
130
131         /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
132            najveću vrednost */
133         if (koren->desno == NULL)
```



```

134     return koren;

136     /* Inace, pretragu treba nastaviti u desnom podstablu */
136     return pronadji_najveci(koren->desno);
138 }

138 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
140 void obrisi_element(Cvor ** adresa_korena, int broj)
142 {
142     Cvor *pomocni_cvor = NULL;

144     /* Ako je stablo prazno, brisanje nije primenljivo */
144     if (*adresa_korena == NULL)
146         return;

148     /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
150        stabla, ona se eventualno nalazi u levom podstablu, pa treba
150        rekursivno primeniti postupak na levo podstablo. Koren ovako
150        modifikovanog stabla je nepromenjen. */
152     if (broj < (*adresa_korena)->broj) {
152         obrisi_element(&(*adresa_korena)->levo, broj);
154         return;
156     }

158     /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
160        stabla, ona se eventualno nalazi u desnom podstablu pa treba
160        rekursivno primeniti postupak na desno podstablo. Koren ovako
160        modifikovanog stabla je nepromenjen. */
162     if ((*adresa_korena)->broj < broj) {
162         obrisi_element(&(*adresa_korena)->desno, broj);
164         return;
166     }

168     /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
168        jednaka broju koji se brise (tj. slucaj kada treba obrisati
168        koren) */

170     /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
172        prazno stablo (vraca se NULL) */
172     if ((*adresa_korena)->levo == NULL
172         && (*adresa_korena)->desno == NULL) {
174         free(*adresa_korena);
174         *adresa_korena = NULL;
176         return;
178     }

180     /* Ako koren ima samo levog sina, tada se brisanje vrsi tako sto se
182        brise koren, a novi koren postaje levi sin */
182     if ((*adresa_korena)->levo != NULL
182         && (*adresa_korena)->desno == NULL) {
184         pomocni_cvor = (*adresa_korena)->levo;
184         free(*adresa_korena);

```

```

186     *adresa_korena = pomocni_cvor;
187     return;
188 }
189
190 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako sto
191 se briše koren, a novi koren postaje desni sin */
192 if ((*adresa_korena)->desno != NULL
193     && (*adresa_korena)->levo == NULL) {
194     pomocni_cvor = (*adresa_korena)->desno;
195     free(*adresa_korena);
196     *adresa_korena = pomocni_cvor;
197     return;
198 }
199
200 /* Slučaj kada koren ima oba sina - najpre se potraži sledbenik
201 korena (u smislu poretka) u stablu. To je upravo po vrednosti
202 najmanji cvor u desnom podstablu. On se može pronaći npr.
203 funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
204 vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
205 broj koji se briše). Zatim se prosto rekurzivno pozove funkcija
206 za brisanje na desno podstablo. S obzirom da u njemu treba
207 obrisati najmanji element, a on zasigurno ima najviše jednog
208 potomka, jasno je da će njegovo brisanje biti obavljeno na jedan
209 od jednostavnijih načina koji su gore opisani. */
210 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
211 (*adresa_korena)->broj = pomocni_cvor->broj;
212 pomocni_cvor->broj = broj;
213 obrisi_element(&(*adresa_korena)->desno, broj);
214 }
215
216 /* h) Funkcija ispisuje stablo u infiksnoj notaciji (Levo postablo -
217 Koren - Desno podstablo) */
218 void ispisi_stablo_infiksno(Cvor * koren)
219 {
220     /* Ako stablo nije prazno */
221     if (koren != NULL) {
222
223         /* Prvo se ispisuju svi cvorovi levo od korena */
224         ispisi_stablo_infiksno(koren->levo);
225
226         /* Zatim se ispisuje vrednost u korenu */
227         printf("%d ", koren->broj);
228
229         /* Na kraju se ispisuju cvorovi desno od korena */
230         ispisi_stablo_infiksno(koren->desno);
231     }
232 }
233
234 /* i) Funkcija ispisuje stablo u prefiksnoj notaciji (Koren - Levo
235 podstablo - Desno podstablo) */
236 void ispisi_stablo_prefiksno(Cvor * koren)
237 {

```

```
238  /* Ako stablo nije prazno */
    if (koren != NULL) {
240      /* Prvo se ispisuje vrednost u korenu */
      printf("%d ", koren->broj);
242
      /* Zatim se ispisuju svi cvorovi levo od korena */
244      ispisi_stablo_prefiksno(koren->levo);
246
      /* Na kraju se ispisuju svi cvorovi desno od korena */
      ispisi_stablo_prefiksno(koren->desno);
248    }
  }
250
  /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo podstablo
252   - Desno postablo - Koren) */
  void ispisi_stablo_postfiksno(Cvor * koren)
254  {
256    /* Ako stablo nije prazno */
    if (koren != NULL) {
258
      /* Prvo se ispisuju svi cvorovi levo od korena */
260      ispisi_stablo_postfiksno(koren->levo);
262
      /* Zatim se ispisuju svi cvorovi desno od korena */
      ispisi_stablo_postfiksno(koren->desno);
264
      /* Na kraju se ispisuje vrednost u korenu */
266      printf("%d ", koren->broj);
    }
268  }
270
  /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
  void oslobodi_stablo(Cvor ** adresa_korena)
272  {
274    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*adresa_korena == NULL)
      return;
276
    /* Inace ... */
278    /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);
280
    /* Oslobadja se memorija zauzeta desnim podstablom */
282    oslobodi_stablo(&(*adresa_korena)->desno);
284
    /* Oslobadja se memorija zauzeta korenom */
    free(*adresa_korena);
286
    /* Proglasava se stablo praznim */
288    *adresa_korena = NULL;
```

```

}
290
int main()
292 {
    Cvor *koren;
294     int n;
    Cvor *trazeni_cvor;
296
    /* Proglasava se stablo praznim */
298     koren = NULL;

300     /* Dodaju se vrednosti u stablo */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
302     while (scanf("%d", &n) != EOF) {
        dodaj_u_stablo(&koren, n);
304     }

306     /* Generisu se trazeni ispisi: */
    printf("\nInfiksni ispisi: ");
308     ispisi_stablo_infiksno(koren);
    printf("\nPrefiksni ispisi: ");
310     ispisi_stablo_prefiksno(koren);
    printf("\nPostfiksni ispisi: ");
312     ispisi_stablo_postfiksno(koren);

314     /* Demonstrira se rad funkcije za pretragu */
    printf("\nTrazi se broj: ");
316     scanf("%d", &n);
    trazeni_cvor = pretrazi_stablo(koren, n);
318     if (trazeni_cvor == NULL)
        printf("Broj se ne nalazi u stablu!\n");
320
    else
322     printf("Broj se nalazi u stablu!\n");

324     /* Demonstrira se rad funkcije za brisanje */
    printf("Brise se broj: ");
326     scanf("%d", &n);
    obrisi_element(&koren, n);
328     printf("Rezultujuce stablo: ");
    ispisi_stablo_infiksno(koren);
330     printf("\n");

332     /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
334     return 0;
}

```

## Rešenje 4.15

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #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 {
11     char *rec;
12     int broj;
13     struct cvor *levo;
14     struct cvor *desno;
15 } Cvor;

16 /* Funkcija koja kreira novi cvor stabla */
18 Cvor *napravi_cvor(char *rec)
19 {
20     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
22     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
23     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 =
29         (char *) malloc((strlen(rec) + 1) * sizeof(char));
30     if (novi_cvor->rec == NULL) {
31         free(novi_cvor);
32         return NULL;
33     }

36     /* Inicijalizuju se polja u novom cvoru */
37     strcpy(novi_cvor->rec, rec);
38     novi_cvor->broj = 1;
39     novi_cvor->levo = NULL;
40     novi_cvor->desno = NULL;

42     /* Vraca se adresa novog cvora */
43     return novi_cvor;
44 }

46 /* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
48 void prover_i_alokaciju(Cvor * novi_cvor)
49 {
50     /* Ukoliko je cvor neuspesno kreiran */
51     if (novi_cvor == NULL) {
52         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje

```

```
54     programa */
55     fprintf(stderr, "Malloc greska za novi cvor!\n");
56     exit(EXIT_FAILURE);
57 }
58 }
59
60 /* Funkcija koja dodaje novu rec u stablo. */
61 void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
62 {
63     /* Ako je stablo prazno */
64     if (*adresa_korena == NULL) {
65         /* Kreira se cvor koji sadrzi zadatu rec */
66         Cvor *novi = napravi_cvor(rec);
67         prover_i_alokaciju(novi);
68
69         /* I proglašava se korenom stabla */
70         *adresa_korena = novi;
71         return;
72     }
73
74     /* U suprotnom se trazi odgovarajuca pozicija za novu rec */
75
76     /* Ako je rec leksikografski manja od reci u korenu ubacuje se
77        u levo podstablo */
78     if (strcmp(rec, (*adresa_korena)->rec) < 0)
79         dodaj_u_stablo(&(*adresa_korena)->levo, rec);
80
81     else
82         /* Ako je rec leksikografski veca od reci u korenu ubacuje
83            se u desno podstablo */
84         if (strcmp(rec, (*adresa_korena)->rec) > 0)
85             dodaj_u_stablo(&(*adresa_korena)->desno, rec);
86
87     else
88         /* Ako je rec jednaka reci u korenu, uvecava se njen broj
89            pojavljivanja */
90         (*adresa_korena)->brojac++;
91 }
92
93 /* Funkcija koja oslobadja memoriju zauzetu stablom */
94 void oslobodi_stablo(Cvor ** adresa_korena)
95 {
96     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
97     if (*adresa_korena == NULL)
98         return;
99
100     /* Inace ... */
101     /* Oslobadja se memorija zauzeta levim podstablom */
102     oslobodi_stablo(&(*adresa_korena)->levo);
103
104     /* Oslobadja se memorija zauzeta desnim podstablom */
105     oslobodi_stablo(&(*adresa_korena)->desno);
```

```
106  /* Oslobadja se memorija zauzeta korenom */
107  free((*adresa_korena)->rec);
108  free(*adresa_korena);

109
110  /* Stablo se proglašava praznim */
111  *adresa_korena = NULL;
112  }

113
114  /* Funkcija koja pronalazi cvor koji sadrži najfrekventniju rec
115   (rec sa najvećim brojem pojavljivanja) */
116  Cvor *nadji_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 = nadji_najfrekventniju_rec(koren->levo);
126
127      /* Pronalazi se najfrekventnija rec u desnom podstablu */
128      max_desno = nadji_najfrekventniju_rec(koren->desno);

129
130      /* Traži 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 pracen 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  }
```

```
158     /* I nastavlja se sa ispisom reci iz desnog podstabla */
    prikazi_stablo(koren->desno);
160 }
161
162 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje
    je u niz rec. Maksimalna duzina reci je odredjena argumentom
    max. Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
164     suprotnom. Rec je niz malih ili velikih slova. */
    int procitaj_rec(FILE * f, char rec[], int max)
166 {
167     /* Karakter koji se cita */
168     int c;
169
170     /* Indeks pozicije na koju se smesta procitani karakter */
171     int i = 0;
172
173     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se
174     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 vrsi konverzija
179             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
185             slovo nove reci */
186             /* Ako jeste, prekida se sa citanjem */
187             if (i > 0)
188                 break;
189
190         /* U suprotnom se ide na sledecu iteraciju */
191     }
192
193     /* Dodaje se na rec terminirajuca nula */
194     rec[i] = '\0';
195
196     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
197     return i > 0 ? 0 : EOF;
198 }
199
200 int main(int argc, char **argv)
201 {
202     Cvor *koren = NULL, *max;
203     FILE *f;
204     char rec[MAX];
205
206     /* Provera da li je navedeno ime datoteke prilikom pokretanja
207     programa */
208     if (argc < 2) {
```



```

210     fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
        exit(EXIT_FAILURE);
    }
212
    /* Priprema datoteke za citanje */
214     if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
216             argv[1]);
        exit(EXIT_FAILURE);
    }
218
    /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
       pretrage. */
220     while (procitaj_rec(f, rec, MAX) != EOF)
        dodaj_u_stablo(&koren, rec);
222
    /* Posto je citanjem reci zavrsono, zatvara se datoteka */
224     fclose(f);
226
    /* Prikazuju se sve reci iz teksta i brojevi njihovih
       pojavljivanja. */
228     prikazi_stablo(koren);
230
    /* Pronalazi se najfrekventnija rec */
232     max = nadji_najfrekventniju_rec(koren);
234
    /* Ako takve reci nema... */
236     if (max == NULL)

        /* Ispisuje se odgovarajuće obavestjenje */
238         printf("U tekstu nema reci!\n");
240
    else
242         /* Inace, ispisuje se broj pojavljivanja reci */
        printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
244             max->rec, max->brojac);

    /* Oslobadja se dinamicki alociran prostor za stablo */
246     oslobodi_stablo(&koren);
248
    return 0;
250 }

```

### Rešenje 4.16

```

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

6  #define MAX_IME_DATOTEKE 50

```

```
8 #define MAX_CIFARA 13
10 #define MAX_IME_I_PREZIME 100
12 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
14 broj telefona i redom pokazivace na levo i desno podstablo */
16 typedef struct cvor {
18     char ime_i_prezime[MAX_IME_I_PREZIME];
20     char telefon[MAX_CIFARA];
22     struct cvor *levo;
24     struct cvor *desno;
26 } Cvor;
28 /* Funkcija koja kreira novi cvor stabla */
30 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
32 {
34     /* Alocira se memorija za novi cvor i proverava se uspesnost
36     alokacije. */
38     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
40     if (novi_cvor == NULL)
42         return NULL;
44     /* Inicijalizuju se polja novog cvora */
46     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
48     strcpy(novi_cvor->telefon, telefon);
50     novi_cvor->levo = NULL;
52     novi_cvor->desno = NULL;
54     /* Vraca se adresa novog cvora */
56     return novi_cvor;
58 }
60 /* Funkcija koja proverava uspesnost kreiranja novog cvora
62 stabla */
64 void proveri_alokaciju(Cvor * novi_cvor)
66 {
68     /* Ukoliko je cvor neuspesno kreiran */
70     if (novi_cvor == NULL) {
72         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
74         programa */
76         fprintf(stderr, "Malloc greska za novi cvor!\n");
78         exit(EXIT_FAILURE);
80     }
82 }
84 /* Funkcija koja dodaje novu osobu i njen broj telefona u
86 stablo. */
88 void
90 dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
92 char *telefon)
94 {
96     /* Ako je stablo prazno */
98     if (*adresa_korena == NULL) {
```

```

60      /* Kreira se novi cvor */
        Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
        prover_i_alokaciju(novi);

62
        /* I proglašava se korenom stabla */
64      *adresa_korena = novi;
        return;
66    }

68    /* U suprotnom trazi se odgovarajuća pozicija za novi unos.
        Kako pretragu treba vrsiti po imenu i prezimenu, stablo
        treba da bude pretraživacko po ovom polju */

70
72    /* Ako je zadato ime i prezime leksikografski manje od imena i
        prezimena sadržanog u korenu, podaci se dodaju u levo
        podstablo */
74    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
76        < 0)
        dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
78            telefon);

80    else
        /* Ako je zadato ime i prezime leksikografski veće od imena
        i prezimena sadržanog u korenu, podaci se dodaju u desno
        podstablo */
82    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
        dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
84            telefon);
86    }

88
89    /* Funkcija koja oslobadja memoriju zauzetu stablom */
90    void oslobodi_stablo(Cvor ** adresa_korena)
    {
92        /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
        if (*adresa_korena == NULL)
94            return;

96        /* Inace ... */
        /* Oslobadja se memorija zauzeta levim podstablom */
98        oslobodi_stablo(&(*adresa_korena)->levo);

100        /* Oslobadja se memorija zauzeta desnim podstablom */
        oslobodi_stablo(&(*adresa_korena)->desno);

102        /* Oslobadja se memorija zauzeta korenom */
104        free(*adresa_korena);

106        /* Stablo se proglašava praznim */
        *adresa_korena = NULL;
108    }

110    /* Funkcija koja ispisuje imenik u leksikografskom poretku */

```

```

112  /* Napomena: ova funkcija nije trazena u zadatku ali se moze
      koristiti za proveru da li je stablo lepo kreirano ili ne */
void prikazi_stablo(Cvor * koren)
114 {
      /* Ako je stablo prazno, završava se sa ispisom */
116     if (koren == NULL)
            return;

118     /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz
      levog podstabla */
120     prikazi_stablo(koren->levo);

122     /* Zatim se ispisuju podaci iz korena */
124     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

126     /* I nastavlja se sa ispisom podataka iz desnog podstabla */
      prikazi_stablo(koren->desno);
128 }

130 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
      ime i prezime i broj telefona u odgovarajuce nizove.
132     Maksimalna duzina imena i prezimena odredjena je konstantom
      MAX_IME_PREZIME, a maksimalna duzina broja telefona
134     konstantom MAX_CIFARA. Funkcija vraca EOF ako nema vise
      kontakata ili 0 u suprotnom. */
136 int procitaj_kontakt(FILE * f, char *ime_i_prezime,
                        char *telefon)
138 {
      /* Karakter koji se cita */
140     int c;

142     /* Indeks pozicije na koju se smesta procitani karakter */
      int i = 0;

144     /* Linije datoteke koje se obradjuju su formata Ime Prezime
      BrojTelefona */

146     /* Preskacu se eventualne praznine sa pocetka linije datoteke */
      while ((c = fgetc(f)) != EOF && isspace(c));

148     /* Prvo procitano slovo upisuje se u ime i prezime */
150     if (!feof(f))
            ime_i_prezime[i++] = c;

152     /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
      cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
154     Pritom treba voditi racuna da li ima dovoljno mesta za
      smestanje procitanog karaktera i da se slucajno ne dodje do
      kraja datoteke */
156     while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
            if (!isdigit(c))
158                 ime_i_prezime[i++] = c;
    }
162

```

```
164     else if (i > 0)
165         break;
166 }
167
168 /* Upisuje se terminirajuca nula na mesto poslednjeg
169    procitanog blanko karaktera */
170 ime_i_prezime[--i] = '\0';
171
172 /* I pocinje se sa citanjem broja telefona */
173 i = 0;
174
175 /* Upisuje se cifra koja je vec procitana */
176 telefon[i++] = c;
177
178 /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
179    karaktera cije prisustvo nije dozvoljeno u broju telefona */
180 while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
181     if (c == '/' || c == '-' || isdigit(c))
182         telefon[i++] = c;
183     else
184         break;
185
186 /* Upisuje se terminirajuca nula */
187 telefon[i] = '\0';
188
189 /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
190 return !feof(f) ? 0 : EOF;
191 }
192
193 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
194    prezimenom */
195 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
196 {
197     /* Ako je imenik prazan, završava se sa pretragom */
198     if (koren == NULL)
199         return NULL;
200
201     /* Ako je trazeno ime i prezime sadržano u korenu, takodje se
202        završava sa pretragom */
203     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
204         return koren;
205
206     /* Ako je zadato ime i prezime leksikografski manje od
207        vrednosti u korenu pretraga se nastavlja levo */
208     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
209         return pretrazi_imenik(koren->levo, ime_i_prezime);
210
211     else
212         /* u suprotnom, pretraga se nastavlja desno */
213         return pretrazi_imenik(koren->desno, ime_i_prezime);
214 }
```

```
216 int main(int argc, char **argv)
217 {
218     char ime_datoteke[MAX_IME_DATOTEKE];
219     Cvor *koren = NULL;
220     Cvor *trazeni;
221     FILE *f;
222     char ime_i_prezime[MAX_IME_I_PREZIME];
223     char telefon[MAX_CIFARA];
224     char c;
225     int i;
226
227     /* Ucitava se ime datoteke i vrsi se njena priprema za citanje
228      */
229     printf("Unesite ime datoteke: ");
230     scanf("%s", ime_datoteke);
231     if ((f = fopen(ime_datoteke, "r")) == NULL) {
232         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
233                 ime_datoteke);
234         exit(EXIT_FAILURE);
235     }
236
237     /* Podaci se citaju iz datoteke i smestanju u binarno stablo
238      pretrage. */
239     while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
240         dodaj_u_stablo(&koren, ime_i_prezime, telefon);
241
242     /* Zatvara se datoteka */
243     fclose(f);
244
245     /* Omogucava se pretraga imenika */
246     while (1) {
247         /* Ucitavaja se ime i prezime */
248         printf("Unesite ime i prezime: ");
249         i = 0;
250         while ((c = getchar()) != '\n')
251             ime_i_prezime[i++] = c;
252         ime_i_prezime[i] = '\0';
253
254         /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja
255          se funkcionalnost */
256         if (strcmp(ime_i_prezime, "KRAJ") == 0)
257             break;
258
259         /* Inace se ispisuje rezultat pretrage */
260         trazeni = pretrazi_imenik(koren, ime_i_prezime);
261         if (trazeni == NULL)
262             printf("Broj nije u imeniku!\n");
263         else
264             printf("Broj je: %s \n", trazeni->telefon);
265     }
266 }
```

```
268  /* Oslobadja se memorija zauzeta imenikom */
    oslobodi_stablo(&koren);
270  return 0;
}
```

### Rešenje 4.17

```
#include<stdio.h>
2  #include<stdlib.h>
  #include<string.h>
4
  #define MAX 51
6
  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8   studenta, ukupan uspeh, uspeh iz matematike, uspeh iz
   maternjeg jezika i redom pokazivace na levo i desno podstablo
10  */
  typedef struct cvor_stabla {
12      char ime[MAX];
      char prezime[MAX];
14      double uspeh;
      double matematika;
16      double jezik;
      struct cvor_stabla *levo;
18      struct cvor_stabla *desno;
  } Cvor;
20
  /* Funkcija kojom se kreira cvor stabla */
22  Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
      double matematika, double jezik)
24  {
      /* Alocira se memorija za novi cvor i proverava se uspesnost
26       alokacije. */
      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
28      if (novi == NULL)
          return NULL;
30
      /* Inicijalizuju se polja strukture */
32      strcpy(novi->ime, ime);
      strcpy(novi->prezime, prezime);
34      novi->uspeh = uspeh;
      novi->matematika = matematika;
36      novi->jezik = jezik;
      novi->levo = NULL;
38      novi->desno = NULL;
40
      /* Vraca se adresa kreiranog cvora */
      return novi;
42  }
```

```
44 /* Funkcija kojom se proverava uspesnost alociranja memorije */
void prover_i_alokaciju(Cvor * novi_cvor)
46 {
    /* Ukoliko je cvor neuspesno kreiran */
48     if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida se izvršavanje
50         programa */
        fprintf(stderr, "Malloc greska za novi cvor!\n");
52         exit(EXIT_FAILURE);
    }
54 }

56 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
void oslobodi_stablo(Cvor ** koren)
58 {
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
60     if (*koren == NULL)
        return;

62     /* Inace ... */
64     /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*koren)->levo);

66     /* Oslobadja se memorija zauzeta desnim podstablom */
    oslobodi_stablo(&(*koren)->desno);

70     /* Oslobadja se memorija zauzeta korenom */
    free(*koren);

72     /* Stablo se proglašava praznim */
74     *koren = NULL;
}

76 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
78                     double uspeh, double matematika,
80                     double jezik)
{
82     /* Ako je stablo prazno */
    if (*koren == NULL) {
        /* Kreira se novi cvor */
        Cvor *novi =
86         napravi_cvor(ime, prezime, uspeh, matematika, jezik);
        prover_i_alokaciju(novi);

88         /* I proglašava se korenom stabla */
90         *koren = novi;

92         return;
    }

94     /* Inace, dodaje se cvor u stablo tako da bude sortirano po
```



```

96     ukupnom broju poena */
97     if (uspeh + matematika + jezik >
98         (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
99         dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
100                      matematika, jezik);
101     else
102         dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
103                      matematika, jezik);
104 }

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     /* Stampaju se informacije iz korenog cvora */
120     if (polozili && koren->matematika + koren->jezik >= 10)
121         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
122              koren->prezime, koren->uspeh, koren->matematika,
123              koren->jezik,
124              koren->uspeh + koren->matematika + koren->jezik);
125     else if (!polozili && koren->matematika + koren->jezik < 10)
126         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
127              koren->prezime, koren->uspeh, koren->matematika,
128              koren->jezik,
129              koren->uspeh + koren->matematika + koren->jezik);

130     /* Stampaju se informacije iz desnog podstabla */
131     stampaj(koren->desno, polozili);
132 }

133 /* Funkcija koja odredjuje koliko studenata nije polozilo
134    prijemni ispit */
135 int nisu_polozili(Cvor * koren)
136 {
137     /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
138     if (koren == NULL)
139         return 0;

140     /* Pretraga se vrši i u levom i u desnom podstablu - ako uslov
141        za polaganje nije ispunjen za koreni cvor, broj studenata
142        se uvecava za 1 */
143     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 }
154
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,
166             "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
167         exit(EXIT_FAILURE);
168     }
169
170     /* Citanje podataka i dodavanje u stablo */
171     koren = NULL;
172     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
173         &matematika, &jezik) != EOF) {
174         dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
175             jezik);
176     }
177
178     /* Zatvaranje datoteke */
179     fclose(in);
180
181     /* Stampaju se prvo podaci o ucenicima koji su polozili
182     prijemni */
183     stampaj(koren, 1);
184
185     /* Linij se iscrtava samo ako postoje ucenici koji nisu
186     polozili prijemni */
187     if (nisu_polozili(koren) != 0)
188         printf("-----\n");
189
190     /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
191     stampaj(koren, 0);
192
193     /* Oslobadja se memorija zauzeta stablom */
194     oslobodi_stablo(&koren);
195
196     return 0;
197 }
```

## Rešenje 4.18

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4
5  #define MAX_NISKA 51
6
7  /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
8     osobe, dan i mesec rođenja i redom pokazivace na levo i desno
9     podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX_NISKA];
12     char prezime[MAX_NISKA];
13     int dan;
14     int mesec;
15     struct cvor_stabla *levo;
16     struct cvor_stabla *desno;
17 } Cvor;
18
19 /* Funkcija koja kreira novi cvor */
20 Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
21 {
22     /* Alocira se memorija */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
26
27     /* Inicijalizuju se polja strukture */
28     strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
30     novi->dan = dan;
31     novi->mesec = mesec;
32     novi->levo = NULL;
33     novi->desno = NULL;
34
35     /* Vraca se adresa novog cvora */
36     return novi;
37 }
38
39 /* Funkcija koja proverava uspesnost alokacije */
40 void prover_i_alokaciju(Cvor * novi_cvor)
41 {
42     /* Ako memorija nije uspesno alocirana */
43     if (novi_cvor == NULL) {
44         /* Ispisuje se poruka i prekida se sa izvršavanjem programa */
45         fprintf(stderr, "Malloc greska za novi cvor!\n");
46         exit(EXIT_FAILURE);
47     }
48 }
49
50 /* Funkcija koja oslobadja memoriju zauzetu stablom */

```

```

void oslobodi_stablo(Cvor ** koren)
52 {
    /* Stablo je prazno */
54     if (*koren == NULL)
        return;

56     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
58     if ((*koren)->levo)
        oslobodi_stablo(&(*koren)->levo);

60     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
62     if ((*koren)->desno)
        oslobodi_stablo(&(*koren)->desno);

64     /* Oslobadja se memorija zauzeta korenom */
66     free(*koren);

68     /* Proglasava se stablo praznim */
    *koren = NULL;
70 }

72 /* Funkcija koja dodaje novi cvor u stablo - stablo treba da bude
    uredjeno po datumu - prvo po mesecu, a zatim po danu */
74 void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
    int dan, int mesec)
76 {
    /* Ako je stablo prazno */
78     if (*koren == NULL) {

80         /* Kreira se novi cvor */
        Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
82         prover_i_alokaciju(novi_cvor);

84         /* I proglasava se korenom */
        *koren = novi_cvor;

86         return;
88     }

90     /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
        meseca */
92     if (mesec < (*koren)->mesec)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
94     else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
96     else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec);
98 }

100 /* Funkcija vrši pretragu stabla i vraća cvor sa traženim datumom.
    */
    Cvor *pretrazi(Cvor * koren, int dan, int mesec)

```

```

102 {
103     /* Stablo je prazno, obustavlja se pretraga */
104     if (koren == NULL)
105         return NULL;
106
107     /* Ako je trazeni datum u korenu */
108     if (koren->dan == dan && koren->mesec == mesec)
109         return koren;
110
111     /* Ako je mesec trazenog datuma manji od meseca sadržanog u korenu
112     ili ako su meseci isti ali je dan trazenog datuma manji od
113     aktuelnog datuma, pretražuje se levo podstablo - pre toga se
114     svakako proverava da li leva grana postoji - ako ne postoji
115     treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
116     */
117     if (mesec < koren->mesec
118         || (mesec == koren->mesec && dan < koren->dan)) {
119         if (koren->levo == NULL)
120             return koren;
121         else
122             return pretrazi(koren->levo, dan, mesec);
123     }
124
125     /* Inace se nastavlja pretraga u desnom delu */
126     return pretrazi(koren->desno, dan, mesec);
127 }
128
129 /* Funkcija koja pronalazi najmanji datum u stablu */
130 Cvor *pronadji_najmanji_datum(Cvor * koren)
131 {
132     /* Stablo je prazno, obustavlja se pretraga */
133     if (koren == NULL)
134         return NULL;
135
136     /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
137     sadrzi najmanji datum */
138     if (koren->levo == NULL)
139         return koren;
140     else
141         /* Inace, trazimo manji datum u levom podstablu */
142         return pronadji_najmanji_datum(koren->levo);
143 }
144
145 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
146 */
147 void datum_u_nisku(int dan, int mesec, char datum[])
148 {
149     if (dan < 10) {
150         datum[0] = '0';
151         datum[1] = dan + '0';
152     } else {
153         datum[0] = dan / 10 + '0';

```

```
152     datum[1] = dan % 10 + '0';
153 }
154 datum[2] = '.';
155
156 if (mesec < 10) {
157     datum[3] = '0';
158     datum[4] = mesec + '0';
159 } else {
160     datum[3] = mesec / 10 + '0';
161     datum[4] = mesec % 10 + '0';
162 }
163 datum[5] = '.';
164 datum[6] = '\\0';
165 }
166
167 int main(int argc, char **argv)
168 {
169     FILE *in;
170     Cvor *koren;
171     Cvor *slavljenik;
172     char ime[MAX_NISKA], prezime[MAX_NISKA];
173     int dan, mesec;
174     char datum[7];
175
176     /* Provera da li je zadato ime ulazne datoteke */
177     if (argc < 2) {
178         /* Ako nije, ispisuje se poruka i prekida se sa izvršavanjem
179            programa */
180         fprintf(stderr, "Nedostaje ime ulazne datoteke!\\n");
181         exit(EXIT_FAILURE);
182     }
183
184     /* Inace, priprema se datoteka za citanje */
185     in = fopen(argv[1], "r");
186     if (in == NULL) {
187         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\\n",
188             argv[1]);
189         exit(EXIT_FAILURE);
190     }
191
192     /* I stablo se popunjava podacima */
193     koren = NULL;
194     while (fscanf
195         (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
196         dodaj_u_stablo(&koren, ime, prezime, dan, mesec);
197
198     /* Datoteka se zatvara */
199     fclose(in);
200
201     /* Omogucuje se pretraga podataka */
202     while (1) {
```

```

204      /* Ucitava se novi datum */
printf("Unesite datum: ");
206      if (scanf("%d.%d.", &dan, &mesec) == EOF)
          break;

208      /* Pretrazuje se stablo */
210      slavljenik = pretrazi(koren, dan, mesec);

212      /* Ispisuju se pronadjeni podaci */

214      /* Ako slavljenik nije pronadjen, to moze znaci da: */
/* 1. drvo je prazno */
216      if (slavljenik == NULL && koren == NULL) {
          printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
          continue;
218      }

220      /* 2. posle datuma koji je unesen, nema podataka u stablu - u
          ovom slucaju se pretraga vrši pocevši od naredne godine i
          ispisuje se najmanji datum */
222      if (slavljenik == NULL) {
          slavljenik = pronadji_najmanji_datum(koren);
          datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
226      printf("Slavljenik: %s %s %s\n", slavljenik->ime,
              slavljenik->prezime, datum);
          continue;
228      }

230      /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
232      /* 1. Pronadjeni su tacni podaci */
      if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
234          printf("Slavljenik: %s %s\n", slavljenik->ime,
              slavljenik->prezime);
          continue;
236      }

238      /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
      datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
      printf("Slavljenik: %s %s %s\n", slavljenik->ime,
242          slavljenik->prezime, datum);
244      }

246      /* Oslobadja se memorija zauzeta stablom */
      oslobodi_stablo(&koren);

248      return 0;
}

```

### 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 alokira 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 koj 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 #endif

```

```

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

4
Cvor *napravi_cvor(int broj)
6 {
   /* Alokira se memorija za novi cvor i proverava se uspesnost
   alokacije. */
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8   if (novi == NULL)
10     return NULL;

```



```
12  /* Inicijalizuju se polja novog cvora. */
    novi->broj = broj;
14  novi->levo = NULL;
    novi->desno = NULL;
16  /* Vraca se adresa novog cvora. */
    return novi;
18 }

20 void prover_i_alokaciju(Cvor * novi_cvor)
{
22  /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
24      /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa
        */
26      fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
28  }
}

30 void dodaj_u_stablo(Cvor ** koren, int broj)
32 {
    /* Ako je stablo prazno */
34    if (*koren == NULL) {
        /* Kreira se novi cvor */
36        Cvor *novi = napravi_cvor(broj);
        prover_i_alokaciju(novi);
38        /* I proglašava se korenom stabla */
        *koren = novi;
40        return;
    }
42    /* U suprotnom se trazi odgovarajuca pozicija za zadati broj: */

44    /* Ako je zadata vrednost manja od vrednosti korena */
    if (broj < (*koren)->broj)
46        /* Broj se dodaje u levo podstablo */
        dodaj_u_stablo(&(*koren)->levo, broj);
48    else
        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
50        dodaje u desno podstablo */
        dodaj_u_stablo(&(*koren)->desno, broj);
52 }

54 Cvor *pretrazi_stablo(Cvor * koren, int broj)
{
56    /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
    if (koren == NULL)
58        return NULL;
    /* Ako je trazena vrednost sadržana u korenu */
60    if (koren->broj == broj) {
        /* Prekida se pretraga */
62        return koren;
    }
}
```

```
64  /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
    if (broj < koren->broj)
66      /* Pretraga se nastavlja u levom podstablu */
        return pretrazi_stablo(koren->levo, broj);
68  else
        /* U suprotnom, pretraga se nastavlja u desnom podstablu */
70      return pretrazi_stablo(koren->desno, broj);
    }

72  Cvor *pronadji_najmanji(Cvor * koren)
73  {
74      /* Ako je stablo prazno, prekida se pretraga */
75      if (koren == NULL)
76          return NULL;
77
78      /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
79      levo od njega */
80
81      /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
82      najmanju vrednost */
83      if (koren->levo == NULL)
84          return koren;
85
86      /* Inace, pretragu treba nastaviti u levom podstablu */
87      return pronadji_najmanji(koren->levo);
88  }

90  Cvor *pronadji_najveci(Cvor * koren)
91  {
92      /* Ako je stablo prazno, prekida se pretraga */
93      if (koren == NULL)
94          return NULL;
95
96      /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
97      desno od njega */
98
99      /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
100      najveću vrednost */
101      if (koren->desno == NULL)
102          return koren;
103
104      /* Inace, pretragu treba nastaviti u desnom podstablu */
105      return pronadji_najveci(koren->desno);
106  }

108  void obrisi_element(Cvor ** adresa_korena, int broj)
109  {
110      Cvor *pomocni_cvor = NULL;
111
112      /* Ako je stablo prazno, brisanje nije primenljivo */
113      if (*adresa_korena == NULL)
114          return;
```

```
116      /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
118      stabla, ona se eventualno nalazi u levom podstablu, pa treba
120      rekurzivno primeniti postupak na levo podstablo. Koren ovako
      modifikovanog stabla je nepromenjen. */
122      if (broj < (*adresa_korena)->broj) {
      obrisi_element(&(*adresa_korena)->levo, broj);
      return;
124  }

126      /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
128      stabla, ona se eventualno nalazi u desnom podstablu pa treba
      rekurzivno primeniti postupak na desno podstablo. Koren ovako
      modifikovanog stabla je nepromenjen. */
130      if ((*adresa_korena)->broj < broj) {
      obrisi_element(&(*adresa_korena)->desno, broj);
132      return;
      }

134      /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
136      jednaka broju koji se brise (tj. slucaj kada treba obrisati
      koren) */

138      /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
      prazno stablo (vraca se NULL) */
140      if ((*adresa_korena)->levo == NULL
      && (*adresa_korena)->desno == NULL) {
142          free(*adresa_korena);
144          *adresa_korena = NULL;
      return;
146  }

148      /* Ako koren ima samo levog sina, tada se brisanje vrši tako sto se
      brise koren, a novi koren postaje levi sin */
150      if ((*adresa_korena)->levo != NULL
      && (*adresa_korena)->desno == NULL) {
152          pomocni_cvor = (*adresa_korena)->levo;
      free(*adresa_korena);
154          *adresa_korena = pomocni_cvor;
      return;
156  }

158      /* Ako koren ima samo desnog sina, tada se brisanje vrši tako sto
      se brise koren, a novi koren postaje desni sin */
160      if ((*adresa_korena)->desno != NULL
      && (*adresa_korena)->levo == NULL) {
162          pomocni_cvor = (*adresa_korena)->desno;
      free(*adresa_korena);
164          *adresa_korena = pomocni_cvor;
      return;
166  }
```

```

168  /* Slučaj kada koren ima oba sina - najpre se potrazi sledbenik
170     korena (u smislu poretka) u stablu. To je upravo po vrednosti
172     najmanji cvor u desnom podstablu. On se može pronaci npr.
174     funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
176     vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
178     broj koji se brise). Zatim se prosto rekursivno pozove funkcija
180     za brisanje na desno podstablo. S obzirom da u njemu treba
182     obrisati najmanji element, a on zasigurno ima najviše jednog
184     potomka, jasno je da će njegovo brisanje biti obavljeno na jedan
186     od jednostavnijih načina koji su gore opisani. */
188  pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
190  (*adresa_korena)->broj = pomocni_cvor->broj;
192  pomocni_cvor->broj = broj;
194  obrisi_element(&(*adresa_korena)->desno, broj);
196  }

198  void prikazi_stablo(Cvor * koren)
199  {
200      /* Ako je stablo prazno, prekida se ispis */
201      if (koren == NULL)
202          return;
203
204      /* Prvo se ispisuju svi cvorovi levo od korena */
205      prikazi_stablo(koren->levo);
206
207      /* Zatim se ispisuje vrednost u korenu */
208      printf("%d ", koren->broj);
209
210      /* Na kraju se ispisuju svi cvorovi desno od korena */
211      prikazi_stablo(koren->desno);
212  }

214  void oslobodi_stablo(Cvor ** koren)
215  {
216      /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
217      if (*koren == NULL)
218          return;
219      /* Inace ... */
220      /* Oslobadja se memorija zauzeta levim podstablom */
221      if ((*koren)->levo)
222          oslobodi_stablo(&(*koren)->levo);
223      /* Oslobadja se memorija zauzeta desnim podstablom */
224      if ((*koren)->desno)
225          oslobodi_stablo(&(*koren)->desno);
226      /* Oslobadja se memorija zauzeta korenom */
227      free(*koren);
228      /* Proglasava se stablo praznim */
229      *koren = NULL;
230  }

```

```

#include<stdio.h>
#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 */
       if (koren1 == NULL || koren2 == NULL)
20         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     /* Inace, 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     /* Učitavaju se elementi prvog stabla */
       koren1 = NULL;
       printf("Prvo stablo: ");
       scanf("%d", &broj);
       while (broj != 0) {
42         dodaj_u_stablo(&koren1, broj);
           scanf("%d", &broj);
       }

46     /* Učitavaju se elementi drugog stabla */
       koren2 = NULL;
       printf("Drugo stablo: ");
       scanf("%d", &broj);
       while (broj != 0) {
50         dodaj_u_stablo(&koren2, broj);
           scanf("%d", &broj);
       }
54 }
```

```

56  /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
    njen rezultat. */
58  if (identitet(koren1, koren2))
    printf("Stabla jesu identicna.\n");
60  else
    printf("Stabla nisu identicna.\n");
62
    /* Oslobadja se memorija zauzeta stablima */
64  oslobodi_stablo(&koren1);
    oslobodi_stablo(&koren2);
66
    return 0;
68 }

```

### Rešenje 4.20

```

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

/* Uklucuje se biblioteka za rad sa stablima */
#include "stabla.h"

6  /* Funkcija kreira novo stablo identicno stablu koje je dato korenom.
   */
8  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 stabla
       */
18     *duplikat = napravi_cvor(koren->broj);
        prover_i_alokaciju(*duplikat);

20     /* Rekurzivno se duplira levo podstablo i njegova adresa se cuva u
       pokazivacu na levo podstablo korena duplikata. */
22     kopiraj_stablo(koren->levo, &(*duplikat)->levo);

24     /* Rekurzivno se duplira desno podstablo i njegova adresa se cuva u
       pokazivacu na desno podstablo korena duplikata. */
26     kopiraj_stablo(koren->desno, &(*duplikat)->desno);
28 }

30 /* Funkcija izracunava uniju dva stabla - rezultuje stablo se
   dobija modifikacijom prvog stabla */
32 void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
{

```

```

34  /* Ako drugo stablo nije prazno */
    if (koren2 != NULL) {
36      /* Dodaje se njegov koren u prvo stablo */
      dodaj_u_stablo(adresa_korena1, koren2->broj);

38      /* Rekurzivno se racuna unija levog i desnog podstabla drugog
40         stabla sa prvim stablom */
      kreiraj_uniju(adresa_korena1, koren2->levo);
42      kreiraj_uniju(adresa_korena1, koren2->desno);
    }
44 }

46 /* Funkcija izracunava presek dva stabla - rezultujuce stablo se
    dobija modifikacijom prvog stabla */
48 void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
    {
50     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
    if (*adresa_korena1 == NULL)
52         return;

54     /* Inace... */
    /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.
56     iz levog i desnog podstabla prvog stabla brisu se svi oni
        elementi koji ne postoje u drugom stablu */
58     kreiraj_presek(&(*adresa_korena1)->levo, koren2);
    kreiraj_presek(&(*adresa_korena1)->desno, koren2);

60     /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on
62        uklanja iz prvog stabla */
    if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
64        obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
    }

66 /* Funkcija izracunava razliku dva stabla - rezultujuce stablo se
    dobija modifikacijom prvog stabla */
68 void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
    {
70     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
    if (*adresa_korena1 == NULL)
72         return;

74     /* Inace... */
    /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
76     tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
        elementi koji postoje i u drugom stablu */
78     kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
    kreiraj_razliku(&(*adresa_korena1)->desno, koren2);

80     /* Ako se koren prvog stabla nalazi i u drugom stablu tada se isti
        uklanja iz prvog stabla */
82     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
84        obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
    }

```

```
86 }
87
88 int main()
89 {
90     Cvor *koren1;
91     Cvor *koren2;
92     Cvor *pomocni = NULL;
93     int n;
94
95     /* Ucitavaju se elementi prvog stabla */
96     koren1 = NULL;
97     printf("Prvo stablo: ");
98     while (scanf("%d", &n) != EOF) {
99         dodaj_u_stablo(&koren1, n);
100     }
101
102     /* Ucitavaju se elementi drugog stabla */
103     koren2 = NULL;
104     printf("Drugo stablo: ");
105     while (scanf("%d", &n) != EOF) {
106         dodaj_u_stablo(&koren2, n);
107     }
108
109     /* Kreira se unija stabala: prvo se napravi kopija prvog stabla
110        kako bi se isto moglo iskoristiti i za preostale operacije */
111     kopiraj_stablo(koren1, &pomocni);
112     kreiraj_uniju(&pomocni, koren2);
113     printf("Unija: ");
114     prikazi_stablo(pomocni);
115     putchar('\n');
116
117     /* Oslobadja se stablo sa rezultatom operacije */
118     oslobodi_stablo(&pomocni);
119
120     /* Kreira se presek stabala: prvo se napravi kopija prvog stabla
121        kako bi se isto moglo iskoristiti i za preostale operacije */
122     kopiraj_stablo(koren1, &pomocni);
123     kreiraj_presek(&pomocni, koren2);
124     printf("Presek: ");
125     prikazi_stablo(pomocni);
126     putchar('\n');
127
128     /* Oslobadja se stablo sa rezultatom operacije */
129     oslobodi_stablo(&pomocni);
130
131     /* Kreira se razlika stabala: prvo se napravi kopija prvog stabla
132        kako bi se isto moglo iskoristiti i za preostale operacije */
133     kopiraj_stablo(koren1, &pomocni);
134     kreiraj_razliku(&pomocni, koren2);
135     printf("Razlika: ");
136     prikazi_stablo(pomocni);
137     putchar('\n');
```



```

138      /* Oslobadja se stablo sa rezultatom operacije */
140      oslobodi_stablo(&pomocni);

142      /* Oslobadjaju se i polazna stabla */
      oslobodi_stablo(&koren2);
144      oslobodi_stablo(&koren1);

146      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 vrednosti koje
11  su smestene u niz. */
   int kreiraj_niz(Cvor * koren, int a[])
13  {
       int r, s;

15

       /* Stablo je prazno - u niz je smesteno 0 elemenata */
17       if (koren == NULL)
           return 0;

19

       /* Dodaju se u niz elementi iz levog podstabla */
21       r = kreiraj_niz(koren->levo, a);

23

       /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
          u niz i na osnovu nje se moze odrediti indeks novog elementa */

25

       /* Smesta se vrednost iz korena */
27       a[r] = koren->broj;

29

       /* Dodaju se elementi iz desnog podstabla */
       s = kreiraj_niz(koren->desno, a + r + 1);

31

       /* Racuna se indeks na koji treba smestiti naredni element */
33       return r + s + 1;
   }

35

   /* Funkcija sortira niz tako sto najpre elemente niza smesti u
37   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva u
       desno.

```

```
39      Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
41      sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
43      sortiranje vrši u pomocnoj dinamičkoj strukturi, a ne razmenom
      elemenata niza. */
void sortiraj(int a[], int n)
45 {
      int i;
47      Cvor *koren;

49      /* Kreira se stablo smestanjem elemenata iz niza u stablo */
      koren = NULL;
51      for (i = 0; i < n; i++)
          dodaj_u_stablo(&koren, a[i]);

53      /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
55      a */
      kreiraj_niz(koren, a);

57      /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
59      */
      oslobodi_stablo(&koren);
}

61 int main()
63 {
      int a[MAX];
65      int n, i;

67      /* Ucitavaju se dimenzija i elementi niza */
      printf("n: ");
69      scanf("%d", &n);
      if (n < 0 || n > MAX) {
71          printf("Greska: pogresna dimenzija niza!\n");
          return 0;
73      }

75      printf("a: ");
      for (i = 0; i < n; i++)
77          scanf("%d", &a[i]);

79      /* Poziva se funkcija za sortiranje */
      sortiraj(a, n);

81      /* Ispisuje se rezultat */
      for (i = 0; i < n; i++)
83          printf("%d ", a[i]);
85      printf("\n");

87      return 0;
}
```

## Rešenje 4.22

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* a) Funkcija koja izracunava broj cvorova stabla */
8 int broj_cvorova(Cvor * koren)
9 {
10     /* Ako je stablo prazno, broj cvorova je nula */
11     if (koren == NULL)
12         return 0;
13
14     /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
15        levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
16        zato sto treba racunati i koren */
17     return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
18 }
19
20 /* b) Funkcija koja izracunava broj listova stabla */
21 int broj_listova(Cvor * koren)
22 {
23     /* Ako je stablo prazno, broj listova je nula */
24     if (koren == NULL)
25         return 0;
26
27     /* Proverava se da li je tekuci cvor list */
28     if (koren->levo == NULL && koren->desno == NULL)
29         /* Ako jeste vraća se 1 - to će kasnije zbog rekurzivnih poziva
30            uvećati broj listova za 1 */
31         return 1;
32
33     /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
34        */
35     return broj_listova(koren->levo) + broj_listova(koren->desno);
36 }
37
38 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
39 void pozitivni_listovi(Cvor * koren)
40 {
41     /* Slučaj kada je stablo prazno */
42     if (koren == NULL)
43         return;
44
45     /* Ako je cvor list i sadrži pozitivnu vrednost */
46     if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
47         /* Stampa se */
48         printf("%d ", koren->broj);
49
50     /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
51 }
```

```
51     pozitivni_listovi(koren->levo);
52     pozitivni_listovi(koren->desno);
53 }

54
55 /* d) Funkcija koja izracunava zbir cvorova stabla */
56 int zbir_svih_cvorova(Cvor * koren)
57 {
58     /* Ako je stablo prazno, zbir cvorova je 0 */
59     if (koren == NULL)
60         return 0;
61
62     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
63        elemenata u podstablama */
64     return koren->broj + zbir_svih_cvorova(koren->levo) +
65            zbir_svih_cvorova(koren->desno);
66 }
67
68 /* e) Funkcija koja izracunava najveći element stabla */
69 Cvor *najveci_element(Cvor * koren)
70 {
71     /* Ako je stablo prazno, obustavlja se pretraga */
72     if (koren == NULL)
73         return NULL;
74
75     /* Zbog prirode pretrazivackog stabla, vrednosti vece od korena se
76        nalaze u desnom podstablu */
77
78     /* Ako desnog podstabla nema */
79     if (koren->desno == NULL)
80         /* Najveća vrednost je koren */
81         return koren;
82
83     /* Inace, najveća vrednost se traži desno */
84     return najveci_element(koren->desno);
85 }
86
87 /* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
89 {
90     /* Dubina praznog stabla je 0 */
91     if (koren == NULL)
92         return 0;
93
94     /* Izracunava se dubina levog podstabla */
95     int dubina_levo = dubina_stabla(koren->levo);
96
97     /* Izracunava se dubina desnog podstabla */
98     int dubina_desno = dubina_stabla(koren->desno);
99
100    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
101       jer se racuna i koren */
102    return dubina_levo >
```

```
103     dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
104 }
105
106 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
107 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
108 {
109     /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazenog
110        nivoa */
111
112     /* Ako nema vise cvorova, nema spustanja niz stablo */
113     if (koren == NULL)
114         return 0;
115
116     /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
117        rekurzivnih poziva uvecati broj cvorova za 1 */
118     if (i == 0)
119         return 1;
120
121     /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
122        */
123     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
124         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
125 }
126
127 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
128 void ispis_nivo(Cvor * koren, int i)
129 {
130     /* Ideja je slicna ideji iz prethodne funkcije */
131
132     /* Nema vise cvorova, nema spustanja kroz stablo */
133     if (koren == NULL)
134         return;
135
136     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
137     if (i == 0) {
138         printf("%d ", koren->broj);
139         return;
140     }
141     /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
142        desnom podstablu */
143     ispis_nivo(koren->levo, i - 1);
144     ispis_nivo(koren->desno, i - 1);
145 }
146
147 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
148    stabla */
149 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
150 {
151     /* Ako je stablo prazno, obustavlja se pretraga */
152     if (koren == NULL)
153         return NULL;
```

```
155  /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
156  if (i == 0)
157      return koren;
158
159  /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
160  Cvor *a = najveći_element_na_itom_nivou(koren->levo, i - 1);
161
162  /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
163  Cvor *b = najveći_element_na_itom_nivou(koren->desno, i - 1);
164
165  /* Trazi se i vraća maksimum izracunatih vrednosti */
166  if (a == NULL && b == NULL)
167      return NULL;
168  if (a == NULL)
169      return b;
170  if (b == NULL)
171      return a;
172  return a->broj > b->broj ? a : b;
173 }
174
175 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
176 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
177 {
178     /* Ako je stablo prazno, zbir je nula */
179     if (koren == NULL)
180         return 0;
181
182     /* Ako se stiglo do trazenog nivoa, vraća se vrednost */
183     if (i == 0)
184         return koren->broj;
185
186     /* Inace, spustanje se nastavlja za jedan nivo nize i traze se sume
187     iz levog i desnog podstabla */
188     return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
189         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
190 }
191
192 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
193 manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
195 {
196     /* Ako je stablo prazno, zbir je nula */
197     if (koren == NULL)
198         return 0;
199
200     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
201     prirode pretrazivackog stabla treba obici i levo i desno
202     podstablo */
203     if (koren->broj <= x)
204         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
205             zbir_manjih_od_x(koren->desno, x);
206 }
```

```

207  /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
    medju njima jedino moze biti onih koje zadovoljavaju uslov */
209  return zbir_manjih_od_x(koren->levo, x);
    }
211
213  int main(int argc, char **argv)
    {
        /* Analiza argumenata komandne linije */
215  if (argc != 3) {
        fprintf(stderr,
217             "Greska! Program se poziva sa: ./a.out nivo
        broj_za_pretragu\n");
        exit(EXIT_FAILURE);
219  }
        int i = atoi(argv[1]);
221  int x = atoi(argv[2]);

223  /* Kreira se stablo */
        Cvor *koren = NULL;
225  int broj;
        while (scanf("%d", &broj) != EOF)
227      dodaj_u_stablo(&koren, broj);

229  /* ispisuju se rezultati rada funkcija */
        printf("Broj cvorova: %d\n", broj_cvorova(koren));
231  printf("Broj listova: %d\n", broj_listova(koren));
        printf("Pozitivni listovi: ");
233  pozitivni_listovi(koren);
        printf("\n");
235  printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
        if (najveci_element(koren) == NULL)
237      printf("Najveci element: ne postoji\n");
        else
239      printf("Najveci element: %d\n", najveci_element(koren)->broj);

241  printf("Dubina stabla: %d\n", dubina_stabla(koren));

243  printf("Broj cvorova na %d. nivou: %d\n", i,
        broj_cvorova_na_itom_nivou(koren, i));
245  printf("Elementi na %d. nivou: ", i);
        ispis_nivo(koren, i);
247  printf("\n");
        if (najveci_element_na_itom_nivou(koren, i) == NULL)
249      printf("Nema elemenata na %d. nivou!\n", i);
        else
251      printf("Maksimalni element na %d. nivou: %d\n", i,
        najveci_element_na_itom_nivou(koren, i)->broj);

253  printf("Zbir elemenata na %d. nivou: %d\n", i,
255      zbir_cvorova_na_itom_nivou(koren, i));
        printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,

```

```

257         zbir_manjih_od_x(koren, x));
259     /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
261     return 0;
263 }

```

### 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 */
    int dubina_desno = dubina_stabla(koren->desno);

20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
        jer se racuna i koren */
22     return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }

26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
28 {
    /* Ideja je slicna ideji iz prethodne funkcije */
30     /* Nema vise cvorova, nema spustanja niz stablo */
    if (koren == NULL)
32         return;

34     /* Ako se stiglo do trazeneog nivoa - ispisuje se vrednost */
    if (i == 0) {
36         printf("%d ", koren->broj);
        return;
38     }
    /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
        podstablu */
40     ispisi_nivo(koren->levo, i - 1);

```



```

42     ispisi_nivo(koren->desno, i - 1);
43 }
44
45 /* Funkcija koja ispisuje stablo po nivoima */
46 void ispisi_stablo_po_nivoima(Cvor * koren)
47 {
48     int i;
49
50     /* Prvo se izracunava dubina stabla */
51     int dubina;
52     dubina = dubina_stabla(koren);
53
54     /* Ispisuje se nivo po nivo stabla */
55     for (i = 0; i < dubina; i++) {
56         printf("%d. nivo: ", i);
57         ispisi_nivo(koren, i);
58         printf("\n");
59     }
60 }
61
62 int main(int argc, char **argv)
63 {
64     Cvor *koren;
65     int broj;
66
67     /* Citaju se vrednosti sa ulaza i dodaju se u stablo */
68     koren = NULL;
69     while (scanf("%d", &broj) != EOF) {
70         dodaj_u_stablo(&koren, broj);
71     }
72
73     /* Ispisuje se stablo po nivoima */
74     ispisi_stablo_po_nivoima(koren);
75
76     /* Oslobadja se memorija zauzeta stablom */
77     oslobodi_stablo(&koren);
78
79     return 0;
80 }

```

### Rešenje 4.25

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija koja izracunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
9  {

```

```
11  /* Dubina praznog stabla je 0 */
12  if (koren == NULL)
13      return 0;
14
15  /* Izracunava se dubina levog podstabla */
16  int dubina_levo = dubina_stabla(koren->levo);
17
18  /* Izracunava se dubina desnog podstabla */
19  int dubina_desno = dubina_stabla(koren->desno);
20
21  /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
22     jer se racuna i koren */
23  return dubina_levo >
24         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
25 }
26
27 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
28    stablo */
29 int avl(Cvor * koren)
30 {
31     int dubina_levo, dubina_desno;
32
33     /* Ako je stablo prazno, zaustavlja se brojanje */
34     if (koren == NULL) {
35         return 0;
36     }
37
38     /* Izracunava se dubina levog podstabla korena */
39     dubina_levo = dubina_stabla(koren->levo);
40
41     /* Izracunava se dubina desnog podstabla korena */
42     dubina_desno = dubina_stabla(koren->desno);
43
44     /* Ako je uslov za AVL stablo ispunjen */
45     if (abs(dubina_desno - dubina_levo) <= 1) {
46         /* Racuna se broj AVL cvorova u levom i desnom podstablu i
47            uvecava za jedan iz razloga sto koren ispunjava uslov */
48         return 1 + avl(koren->levo) + avl(koren->desno);
49     } else {
50         /* Inace, racuna se samo broj AVL cvorova u podstablama */
51         return avl(koren->levo) + avl(koren->desno);
52     }
53 }
54
55 int main(int argc, char **argv)
56 {
57     Cvor *koren;
58     int broj;
59
60     /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo */
61     koren = NULL;
62     while (scanf("%d", &broj) != EOF) {
```

```

        dodaj_u_stablo(&koren, broj);
63     }

65     /* Racuna se i ispisuje broj AVL cvorova */
    printf("%d\n", avl(koren));

67     /* Oslobadja se memorija zauzeta stablom */
69     oslobodi_stablo(&koren);

71     return 0;
}

```

### Rešenje 4.26

```

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

4  /* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

6  /* Funkcija proverava da li je zadato binarno stablo celih
8     pozitivnih brojeva hip. Ideja koja ce biti implementirana u
osnovi ima pronalazenje maksimalne vrednosti levog i
10    maksimalne vrednosti desnog podstabla - ako je vrednost u
korenu veca od izracunatih vrednosti, uoceni fragment stabla
12    zadovoljava uslov za hip. Zato ce funkcija vratiti maksimalne
vrednosti iz uocenog podstabala ili vrednost -1 ukoliko se
14    zakljuci da stablo nije hip. */
int heap(Cvor * koren)
16 {
    int max_levo, max_desno;

18     /* Prazno sablo je hip - kao rezultat se vraca 0 kao najmanji
20     pozitivan broj */
    if (koren == NULL) {
22         return 0;
    }

24     /* Ukoliko je stablo list... */
    if (koren->levo == NULL && koren->desno == NULL) {
26         /* Vraca se njegova vrednost */
        return koren->broj;
    }

28     /* Inace... */

30     /* Proverava se svojstvo za levo podstablo. */
    max_levo = heap(koren->levo);

32     /* Proverava se svojstvo za desno podstablo. */
    max_desno = heap(koren->desno);

34     /* Ako levo ili desno podstablo uocenog cvora nije hip, onda
36

```

```
38     nije ni celo stablo. */
39     if (max_levo == -1 || max_desno == -1) {
40         return -1;
41     }
42
43     /* U suprotnom proverava se da li svojstvo vazi za uoceni
44        cvor. */
45     if (koren->broj > max_levo && koren->broj > max_desno) {
46         /* Ako vazi, vraca se vrednost korena */
47         return koren->broj;
48     }
49
50     /* U suprotnom zakljucuje se da stablo nije hip */
51     return -1;
52 }
53
54 int main(int argc, char **argv)
55 {
56     Cvor *koren;
57     int hip_indikator;
58
59     /* Kreira se stablo koje sadrzi brojeve 100 19 36 17 3 25 1 2
60        7 */
61     koren = NULL;
62     koren = napravi_cvor(100);
63     koren->levo = napravi_cvor(19);
64     koren->levo->levo = napravi_cvor(17);
65     koren->levo->levo->levo = napravi_cvor(2);
66     koren->levo->levo->desno = napravi_cvor(7);
67     koren->levo->desno = napravi_cvor(3);
68     koren->desno = napravi_cvor(36);
69     koren->desno->levo = napravi_cvor(25);
70     koren->desno->desno = napravi_cvor(1);
71
72     /* Poziva se funkcija kojom se proverava da li je stablo hip */
73     hip_indikator = heap(koren);
74
75     /* Ispisuje se rezultat */
76     if (hip_indikator == -1) {
77         printf("Zadato stablo nije hip!\n");
78     } else {
79         printf("Zadato stablo je hip!\n");
80     }
81
82     /* Oslobadja se memorija zauzeta stablom. */
83     oslobodi_stablo(&koren);
84
85     return 0;
86 }
```

## 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
```

```
ULAZ.TXT
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
```

```
INTERAKCIJA PROGRAMA:
```

```
Ispit
Matematika
Programiranje
```

#### *Primer 2*

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

```
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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<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>	<pre> ULAZ: 2 3 0 0 -5 1 2 -4 IZLAZ: 2 </pre>	<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  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 .  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 učitava 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*

```

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*

```

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 4*

```

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 Programiranje 2, praktični deo ispita, septembar 2015.

[illegible]

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 6 1	ULAZ: 15 3	ULAZ: 31 100
IZLAZ: 3	IZLAZ: 3758096385	IZLAZ: 4026531841

[Rešenje 5.7]

**Zadatak 5.8** Napisati funkciju `void dopuni_listu(Cvor** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

*Test 1*

```

|| ULAZ:
|| 1 2 3 4 5
|| IZLAZ:
|| 1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00

```

*Test 2*

```

|| ULAZ:
|| 12
|| IZLAZ:
|| 12.00

```

*Test 3*

```

|| ULAZ:
|| prazna lista
|| IZLAZ:

```

*Test 4*

```

|| ULAZ:
|| 13.3 15.8
|| IZLAZ:
|| 13.30 14.55

```

[Rešenje 5.8]

**Zadatak 5.9** Sa standardnog ulaza se učitava dimenzija  $n$  kvadratne celobrojne matrice  $A$  ( $n > 0$ ), a zatim i elementi matrice  $A$ . Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje je suma brojeva u svakom redu i svakoj koloni jednaka). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati "-". Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati -1.

*Test 1*

```

|| ULAZ:
|| 4
|| 1 2 3 4
|| 2 1 4 3
|| 3 4 2 1
|| 4 3 1 2
|| IZLAZ:
|| 10

```

*Test 2*

```

|| ULAZ:
|| 3
|| 1 1 1
|| 1 1 1
|| 1 1 1
|| IZLAZ:
|| 3

```

*Test 3*

```

|| ULAZ:
|| 2
|| 1 1
|| 2 2
|| IZLAZ:
|| -

```

*Test 4*

```

|| ULAZ:
|| 2
|| 1 2
|| 1 2
|| IZLAZ:
|| -

```

*Test 5*

```

|| ULAZ:
|| 1
|| 5
|| IZLAZ:
|| 5

```

*Test 6*

```

|| ULAZ:
|| 0
|| IZLAZ:
|| -1

```

[Rešenje 5.9]

## 5.4 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 komandne
24        linije neposredno nakon imena programa koji se poziva. */
25     ulaz = fopen(argv[1], "r");
26     if (ulaz == NULL) {
27         greska();
28     }
29
30     /* Ucitavanje broja linija. */
31     fscanf(ulaz, "%d", &n);
32
33     /* Alociranje memorije na osnovu ucitanog broja linija. */
34     linije = (char **) malloc(n * sizeof(char *));
35     if (linije == NULL) {
36         greska();
37     }
38     for (i = 0; i < n; i++) {
39         linije[i] = malloc(MAX * sizeof(char));
40         if (linije[i] == NULL) {
41             for (j = 0; j < i; j++) {
42                 free(linije[j]);
43             }
44             free(linije);
45             greska();
46         }
47     }
```

```

    }
48 }

50 /* Ucitavanje svih n linija iz datoteke. */
51 for (i = 0; i < n; i++) {
52     fscanf(ulaz, "%s", linije[i]);
53 }
54
55 /* Ispisivanje u odgovarajucem poretку ucitane linije koje
56    zadovoljavaju kriterijum. */
57 for (i = n - 1; i >= 0; i--) {
58     if (isupper(linije[i][0])) {
59         printf("%s\n", linije[i]);
60     }
61 }
62
63 /* Oslobadjanje memorije koja je dinamicki alocirana. */
64 for (i = 0; i < n; i++) {
65     free(linije[i]);
66 }
67
68 free(linije);
69
70 /* Zatvaranje datoteku. */
71 fclose(ulaz);
72
73 return 0;
74 }

```

## Rešenje 5.2

```

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 alokira 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 dinamicki alocirani prostor za stablo Nakon
15    oslobadjanja se u pozivajucoj funkciji koren postavljana NULL, jer
16    je stablo prazno */
17 void oslobodi_stablo(Cvor ** adresa_korena);
18
19 /* Funkcija proverava da li je novi Cvor ispravno alocirani, i nakon
20    toga prekida program */

```

```
void prover_i_alokaciju(Cvor * novi);
22
/* Funkcija dodaje nov Cvor u stablo i azurira vrednost korena stabla
24   u pozivajucoj funkciji. */
void dodaj_u_stablo(Cvor ** adresa_korena, int broj);
26
#endif

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

5  Cvor *napravi_cvor(int b)
   {
7      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
9         return NULL;

   /* Inicijalizacija polja novog Cvora */
   novi->broj = b;
13  novi->levo = NULL;
   novi->desno = NULL;

15  return novi;
17 }

19 void oslobodi_stablo(Cvor ** adresa_korena)
   {
21     /* Prazno stablo i nema sta da se oslobadja */
       if (*adresa_korena == NULL)
23         return;

   /* Rekurzivno se oslobadja najpre levo, a onda i desno podstablo */
   if ((*adresa_korena)->levo)
25     oslobodi_stablo(&(*adresa_korena)->levo);
   if ((*adresa_korena)->desno)
27     oslobodi_stablo(&(*adresa_korena)->desno);

31     free(*adresa_korena);
       *adresa_korena = NULL;
33 }

35 void prover_i_alokaciju(Cvor * novi)
   {
37     if (novi == NULL) {
         fprintf(stderr, "Malloc greska za nov cvor!\n");
39         exit(EXIT_FAILURE);
       }
41 }

43 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
   {
```

```

45  /* Postojeće stablo je prazno */
46  if (*adresa_korena == NULL) {
47      Cvor *novi = napravi_cvor(broj);
48      prover_i_alokaciju(novi);
49      /* Kreirani Cvor novi ce biti od sada koren stabla */
50      *adresa_korena = novi;
51      return;
52  }
53
54  /* Brojevi se smestaju u uredjeno binarno stablo, pa ako je broj
55     koji se ubacuje manji od broja koji je u korenu onda se dodaje u
56     levo podstablo. */
57  if (broj < (*adresa_korena)->broj)
58      dodaj_u_stablo(&(*adresa_korena)->levo, broj);
59  /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
60     dodaje se nov Cvor desno od korena. */
61  else
62      dodaj_u_stablo(&(*adresa_korena)->desno, broj);
63  }

```

```

1  #include <stdio.h>
2  #include "stabla.h"
3
4  int sumirajN(Cvor * koren, int n)
5  {
6      if (koren == NULL)
7          return 0;
8
9      if (n == 0)
10         return koren->broj;
11
12     return sumirajN(koren->levo, n - 1) + sumirajN(koren->desno, n - 1)
13         ;
14 }
15
16 int main()
17 {
18     Cvor *koren = NULL;
19     int n;
20     int nivo;
21
22     scanf("%d", &nivo);
23
24     while (1) {
25         scanf("%d", &n);
26         /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
27         if (n == 0)
28             break;
29
30         /* Ako nije, dodaje se procitani broj u stablo. */
31         dodaj_u_stablo(&koren, n);
32     }

```

```
33  /* Ispisuje se rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren, nivo));
35
    /* Oslobadja se memorija */
37    oslobodi_stablo(&koren);
39    return 0;
}
```

### Rešenje 5.3

```
#include <stdio.h>
2  #define MAX 50

4  int main()
    {
6      int m[MAX][MAX];
        int v, k;
8      int i, j;
        int max_broj_negativnih, max_indeks_kolone;
10     int broj_negativnih;

12     /* Ucitavanje dimenzije matrice */
        scanf("%d", &v);
14     if (v < 0 || v > MAX) {
            fprintf(stderr, "-1\n");
16         return 0;
        }

18     scanf("%d", &k);
20     if (k < 0 || k > MAX) {
            fprintf(stderr, "-1\n");
22         return 0;
        }

24     /* Ucitavanje elemenata matrice */
26     for (i = 0; i < v; i++) {
        for (j = 0; j < k; j++) {
28         scanf("%d", &m[i][j]);
        }
30     }

32     /* Pronalazenje kolone koja sadrzi najveći broj negativnih
        elemenata */
34     max_indeks_kolone = 0;

36     max_broj_negativnih = 0;
        for (i = 0; i < v; i++) {
38         if (m[i][0] < 0) {
            max_broj_negativnih++;
        }
    }
```

```

40     }
42 }

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 }

56 /* Ispisivanje trazenog rezultata */
57 printf("%d\n", max_indeks_kolone);

60 return 0;
}

```

### Rešenje 5.4

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 128

int main(int argc, char **argv)
{
    FILE *f;
    int brojac = 0;
    char linija[MAX], *p;

    if (argc != 3) {
        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
    }

    /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
       linije */
    if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
    }

    while (fgets(linija, MAX, f) != NULL) {
        p = linija;
        while (1) {

```



```

28     p = strstr(p, argv[2]);
    if (p == NULL)
        break;
30     brojac++;
    p = p + strlen(argv[2]);
32 }
}
34
36 fclose(f);
38 printf("%d\n", brojac);
40 return 0;
}

```

### Rešenje 5.5

```

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

5  /* Struktura trougao */
   typedef struct _trougao {
7      double xa, ya, xb, yb, xc, yc;
   } trougao;

9
11 /* Funkcija racuna duzinu duzi */
   double duzina(double x1, double y1, double x2, double y2)
   {
13     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
   }

15
17 /* Funkcija racuna povrsinu trougla */
   double povrsina(trougao t)
   {
19     double a = duzina(t.xb, t.yb, t.xc, t.yc);
     double b = duzina(t.xa, t.ya, t.xc, t.yc);
21     double c = duzina(t.xa, t.ya, t.xb, t.yb);
     double s = (a + b + c) / 2;
23     return sqrt(s * (s - a) * (s - b) * (s - c));
   }

25
27 /* Funkcija racuna poredi dva trougla, napisana tako da se moze
   proslediti funkciji qsort */
   int poredi(const void *a, const void *b)
29 {
   {
31     trougao x = *(trougao *) a;
     trougao y = *(trougao *) b;
     double xp = povrsina(x);
33     double yp = povrsina(y);
     if (xp < yp)

```

```

35     return 1;
36     if (xp > yp)
37         return -1;
38         return 0;
39 }

41 int main()
42 {
43     FILE *f;
44     int n, i;
45     trougao *niz;

46     if ((f = fopen("trouglovi.txt", "r")) == NULL) {
47         fprintf(stderr, "-1\n");
48         exit(EXIT_FAILURE);
49     }

50     if (fscanf(f, "%d", &n) != 1) {
51         fprintf(stderr, "-1\n");
52         exit(EXIT_FAILURE);
53     }

54     if ((niz = malloc(n * sizeof(trougao))) == NULL) {
55         fprintf(stderr, "-1\n");
56         exit(EXIT_FAILURE);
57     }

58     for (i = 0; i < n; i++) {
59         if (fscanf(f, "%lf%lf%lf%lf%lf%lf",
60                 &niz[i].xa, &niz[i].ya,
61                 &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
62         {
63             fprintf(stderr, "-1\n");
64             exit(EXIT_FAILURE);
65         }
66     }

67     qsort(niz, n, sizeof(trougao), &poredi);

68     for (i = 0; i < n; i++)
69         printf("%g %g %g %g %g %g\n",
70             niz[i].xa, niz[i].ya,
71             niz[i].xb, niz[i].yb, niz[i].xc, niz[i].yc);

72     free(niz);
73     fclose(f);

74     return 0;
75 }

```

## Rešenje 5.6

```
1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1

4  /* Struktura koja predstavlja cvor stabla, sadrzi vrednost koja se
   cuva i pokazivace na levo i desno podstablo. */
6  typedef struct cvor {
7      int vrednost;
8      struct cvor *levi;
9      struct cvor *desni;
10 } Cvor;

12 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira dinamicki,
   funkcijom malloc(). U slucaju greske program se prekida i ispisuje
14 se poruka o gresci. U slucaju uspeha inicijalizuje se vrednost
   datim brojem, a pokazivaci na podstabla se inicijalizuju na NULL.
16 Funkcija vraca adresu novokreiranog cvora */
Cvor *napravi_cvor(int broj);

18 /* Funkcija dodaje novi cvor u stablo sa datim korenom. Ukoliko broj
   vec postoji u stablu, ne radi nista. Cvor se kreira funkcijom
20 napravi_cvor(). */
void dodaj_u_stablo(Cvor ** koren, int broj);

22 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje elemente u
   rastucem poretku) */
24 void prikazi_stablo(Cvor * koren);

26 /* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
   vraca pokazican na njegov koren */
28 Cvor *ucitaj_stablo();

30 /* Funkcija oslobadja prostor koji je alociran za cvorove stabla. */
32 void oslobodi_stablo(Cvor ** koren);
34 #endif

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

5  Cvor *napravi_cvor(int broj)
6  {
7      /* Dinamicki kreiramo cvor */
8      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9
10     /* U slucaju greske ... */
11     if (novi == NULL) {
12         fprintf(stderr, "-1\n");
13         exit(1);
14     }
15 }
```

```
17  /* Inicijalizacija */
    novi->vrednost = broj;
    novi->levi = NULL;
19  novi->desni = NULL;

21  /* Vracamo adresu novog cvora */
    return novi;
23 }

25 void dodaj_u_stablo(Cvor ** koren, int broj)
{
27  /* Izlaz iz rekurzije: ako je stablo bilo prazno, novi koren je
    upravo novi cvor */
29  if (*koren == NULL) {
        *koren = napravi_cvor(broj);
31  return;
    }

33  /* Ako je stablo neprazno, i koren sadrzi manju vrednost od datog
    broja, broj se umece u desno podstablo, rekurzivnim pozivom */
35  if ((*koren)->vrednost < broj)
37  dodaj_u_stablo(&(*koren)->desni, broj);
    /* Ako je stablo neprazno, i koren sadrzi vecu vrednost od datog
    broja, broj se umece u levo podstablo, rekurzivnim pozivom */
39  else if ((*koren)->vrednost > broj)
41  dodaj_u_stablo(&(*koren)->levi, broj);
}

43 void prikazi_stablo(Cvor * koren)
{
45  /* Izlaz iz rekurzije */
47  if (koren == NULL)
        return;

49  prikazi_stablo(koren->levi);
    printf("%d ", koren->vrednost);
    prikazi_stablo(koren->desni);
53 }

55 Cvor *ucitaj_stablo()
{
57  Cvor *koren = NULL;
    int x;
59  while (scanf("%d", &x) == 1)
        dodaj_u_stablo(&koren, x);
61  return koren;
}

63 void oslobodi_stablo(Cvor ** koren)
{
65  /* Izlaz iz rekurzije */
67  if (*koren == NULL)
```

```

    return;
69
    oslobodi_stablo(&(*koren)->levi);
71    oslobodi_stablo(&(*koren)->desni);
    free(*koren);
73
    *koren = NULL;
75 }

```

```

1  #include <stdio.h>
   #include "stabla.h"
3
   int f3(Cvor * koren, int n)
5  {
   if (koren == NULL || n < 0)
7      return 0;
   if (n == 0) {
9       if (koren->levi == NULL && koren->desni != NULL)
           return 1;
11      if (koren->levi != NULL && koren->desni == NULL)
           return 1;
13      return 0;
   }
15     return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
   }
17
   int main()
19 {
   Cvor *koren;
21     int n;

23     scanf("%d", &n);
   koren = ucitaj_stablo();
25
   printf("%d\n", f3(koren, n));
27
   oslobodi_stablo(&koren);
29
   return 0;
31 }

```

### Rešenje 5.7

```

   #include <stdio.h>
2
   unsigned int Rotiraj(unsigned int x, unsigned int n)
4  {
   int i;
6     unsigned int maska = 1;

```

```

8  /* Formiranje maske sa n jedinica na kraju 000...00001111 */
   for (i = 1; i < n; i++)
10     maska = (maska << 1) | 1;

12   return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
   }

14
16   int main()
17   {
18     unsigned int x, n;

20     scanf("%u%u", &x, &n);

22     printf("%u\n", Rotiraj(x, n));

24     return 0;
   }

```

### Rešenje 5.8

```

2  #ifndef __LISTE_H__
   #define __LISTE_H__ 1

4  /* Struktura koja predstavlja cvor liste */
   typedef struct cvor {
6     double vrednost;
       struct cvor *sledeci;
8   } Cvor;

10  /* Pomocna funkcija koja kreira cvor. */
   Cvor *napravi_cvor(double broj);

12
14  /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
   void oslobodi_listu(Cvor ** adresa_glave);

16
18  /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
   ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
   zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
20  adresa_glave. */
   void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi);

22
24  /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
   Cvor *pronadji_poslednji(Cvor * glava);

26
28  /* Funkcija dodaje novi cvor na kraj liste. */
   void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);

30  /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
   */

```

```

32 void ispisi_listu(Cvor * glava);

/* Funkcija koja dopunjuje listu na nacin opisan u zadatku */
34 void dopuni_listu(Cvor ** adresa_glave);

36 #endif

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

4
/* Pomocna funkcija koja kreira cvor. */
6 Cvor *napravi_cvor(double broj)
{
8     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
10         return NULL;

12     /* inicijalizacija polja u novom cvoru */
    novi->vrednost = broj;
14     novi->sledeci = NULL;

16     return novi;
}

18
/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
20 void oslobodi_listu(Cvor ** adresa_glave)
22 {
    Cvor *pomocni = NULL;

24     while (*adresa_glave != NULL) {
26         pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
28         *adresa_glave = pomocni;
    }
30 }

32 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
   ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
   zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
   adresa_glave. */
34 void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
36 {
38     /* Ukoliko je novi NULL */
    if (novi == NULL) {
40         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_listu(adresa_glave);
42         exit(EXIT_FAILURE);
    }
44 }

```

```
46 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,  
    ili NULL kao je lista prazna */  
48 Cvor *pronadji_poslednji(Cvor * glava)  
{  
50     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju  
        funkcija vraca NULL. */  
52     if (glava == NULL)  
        return NULL;  
54  
    while (glava->sledeci != NULL)  
56         glava = glava->sledeci;  
58     return glava;  
}  
60  
/* Funkcija dodaje novi cvor na kraj liste. */  
62 void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)  
{  
64     Cvor *novi = napravi_cvor(broj);  
    prover_i_alokaciju(adresa_glave, novi);  
66  
    if (*adresa_glave == NULL) {  
68         *adresa_glave = novi;  
        return;  
70     }  
72  
    Cvor *poslednji = pronadji_poslednji(*adresa_glave);  
    poslednji->sledeci = novi;  
74 }  
76  
/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.  
    */  
void ispisi_listu(Cvor * glava)  
78 {  
    for (; glava != NULL; glava = glava->sledeci)  
80         printf("%.21f ", glava->vrednost);  
82  
    putchar('\n');  
}  
84  
/* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka */  
86 void dopuni_listu(Cvor ** adresa_glave)  
{  
88     Cvor *tekuci;  
    Cvor *novi;  
    double aritmeticka_sredina;  
90     if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)  
92         return;  
94  
    tekuci = *adresa_glave;  
    while (tekuci->sledeci != NULL) {  
96         aritmeticka_sredina =
```



```

    ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
98  novi = napravi_cvor(aritmeticka_sredina);
    prover_i_alokaciju(adresa_glave, novi);
100
    novi->sledeci = tekuci->sledeci;
102  tekuci->sledeci = novi;
    tekuci = tekuci->sledeci;
104  tekuci = tekuci->sledeci;
    }
106
    return;
108 }

```

```

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

4  int main()
{
6  Cvor *glava = NULL;
    double broj;
8
    /* Ucitavanje se vrši do kraja ulaza. Elementi se dodaju na kraj
10     liste! */
    while (scanf("%lf", &broj) > 0)
12     dodaj_na_kraj_liste(&glava, broj);

14     dopuni_listu(&glava);

16     ispisi_listu(glava);

18     oslobodi_listu(&glava);

20     return 0;
}

```

### Rešenje 5.9

```

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

4  /* Funkcija proverava da li je magican kvadrat koji joj se
    prosledjuje kao argument. Ukoliko jeste magican funkcija vraca 1,
6     inace 0. */
    int magicni_kvadrat(int **M, int n)
8  {
    int i, j;
10  int zbir = 0, zbir_pom;

12  for (j = 0; j < n; j++)
        zbir += M[0][j];

```

```
14     for (i = 1; i < n; i++) {
16         zbir_pom = 0;
17         for (j = 0; j < n; j++)
18             zbir_pom += M[i][j];
19         if (zbir_pom != zbir)
20             return 0;
21     }
22
23     for (j = 0; j < n; j++) {
24         zbir_pom = 0;
25         for (i = 0; i < n; i++)
26             zbir_pom += M[i][j];
27         if (zbir_pom != zbir)
28             return 0;
29     }
30     return 1;
31 }
32
33 int main()
34 {
35     int n, i, j;
36     int **matrica = NULL;
37     int zbir = 0;
38
39     scanf("%d", &n);
40
41     if (n <= 0) {
42         printf("-1\n");
43         exit(EXIT_FAILURE);
44     }
45
46     matrica = (int **) malloc(n * sizeof(int *));
47     if (matrica == NULL) {
48         printf("-1\n");
49         exit(EXIT_FAILURE);
50     }
51
52     for (i = 0; i < n; i++) {
53         matrica[i] = (int *) malloc(n * sizeof(int));
54
55         if (matrica[i] == NULL) {
56             fprintf(stderr, "-1\n");
57             for (j = 0; j < i; j++)
58                 free(matrica[j]);
59
60             free(matrica);
61             exit(EXIT_FAILURE);
62         }
63     }
64
65     for (i = 0; i < n; i++)
```

```
66     for (j = 0; j < n; j++)
67         scanf("%d", &matrica[i][j]);
68
69     if (magicni_kvadrat(matrica, n)) {
70         for (i = 0; i < n; i++)
71             zbir += matrica[0][i];
72         printf("%d\n", zbir);
73     } else
74         printf("-\n");
75
76     for (j = 0; j < n; j++)
77         free(matrica[j]);
78
79     free(matrica);
80
81     return 0;
82 }
```