

Univerzitet u Beogradu  
Matematički fakultet

Milena, Jelena, Ana, Mirko, Anđelka, Nina

## Zbirka programa

Beograd, 2015.



# Predgovor

U okviru kursa *Programiranje 2* na Matematičkom fakultetu vežbaju se zadaci koji imaju za cilj da studente nauče rekurzivnom pristupu rešavanju problema, ispravan rad sa pokazivačima i dinamički alociranom memorijom, osnovne algoritme pretraživanja i sortiranja, kao i rad 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.

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

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

*Autori*

---

# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>3</b>
1.1	Podela koda po datotekama . . . . .	3
1.2	Algoritmi za rad sa bitovima . . . . .	3
1.3	Rekurzija . . . . .	9
1.4	Rešenja . . . . .	12
<b>2</b>	<b>Pokazivači</b>	<b>15</b>
2.1	Pokazivačka aritmetika . . . . .	15
2.2	Višedimenzioni nizovi . . . . .	18
2.3	Dinamička alokacija memorije . . . . .	22
2.4	Pokazivači na funkcije . . . . .	25
2.5	Rešenja . . . . .	27
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>63</b>
3.1	Pretraživanje . . . . .	63
3.2	Sortiranje . . . . .	67
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	78
3.4	Rešenja . . . . .	82
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>147</b>
4.1	Liste . . . . .	147
4.2	Stabla . . . . .	152
4.3	Rešenja . . . . .	160
<b>5</b>	<b>Ispitni rokovi</b>	<b>165</b>
5.1	Programiranje 2, praktični deo ispita, jun 2015. . . . .	165
5.2	Programiranje 2, praktični deo ispita, jul 2015. . . . .	166
5.3	Rešenja . . . . .	168
	<b>Literatura</b>	<b>174</b>



# Glava 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

### 1.2 Algoritmi za rad sa bitovima

#### Zadatak 1.1

- (a) Napisati funkciju `print_bits` koja štampa bitove u binarnom zapisu celog broja  $x$ .
- (b) Napisati program koja testira funkciju `print_bits` za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

#### *Test 1*

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

#### *Test 2*

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

#### *Test 3*

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

#### *Test 4*

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

#### *Test*

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

## 1 Uvodni zadaci

---

**Zadatak 1.2** Napisati funkciju koja broji bitove postavljene na 1 u zapisu broja  $x$ . Napisati program koji testira tu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Ulaz: 0x7F</code> <code>Izlaz: 7</code>	<code>Ulaz: 0x80</code> <code>Izlaz: 1</code>	<code>Ulaz: 0x00FF00FF</code> <code>Izlaz: 16</code>
<i>Test 4</i>	<i>Test 4</i>	
<code>Ulaz: 0xFFFFFFFF</code> <code>Izlaz: 32</code>	<code>Ulaz: 0xABCDE123</code> <code>Izlaz: 17</code>	

### Zadatak 1.3

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

*Test 1*

```
Ulaz: 0x7F
Izlaz:
Najveci:
1111 1110 0000 0000 0000 0000 0000 0000
Najmanji:
0000 0000 0000 0000 0000 0000 0111 1110
```

*Test 2*

```
Ulaz: 0x80
Izlaz:
Najveci:
1000 0000 0000 0000 0000 0000 0000 0000
Najmanji:
0000 0000 0000 0000 0000 0000 0000 0001
```

*Test 3*

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



Test 4

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

Test 4

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

**Zadatak 1.4** Napisati program za rad sa bitovima.

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

Program treba da testira prethodno napisane funkcije nad neoznačenim celim brojem koji se unosi sa standardnog ulaza. *Napomena: Pozicije se broje počev od pozicije najnižeg bita, pri čemu se broji od nule.*

**Zadatak 1.5**

- (a) Napisati funkciju koja određuje broj koji se dobija rotiranjem u levo datog celog broja. *Napomena: Rotiranje podrazumeva pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najviše težine pomera na mesto najmanje težine.*
- (b) Napisati funkciju koja određuje broj koji se dobija rotiranjem u desno datog celog neoznačenog broja.

## 1 Uvodni zadaci

---

- (c) Napisati funkciju koja određuje broj koji se dobija rotiranjem u desno datog celog broja.
- (d) Napisati program koji testira prethodno napisane funkcije za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu.

**Zadatak 1.6** Napisati funkciju `mirror` koja određuje ceo broj čiji binarni zapis je slika u ogledalu binarnog zapisa argumenta funkcije. Napisati i program koji testira datu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekaskom formatu, tako što najpre ispisuje binarnu reprezentaciju unetog broja, a potom i binarnu reprezentaciju broja dobijenog nakon poziva funkcije `mirror` za uneti broj.

**Zadatak 1.7** 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.

	<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<b>Ulaz:</b>	10	1024	2147377146
<b>Izlaz:</b>	0	0	1

  

	<i>Test 4</i>
<b>Ulaz:</b>	1111111115
<b>Izlaz:</b>	0

**Zadatak 1.8** Napisati funkciju koja broji koliko se puta kombinacija 11 (dve uzastopne jedinice) pojavljuje u binarnom zapisu celog neoznačenog broja  $x$ . Tri uzastopne jedinice se broje dva puta. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

	<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<b>Ulaz:</b>	11	1024	2147377146
<b>Izlaz:</b>	1	0	22

  

	<i>Test 4</i>
<b>Ulaz:</b>	1111111115
<b>Izlaz:</b>	6

**Zadatak 1.9** Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama  $i$ ,  $j$ . Pozicije  $i$ ,  $j$  se učitavaju kao parametri komandne linije. Smatrati da je krajnji desni bit binarne reprezentacije 0-ti bit. Pri rešavanju nije dozvoljeno koristiti pomoćni niz niti aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ .

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

**Zadatak 1.10** Napisati funkciju koja na osnovu neoznačenog broja  $x$  formira nisku  $s$  koja sadrži heksadekadni zapis broja  $x$ , koristeći algoritam za brzo prevođenje binarnog u heksadekadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksadekadnom cifrom). Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

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

**Zadatak 1.11** Napisati funkciju koja za dva data neoznačena broja  $x$  i  $y$  invertuje u podatku  $x$  one bitove koji se poklapaju sa odgovarajućim bitovima u broju  $y$ . Ostali bitovi ostaju nepromenjeni. Napisati program koji tu funkciju testira za brojeve koji se zadaju sa standardnog ulaza.

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

**Zadatak 1.12** Napisati funkciju koja računa koliko petica bi imao ceo neoznačen broj  $x$  u oktalnom zapisu. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

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

Milena: Naredne zadatke prebaciti da budu nakon rekurzije.

**Zadatak 1.13** Napisati rekurzivnu funkciju vraća broj bitova koji su postavljeni na 1, u binarnoj reprezentaciji njenog celobrojnog argumenta. Napisati program koji testira napisanu funkciju za brojeve koji se učitavaju sa standardnog ulaza zadati u heksadekadnom formatu.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
Ulaz: 0x7F    Izlaz: 7	Ulaz: 0x80    Izlaz: 1	Ulaz: 0x00FF00FF    Izlaz: 16
		<i>Test 4</i>
		Ulaz: 0xFFFFFFFF    Izlaz: 32

### Zadatak 1.14

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

*Test 1*

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

**Zadatak 1.15** Nina: Ovo je prebaceno iz poglavlja sa pokazivacima. Napisati rekurzivnu funkciju za određivanje najveće cifre u oktalnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. Uputstvo: binarne cifre grupisati u podgrupe od po tri cifre, počev od bitova najmanje težine.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
Ulaz: 5    Izlaz: 5	Ulaz: 125    Izlaz: 7	Ulaz: 8    Izlaz: 1
<i>Test 4</i>		
Ulaz: 10    Izlaz: 2		

**Zadatak 1.16** Nina: Ovo je prebaceno iz poglavlja sa pokazivacima. Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadmnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. Uputstvo: binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
Ulaz: 5	Ulaz: 16	Ulaz: 18
Izlaz: 5	Izlaz: 1	Izlaz: 2
<i>Test 4</i>		
Ulaz: 165		
Izlaz: 10		

## 1.3 Rekurzija

**Zadatak 1.17** Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva. Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojong niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom.

**Zadatak 1.18** Napisati rekurzivnu funkciju koja izračunava  $x^k$ , za dati realni broj  $x$  i prirodan broj  $k$ . Napisati program koji testira napisanu funkciju za vrednosti koje se unose sa standardnog ulaza.

**Zadatak 1.19** Koristeći uzajamnu (posrednu) rekurziju napisati naredne dve funkcije:

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

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

**Zadatak 1.20** Napisati repno-rekurzivnu funkciju koja izračunava faktorijel broja  $n$ . Napisati program koji testira napisanu funkciju za poizvoljan broj  $n$  ( $n \leq 12$ ) unet sa standardnog ulaza.

**Zadatak 1.21** 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  $F$ .
- Napisati rekurzivnu funkciju koja računa  $n$ -ti element u nizu  $F$  ali tako da se problemi manje dimenzije rešavaju samo jedan put.

Napisati program koji testira napisane funkcije za poizvoljan broj  $n$  ( $n \in \mathbb{N}$ ) unet sa standardnog ulaza.

**Zadatak 1.22** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za poizvoljan prirodan broj  $n$  ( $n \leq 50$ ) unet sa standardnog ulaza.

**Zadatak 1.23** Paskalov trougao se dobija tako što mu je svako polje (izuzev jedinica po krajevima) zbir jednog polja levo i jednog polja iznad.

## 1 Uvodni zadaci

---

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

- (a) Napisati rekurzivnu funkciju koja izračunava  $d_n$  kao sumu elemenata  $n$ -te hipotenuze Paskalovog trougla.
- (b) Napisati rekurzivnu funkciju koja izračunava vrednost polja  $(i, j)$ . **Milena:** dodati sta je  $i$  a sta  $j$  tj odakle se broji

**Zadatak 1.24** Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju, za niz koji se unosi sa standardnog ulaza. Niz neće imati više od 256 elemenata, i njegovi elementi se unose sve do kraja ulaza.

	<i>Test 1</i>	<i>Test 2</i>
<b>Ulaz:</b>	3 2 1 4 21	2 -1 0 -5 -10
<b>Izlaz:</b>	21	2

  

	<i>Test 3</i>	<i>Test 4</i>
<b>Ulaz:</b>	1 11 3 5 8 1	5
<b>Izlaz:</b>	11	5

**Zadatak 1.25** Napisati rekurzivnu funkciju koja izračunava skalarni proizvod dva data vektora. Napisati program koji testira ovu funkciju, za nizove koji se unose sa standardnog ulaza. Nizovi neće imati više od 256 elemenata. Prvo se unosi dimenzija nizova, a zatim i sami njihovi elementi.

	<i>Test 1</i>	<i>Test 2</i>
<b>Ulaz:</b>	3 1 2 3 1 2 3	2 3 5 2 6
<b>Izlaz:</b>	14	36

  

	<i>Test 3</i>
<b>Ulaz:</b>	0
<b>Izlaz:</b>	0

**Zadatak 1.26** Napisati rekurzivnu funkciju koja sabira dekadne cifre datog celog broja  $x$ . Napisati program koji testira ovu funkciju, za broj koji se unosi sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<b>Ulaz:</b> 123    <b>Izlaz:</b> 6	<b>Ulaz:</b> 23156    <b>Izlaz:</b> 17	<b>Ulaz:</b> 1432    <b>Izlaz:</b> 10
<i>Test 4</i>	<i>Test 5</i>	
<b>Ulaz:</b> 1    <b>Izlaz:</b> 1	<b>Ulaz:</b> 0    <b>Izlaz:</b> 0	

**Zadatak 1.27** Napisati rekurzivnu funkciju koja računa broj pojavljivanja elementa  $x$  u nizu  $a$  dužine  $n$ . Napisati program koji testira ovu funkciju, za  $x$  i niz koji se unose sa standardnog ulaza. Niz neće imati više od 256 elemenata. Prvo se unosi  $x$ , a zatim elementi niza sve do kraja ulaza.

<i>Test 1</i>	<i>Test 2</i>
<b>Ulaz:</b> 4 1 2 3 4    <b>Izlaz:</b> 1	<b>Ulaz:</b> 11 3 2 11 14 11 43 1    <b>Izlaz:</b> 2
<i>Test 3</i>	
<b>Ulaz:</b> 1 3 21 5 6    <b>Izlaz:</b> 0	

**Zadatak 1.28** Napisati rekurzivnu funkciju koja ispituje da li je data niska palindrom. Napisati program koji testira ovu funkciju. Pretpostaviti da niska neće imati više od 31 karaktera, i da se unosi sa standardnog ulaza.

<i>Test 1</i>		<i>Test 2</i>			
Ulaz:	programiranje	Ulaz:	anavolimilovana		
Izlaz:	ne	Izlaz:	da		
<i>Test 3</i>		<i>Test 4</i>		<i>Test</i>	
Ulaz:	a	Ulaz:	aba	Ulaz:	aa
Izlaz:	da	Izlaz:	da	Izlaz:	da

**Zadatak 1.29** Napisati rekurzivnu funkciju kojom se proverava da li su tri zadata broja uzastopni članovi niza. Potom, napisati program koji je testira. Sa standardnog ulaza se unose najpre tri tražena broja, a zatim elementi niza, sve do kraja ulaza. Pretpostaviti da neće biti uneto više od 256 brojeva.

<i>Test 1</i>	<i>Test 2</i>
<b>Ulaz:</b> 1 2 3 4 1 2 3 4 5    <b>Izlaz:</b> da	<b>Ulaz:</b> 1 2 3 11 1 2 4 3 6    <b>Izlaz:</b> ne

### Test 3

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

**Zadatak 1.30** 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 1.31** *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 1.32** *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](#)

Rešenje [1.2](#)

Rešenje [1.3](#)



Rešenje [1.4](#)

Rešenje [1.5](#)

Rešenje [1.6](#)

Rešenje [1.7](#)

Rešenje [1.8](#)

Rešenje [1.9](#)

Rešenje [1.10](#)

Rešenje [1.11](#)

Rešenje [1.12](#)

Rešenje [1.13](#)

```
#include<stdio.h>
2
int main(){
4     printf("Hello bitovi!\n"); /* Da li komentari rade ččžš*/
    return 0;
6 }
```

Rešenje [1.15](#)

Rešenje [1.16](#)

Rešenje [1.17](#)

```
#include<stdio.h>
2
int main(){
4     printf("Hello bitovi!\n"); /* Da li komentari rade ččžš*/
    return 0;
6 }
```

Rešenje [1.18](#)

```
#include<stdio.h>
2
int main(){
4     printf("Hello bitovi!\n"); /* Da li komentari rade ččžš*/
    return 0;
6 }
```

Rešenje [1.19](#)

Rešenje [1.20](#)

Rešenje [1.21](#)

Rešenje [1.22](#)

Rešenje [1.23](#)

Rešenje [1.24](#)

Rešenje [1.25](#)

Rešenje [1.26](#)

Rešenje [1.27](#)

Rešenje [1.28](#)

Rešenje [1.29](#)

# Glava 2

## Pokazivači

### 2.1 Pokazivačka aritmetika

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

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

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

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

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

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

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 učitanoog niza.

### Test 1

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

**Zadatak 2.3** Korišćenjem pokazivačke sintakse, napisati funkciju koja vrednosti elemenata u prvoj polovini niza povećava za jedan, a u drugoj polovini smanjuje za jedan. Ukoliko niz ima neparan broj elemenata, onda vrednost srednjeg elementa niza ostaviti nepromenjenim. Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojong niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom. *Jelena: Sta kazete na to da prekoracenja dimenzije niza u razlicitim zadacima razlicito obradjujemo. Na primer, mozemo da unosimo dimenziju niza sve dok se ne unese broj koji je u odgovarajucem opsegu, ili mozemo da dimenziju postavimo na 1 ako je korisnik uneo broj manji od 1, a na MAX ako je korisnik uneo broj veci od MAX, itd?*

### Test 1

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

### Test 2

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

### Test 3

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

### Test 4

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

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

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

*Jelena: Da li je ok da ovaj zadatak pod a i b resim na nacin na koji sam resila, odnosno, da jedno od ta dva resenja iskomentarisem? Milena: Meni se cini da je bolje bez komentarisanja, vec da su oba prisutna.*

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

**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.

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

**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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> Poziv: ./a.out ulaz.txt 1 ulaz.txt: Ovo je sadržaj datoteke i u njoj ima reci koje imaju           1 karakter Izlaz: 3 </pre>	<pre> Poziv: ./a.out ulaz.txt Izlaz: Greska: Nedovoljan broj argumenata komandne linije.         Program se poziva sa ./a.out ime_dat br_karaktera. </pre>	<pre> Poziv: ./a.out ulaz.txt 2 (ne postoji datoteka ulaz.txt) Izlaz: Greska: Neuspesno otvaranje datoteke ulaz.txt. </pre>

**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.

### Test 1

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

### Test 2

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

### Test 3

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

### Test 3

```
Poziv: ./a.out ulaz.txt
Izlaz: Greska: Nedovoljan broj argumenata komandne linije.
       Program se poziva sa ./a.out ime_dat suf/pref -s/-p.
```

## 2.2 Višedimenzioni nizovi

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

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

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

## Test 1

```

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

```

## Test 2

```

Ulaz:  0
Izlaz: Greska: neodgovarajuca dimenzija matrice.

```

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

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

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

## Test 1

```

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

```

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

- (a) Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- (b) Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- (c) Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.

- (d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).
- (e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).
- (f) Napisati funkciju koja određuje refleksivno-tranzitivno zatvorenje relacije (najmanju refleksivnu i tranzitivnu relaciju koja sadrži datu)(Napomena: koristiti Varšalov algoritam).

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

### Test 1

```
Poziv: ./a.out ulaz.txt
ulaz.txt: 4
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 0
Izlaz:    Refleksivnost: ne
          Simetricnost: ne
          Tranzitivnost: da
          Refleksivno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 1
          Simetricno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 1 1 0
          0 0 0 0
          Refleksivno-tranzitivno zatvorenje:
          1 0 0 0
          0 1 1 0
          0 0 1 0
          0 0 0 0
```

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

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

<i>Test 1</i>	<i>Test 2</i>
<pre> Poziv: ./a.out 3 Ulaz:  1 2 3         -4 -5 -6         7 8 9 Izlaz:  7 2 2 3           </pre>	<pre> Poziv: ./a.out 4 Ulaz:  -1 -2 -3 -4         -5 -6 -7 -8         -9 -10 -11 -12         -13 -14 -15 -16 Izlaz:  -4 3 0 16           </pre>
<i>Test 3</i>	
<pre> Poziv: ./a.out Izlaz: Greska: Nedovoljan broj argumenata komandne linije.         Program se poziva sa ./a.out dim_matrice.           </pre>	

**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.

<i>Test 1</i>	<i>Test 2</i>
<pre> Ulaz:  4         1 0 0 0         0 1 0 0         0 0 1 0         0 0 0 1 Izlaz: da           </pre>	<pre> Ulaz:  3         1 2 3         5 6 7         1 4 2 Izlaz: ne           </pre>
<i>Test 3</i>	
<pre> Ulaz:  33 Izlaz: Greska: neodgovarajuca dimenzija matrice.           </pre>	

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

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

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

## 2 Pokazivači

---

(pozivom gore napisane funkcije). Na standardni izlaz spiralno ispisati elemente učitane matrice.

<i>Test 1</i>	<i>Test 2</i>
<pre>   Ulaz:  3 3          1 2 3          4 5 6          7 8 9    Izlaz: 1 2 3 6 9 8 7 4 5</pre>	<pre>   Ulaz:  3 4          1 2 3 4          5 6 7 8          9 10 11 12    Izlaz: 1 2 3 4 8 12 11 10 9 5 6 7</pre>
<i>Test 3</i>	
<pre>   Ulaz:  11 4    Izlaz: Greska: neodgovarajuće dimenzije matrice.</pre>	

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

*Test 1*

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

## 2.3 Dinamička alokacija memorije

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

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

**Zadatak 2.16** Napisati program koji sa standardnog ulaza učitava niz celih brojeva. Brojevi se unose sve dok se ne unese nula. Ne praviti nikakve pretpostavke

o dimenziji niza. Na standardni izlaz ispisati ovaj niz brojeva u obrnutom poretku. Zadatak uraditi na dva načina:

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

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

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

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

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

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

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati  $n$  i  $m$  (ne praviti nikakve pretpostavke o njihovoj veličini), zatim učitati elemente matrice i na standardni izlaz ispisati elemente ispod glavne dijagonale matrice.

### Test 1

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

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

### Test 1

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

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

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

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

### Test 1

```
Poziv: ./a.out matrica.txt
matrica.txt:  3
              1.1 -2.2 3.3
              -4.4 5.5 -6.6
              7.7 -8.8 9.9
Izlaz: Zbir apsolutnih vrednosti ispod sporedne dijagonale je 25.30.
       Transformisana matrica je:
       1.10 -1.10 1.65
       -8.80 5.50 -3.30
       15.40 -17.60 9.90
```

**Zadatak 2.22** Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci „slicice.txt“ se nalaze informacije o sličicama koje mu nedostaju

u formatu: `redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`. Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronade ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. Napomena: za realokaciju memorije koristiti `realloc()` funkciju. **Jelena: treba dodati test primer.**

**Zadatak 2.23** U datoteci „temena.txt“ se nalaze tačke koje predstavljaju temena nekog  $n$ -tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom  $n$ -touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan. **Jelena: treba dodati test primer.**

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

**Zadatak 2.25** Na ulazu se zadaje niz celih brojeva čiji se unos završava nulom. Napisati funkciju koja od zadatog niza formira matricu tako da prva vrsta odgovara unetom nizu, a svaka naredna se dobija cikličkim pomeranjem elemenata niza za jednu poziciju ulevo. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz. **Jelena: treba dodati test primer.**

## 2.4 Pokazivači na funkcije

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

### Test 1

```

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

```

### Test 2

```

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

```

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

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

### Test 1

```

Ulaz:  tan 1.570795 10000
Izlaz: -10134.5

```

### Test 2

```

Ulaz:  log 0 1000000
Izlaz: -13.81551

```

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

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

Vrednost  $h$  se izračunava po formuli  $h = (b - a)/n$ , dok se vrednosti  $n$ ,  $a$  i  $b$  unose

sa standardnog ulaza kao i ime funkcije iz zaglavlja `math.h`. Na standardni izlaz ispisati vrednost integrala. **Jelena: treba dodati test primer.**

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

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

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

## 2.5 Rešenja

### Rešenje 2.1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija obrne 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 obrne elemente niza koriscenjem pokazivacke
19    sintakse. Umesto "void obrni_niz(int *a, int n)" potpis
20    metode bi mogao da bude i "void obrni_niz(int a[], int n)".
21    U oba slucaja se argument funkcije "a" tumaci kao pokazivac,
22    ili tacnije, kao adresa prvog elementa niza. U odnosu na
23    njega se odredjuju adrese ostalih elemenata u nizu */
24 void obrni_niz_v2(int *a, int n)
25 {
26     /* Pokazivaci na elemente niza a */
27     int *prvi, *poslednji;
28
29     for(prvi = a, poslednji = a + n - 1;
30         prvi < poslednji; prvi++, poslednji--) {
31         int t = *prvi;

```

```

    *prvi = *poslednji;
34    *poslednji = t;
    }
36 }

38 /* Funkcija obrće elemente niza koriscenjem pokazivacke
    sintakse - modifikovano koriscenje pokazivaca */
40 void obrni_niz_v3(int *a, int n)
{
42     /* Pokazivaci na elemente niza a */
    int *prvi, *poslednji;
44
    /* Obrćemo niz */
46     for(prvi = a, poslednji = a + n - 1; prvi < poslednji; ) {
        int t = *prvi;
48
        /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
50         vrednost koja se nalazi na adresi na koju pokazuje
        pokazivac "poslednji". Nakon toga se pokazivac "prvi"
52         uvecava za jedan sto za posledicu ima da "prvi" pokazuje
        na sledeci element u nizu */
54         *prvi++ = *poslednji;

56         /* Vrednost promenljive "t" se postavlja na adresu na koju
        pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
58         umanjuje za jedan, sto za posledicu ima da pokazivac
        "poslednji" sada pokazuje na element koji mu prethodi u
60         nizu */
        *poslednji-- = t;
62     }
    }
64

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

70     /* Broj elemenata niza a */
    int n;
72
74     /* Pokazivac na elemente niza a */
    int *p;

76     /* Unosimo dimenziju niza */
    scanf("%d", &n);
78
80     /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
    if(n <= 0 || n > MAX) {
        fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
82         exit(EXIT_FAILURE);
    }
84

    /* Unosimo elemente niza */
86     for(p = a; p - a < n; p++)
        scanf("%d", p);
88
```



```

    obrni_niz_v1(a,n);
90 // obrni_niz_v2(a,n);
    // obrni_niz_v3(a,n);

92
    /* Prikazujemo sadržaj niza nakon obrtanja */
94 for(p = a; p - a < n; p++)
    printf("%d ", *p);
96 printf("\n");

98 return 0;
}

```

## Rešenje 2.2

```

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

4 #define MAX 100

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

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

14    return s;
}

16
/* Funkcija racuna proizvod elemenata niza */
18 double proizvod(double a[], int n)
{
20    double p = 1;

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

24    return p;
26 }

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

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

40
    return min;
}

```

```
42 }
44 /* Funkcija racuna najveći element niza */
double max(double *a, int n)
46 {
    /* Za najveći element se najpre postavlja prvi element */
48     double max = *a;

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

56     return max;
}
58

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

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

68     /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
    if(n <= 0 || n > MAX) {
70         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
        exit(EXIT_FAILURE);
72     }

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

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

84     return 0;
}
```

### Rešenje 2.3

```
#include <stdio.h>
2 #include <stdlib.h>
#define MAX 100
4
/* Funkcija povecava za jedan sve elemente u prvoj polovini niza
6  a smanjuje za jedan sve elemente u drugoj polovini niza.
    Ukoliko niz ima neparan broj elemenata, srednji element
8  ostaje nepromenjen */
```

```
void povecaj_smanji (int *a , int n)
10 {
    int *prvi = a;
12     int *poslednji = a+n-1;

14     while( prvi < poslednji ){

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

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

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

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

32 void povecaj_smanji_sazetije(int *a , int n)
{
34     int *prvi = a;
    int *poslednji = a+n-1;

36     while( prvi < poslednji ){
38         (*prvi++)++;
        (*poslednji--)--;
40     }
}

42 int main()
44 {
    int a[MAX];
46     int n;
    int *p;

48     /* Unosimo broj elemenata */
50     scanf("%d", &n);

52     /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
    if(n <= 0 || n > MAX) {
54         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
        exit(EXIT_FAILURE);
56     }

58     /* Unosimo elemente niza */
    for(p = a; p - a < n; p++)
60         scanf("%d", p);

62     povecaj_smanji(a,n);
    /* povecaj_smanji_sazetije(a,n); */
64 }
```

```
/* Prikaz niza nakon modifikacije */
66 for(p = a; p - a < n; p++)
    printf("%d ", *p);
68 printf("\n");

70 return 0;
}
```

### Rešenje 2.4

```
#include <stdio.h>

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

10    /* Ispisujemo broj argumenata komandne linije */
    printf("%d\n", argc);

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

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

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

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

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

40
    /* koristeći pokazivacku sintaksu */

42
    for (i=0 ; i<argc; i++ )
44        printf("%c ", **argv++);
```

```
46     return 0;
    }
```

### Rešenje 2.5

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

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

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

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

    printf("%d\n", n);
    return 0;
}
```

### Rešenje 2.6

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   #define MAX_KARAKTERA 100
5
   /* Funkcija strlen() iz standardne biblioteke */
7  int duzina(char *s)
   {
9      int i;
      for(i = 0; *(s+i); i++)
11         ;
      return i;
13 }

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

```
21  /* Ako korisnik nije uneo trazene argumente,
    prijavljujemo gresku */
23  if(argc < 3) {
    printf("Greska: ");
25  printf("Nedovoljan broj argumenata komandne linije.\n");
    printf("Program se poziva sa %s ime_dat br_karaktera.\n",
27                                     argv[0]);
    exit(EXIT_FAILURE);
29  }

31  /* Otvaramo datoteku sa imenom koje se zadaje kao prvi
    argument komandne linije. */
33  in = fopen(*(argv+1), "r");
    if(in == NULL){
35      fprintf(stderr, "Greska: ");
      fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
37                                     argv[1]);
      exit(EXIT_FAILURE);
39  }

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

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

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

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

### Rešenje 2.7

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

3

5  #define MAX_KARAKTERA 100

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

13

    /* Funkcija strcmp() iz standardne biblioteke */
15  int poredjenje_niski(char *s, char *t)
    {
17      int i;
```

```

19     for (i = 0; *(s+i) == *(t+i); i++)
        if(*(s+i) == '\0')
            return 0;
21     return *(s+i) - *(t+i);
}

23
/* Funkcija strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
{
27     int i;
    for(i = 0; *(s+i); i++)
29         ;
    return i;
31 }

33 /* Funkcija ispituje da li je niska zadata drugim argumentom
    funkcije sufiks niske zadate prvi argumentom funkcije */
35 int sufiks_niske(char *niska, char *sufiks) {
    if(duzina_niske(sufiks) <= duzina_niske(niska) &&
37         poredjenje_niski(niska + duzina_niske(niska) -
            duzina_niske(sufiks), sufiks) == 0)
39         return 1;
    return 0;
41 }

43 /* Funkcija ispituje da li je niska zadata drugim argumentom
    funkcije prefiks niske zadate prvi argumentom funkcije */
45 int prefiks_niske(char *niska, char *prefiks) {
    int i;
47     if(duzina_niske(prefiks) <= duzina_niske(niska)) {
        for(i=0; i<duzina_niske(prefiks); i++)
49             if(*(prefiks+i) != *(niska+i))
                return 0;
51         return 1;
    }
53     else return 0;
}

55
int main(int argc, char **argv)
57 {
    /* Ako korisnik nije uneo trazene argumente,
59     prijavljujemo gresku */
    if(argc < 4) {
61         printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
63         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
            argv[0]);
65         exit(EXIT_FAILURE);
    }

67
    FILE *in;
69     int br = 0, i = 0, n;
    char rec[MAX_KARAKTERA];

71
    in = fopen(*(argv+1), "r");
73     if(in == NULL) {

```

```
75     fprintf(stderr, "Greska: ");
76     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
77               argv[1]);
78     exit(EXIT_FAILURE);
79 }
80
81 /* Proveravamo kojom opcijom je pozvan program a zatim
82    učitavamo reci iz datoteke brojimo koliko reci
83    zadovoljava trazeni uslov */
84 if(!(poredjenje_niski(*(argv + 3), "-s")))
85     while(fscanf(in, "%s", rec) != EOF)
86         br += sufiks_niske(rec, *(argv+2));
87 else if (!(poredjenje_niski(*(argv+3), "-p")))
88     while(fscanf(in, "%s", rec) != EOF)
89         br += prefiks_niske(rec, *(argv+2));
90
91 printf("%d\n", br);
92 fclose(in);
93 return 0;
94 }
```

### Rešenje 2.8

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 #define MAX 100
6
7 /* Deklarisemo funkcije koje cemo kasnije da definisemo */
8 double euklidska_norma( int M[][MAX], int n);
9 int trag(int M[][MAX], int n);
10 int gornja_vandijagonalna_norma(int M[][MAX], int n);
11
12 int main()
13 {
14     int A[MAX][MAX];
15     int i,j,n;
16
17     /* Unosimo dimenziju kvadratne matrice */
18     scanf("%d",&n);
19
20     /* Proveravamo da li je prekoraceno ogranicenje */
21     if( n > MAX || n <= 0) {
22         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
23         fprintf(stderr, "matrice.\n");
24         exit(EXIT_FAILURE);
25     }
26
27     /* Popunjavamo vrstu po vrstu matrice */
28     for(i = 0; i<n; i++)
29         for (j=0 ; j<n; j++)
30             scanf("%d",&A[i][j]);
31
32     /* Ispis elemenata matrice koriscenjem indeksne sintakse.
```



```

    Ispis vrsimo vrstu po vrstu */
34 for(i = 0; i<n; i++) {
    /* Ispisujemo elemente i-te vrste */
36     for ( j=0 ; j<n; j++)
        printf("%d ", A[i][j]);
38     printf("\n");
    }
40
    /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
    Kod ovako definisane matrice, elementi su uzastopno
    smesteni u memoriju, kao na traci. To znaci da su svi
    elementi prve vrste redom smesteni jedan iza drugog. Odmah
    44 iza poslednjeg elementa prve vrste smesten je prvi element
    druge vrste za kojim slede svi elementi te vrste
    i tako dalje redom */
46
    /*
    48 for( i = 0; i<n; i++) {
        for ( j=0 ; j<n; j++)
            printf("%d ", *(A+i+j));
    52     printf("\n");
    }
    54 */

    int tr = trag(A,n);
    printf("trag = %d\n",tr);
    58
    printf("euklidska norma = %.2f\n",euklidska_norma(A,n));
    60 printf ("vandijagonalna norma = %d\n",
        gornja_vandijagonalna_norma(A,n));
    62
    return 0;
    64 }

    66 /* Definisemo funkcije koju smo ranije deklarirali */

    68 /* Funkcija izracunava trag matrice */
    int trag(int M[][MAX], int n)
    70 {
        int trag = 0,i;
    72     for(i=0; i<n; i++)
        trag += M[i][i];
    74     return trag;
    }
    76

    /* Funkcija izracunava euklidsku normu matrice */
    78 double euklidska_norma(int M[][MAX], int n)
    {
    80     double norma = 0.0;
        int i,j;
    82
        for(i= 0; i<n; i++)
    84             for(j = 0; j<n; j++)
                norma += M[i][j] * M[i][j];
    86
        return sqrt(norma);
    88 }

```

```
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

```
#include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
   standardnog ulaza */
7 void ucitaj_matricu(int m[][MAX], int n)
8 {
9     int i, j;
10
11     for(i=0; i<n; i++)
12         for(j=0; j<n; j++)
13             scanf("%d", &m[i][j]);
14 }
15
16 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
   standardni izlaz */
17 void ispisi_matricu(int m[][MAX], int n) {
18     int i, j;
19
20     for(i=0; i<n; i++) {
21         for(j=0; j<n; j++)
22             printf("%d ", m[i][j]);
23         printf("\n");
24     }
25 }
26
27
28 /* Funkcija proverava da li su zadate kvadratne matrice a i b
   dimenzije n jednake */
29 int jednake_matrice(int a[][MAX], int b[][MAX], int n) {
30     int i, j;
31
32     for(i=0; i<n; i++)
33         for(j=0; j<n; j++)
34             /* Nasli smo elemente na istim pozicijama u matricama
               koji se razlikuju */
35             if(a[i][j]!=b[i][j])
36                 return 0;
37     return 1;
38 }
```

```
        return 0;
40
    /* Prosla je provera jednakosti za sve parove elemenata koji
42       su na istim pozicijama sto znaci da su matrice jednake */
    return 1;
44 }

46 /* Funkcija izracunava zbir dve kvadratne matice */
void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
48 {
    int i, j;
50
    for(i=0; i<n; i++)
52         for(j=0; j<n; j++)
            c[i][j] = a[i][j] + b[i][j];
54 }

56 /* Funkcija izracunava proizvod dve kvadratne matice */
void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
58 {
    int i, j, k;
60
    for(i=0; i<n; i++)
62         for(j=0; j<n; j++) {
            /* Mnozimo i-tu vrstu prve sa j-tom kolonom druge matrice */
64             c[i][j] = 0;
            for(k=0; k<n; k++)
66                 c[i][j] += a[i][k] * b[k][j];
        }
68 }

70 int main()
{
72     /* Matrice ciji se elementi zadaju sa ulaza */
    int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
74
    /* Matrice zbira i proizvoda */
76     int zbir[MAX][MAX], proizvod[MAX][MAX];
78
    /* Dimenzija matrica */
    int n;
80     int i, j;

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

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

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

```
    ucitaj_matricu(b, n);
96
    /* Izracunavamo zbir i proizvod matrica */
98    saberi(a, b, zbir, n);
    pomnozi(a, b, proizvod, n);
100
    /* Ispisujemo rezultat */
102    if(jednake_matrice(a, b, n) == 1)
        printf("da\n");
104    else
        printf("ne\n");
106
    printf("Zbir matrica je:\n");
108    ispisi_matricu(zbir, n);
110
    printf("Proizvod matrica je:\n");
    ispisi_matricu(proizvod, n);
112
    return 0;
114 }
```

### Rešenje 2.10

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

4 #define MAX 64

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

14     /* Obilazimo glavnu dijagonalu matrice. Za elemente na glavnoj
       dijagonali vazi da je indeks vrste jednak indeksu kolone */
16     for(i=0; i<n; i++) {
        if(m[i][i] != 1)
18         return 0;
    }
20
    return 1;
22 }

24 /* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono
   je odredjeno matricom koja sadrzi sve elemente polazne matrice
26   dopunjene jedinicama na glavnoj dijagonali */
void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
28 {
    int i, j;
30
    /* Prepisujemo vrednosti elemenata matrice pocetne matrice */
32     for(i=0; i<n; i++)
```

```

34     for(j=0; j<n; j++)
        zatvorenje[i][j] = m[i][j];

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

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

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

56         return 1;
58     }

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

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

72         /* Odredjujemo simetricno zatvorenje matrice */
        for(i=0; i<n; i++)
            for(j=0; j<n; j++)
74                 if(zatvorenje[i][j] == 1)
                    zatvorenje[j][i] = 1;

76         }

78     }

80
82     /* 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
84     elementom "k", onda je i element "i" u relaciji sa elementom
    "k" */
86     int tranzitivnost (int m[][MAX], int n)
    {
88         int i, j, k;

```

```

90     for(i=0; i<n; i++)
91         for(j=0; j<n; j++)
92             /* Pokusavamo da pronadjemo element koji narusava
93              * tranzitivnost */
94             for(k=0; k<n; k++)
95                 if(m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
96                     return 0;
97
98     return 1;
99 }
100
101 /* Funkcija odredjuje reflektivno-tranzitivno zatvorenje
102    zadate relacije koriscenjem Varsalovog algoritma */
103 void tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
104 {
105     int i, j, k;
106
107     /* Kopiramo pocetnu matricu u matricu rezultata */
108     for(i=0; i<n; i++)
109         for(j=0; j<n; j++)
110             zatvorenje[i][j] = m[i][j];
111
112     /* Primenom Varsalovog algoritma odredjujemo
113        reflektivno-tranzitivno zatvorenje matrice */
114     for(k=0; k<n; k++)
115         for(i=0; i<n; i++)
116             for(j=0; j<n; j++)
117                 if((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
118                    && (zatvorenje[i][j] == 0))
119                     zatvorenje[i][j] = 1;
120 }
121
122 /* Funkcija ispisuje elemente matrice */
123 void pisi_matricu(int m[][MAX], int n)
124 {
125     int i, j;
126
127     for(i=0; i<n; i++) {
128         for(j=0; j<n; j++)
129             printf("%d ", m[i][j]);
130         printf("\n");
131     }
132 }
133
134 int main(int argc, char* argv[])
135 {
136     FILE* ulaz;
137     int m[MAX][MAX];
138     int pomocna[MAX][MAX];
139     int n, i, j, k;
140
141     /* Ako korisnik nije uneo trazene argumente,
142        prijavljujemo gresku */
143     if(argc < 2) {

```

```

146     printf("Greska: ");
147     printf("Nedovoljan broj argumenata komandne linije.\n");
148     printf("Program se poziva sa %s ime_dat.\n", argv[0]);
149     exit(EXIT_FAILURE);
150 }
151
152 /* Otvaramo datoteku za citanje */
153 ulaz = fopen(argv[1], "r");
154 if(ulaz == NULL) {
155     fprintf(stderr, "Greska: ");
156     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
157             argv[1]);
158     exit(EXIT_FAILURE);
159 }
160
161 /* Ucitavamo dimenziju matrice */
162 fscanf(ulaz, "%d", &n);
163
164 /* Proveravamo da li je prekoraceno ogranicenje */
165 if( n > MAX || n <= 0) {
166     fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
167     fprintf(stderr, "matrice.\n");
168     exit(EXIT_FAILURE);
169 }
170
171 /* Ucitavamo element po element matrice */
172 for(i=0; i<n; i++)
173     for(j=0; j<n; j++)
174         fscanf(ulaz, "%d", &m[i][j]);
175
176 /* Ispisujemo trazene vrednosti */
177 printf("Refleksivnost: %s\n",
178        refleksivnost(m, n) == 1 ? "da" : "ne");
179
180 printf("Simetricnost: %s\n",
181        simetricnost(m, n) == 1 ? "da" : "ne");
182
183 printf("Tranzitivnost: %s\n",
184        tranzitivnost(m, n) == 1 ? "da" : "ne");
185
186 printf("Refleksivno zatvorenje:\n");
187 ref_zatvorenje(m, n, pomocna);
188 pisi_matricu(pomocna, n);
189
190 printf("Simetricno zatvorenje:\n");
191 sim_zatvorenje(m, n, pomocna);
192 pisi_matricu(pomocna, n);
193
194 printf("Refleksivno-tranzitivno zatvorenje:\n");
195 tran_zatvorenje(m, n, pomocna);
196 pisi_matricu(pomocna, n);
197
198 /* Zatvaramo datoteku */
199 fclose(ulaz);
200
201 return 0;

```

```
}
```

### Rešenje 2.11

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 32
5
6 int max_sporedna_dijagonala(int m[][MAX], int n)
7 {
8     int i, j;
9     /* Trazimo najveći element na sporednoj dijagonali. Za
10        elemente sporedne dijagonale vazi da je zbir indeksa vrste
11        i indeksa kolone jednak n-1. Za početnu vrednost maksimuma
12        uzimamo element u gornjem desnom uglu */
13     int max_na_sporednoj_dijagonali = m[0][n-1];
14     for(i=1; i<n; i++)
15         if(m[i][n-1-i] > max_na_sporednoj_dijagonali)
16             max_na_sporednoj_dijagonali = m[i][n-1-i];
17
18     return max_na_sporednoj_dijagonali;
19 }
20
21 /* Funkcija izracunava indeks kolone najmanjeg elementa */
22 int indeks_min(int m[][MAX], int n)
23 {
24     int i, j;
25     /* Za početnu vrednost minimuma uzimamo element u gornjem
26        levom uglu */
27     int min=m[0][0], indeks_kolone=0;
28
29     for(i=0; i<n; i++)
30         for(j=0; j<n; j++)
31             /* Ako je tekuci element manji od minimalnog */
32             if(m[i][j]<min) {
33                 /* cuvamo njegovu vrednost */
34                 min=m[i][j];
35                 /* i cuvamo indeks kolone u kojoj se nalazi */
36                 indeks_kolone=j;
37             }
38     return indeks_kolone;
39 }
40
41 /* Funkcija izracunava indeks vrste najveceg elementa */
42 int indeks_max(int m[][MAX], int n) {
43     int i, j;
44     /* Za maksimalni element uzimamo gornji levi ugao */
45     int max=m[0][0], indeks_vrste=0;
46
47     for(i=0; i<n; i++)
48         for(j=0; j<n; j++)
49             /* Ako je tekuci element manji od minimalnog */
50             if(m[i][j]>max) {
51                 /* cuvamo njegovu vrednost */
```



```

        max=m[i][j];
53     /* i cuvamo indeks vrste u kojoj se nalazi */
        indeks_vrste=i;
55     }
    return indeks_vrste;
57 }

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

63     int broj_negativnih=0;

65     for(i=0; i<n; i++)
        for(j=0; j<n; j++)
67         if(m[i][j]<0)
            broj_negativnih++;
69     return broj_negativnih;
}

71
int main(int argc, char* argv[])
73 {
    int m[MAX][MAX];
75     int n;
    int i, j;

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

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

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

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

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

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

```

```
109  /* Prekidamo izvršavanje programa */
      return 0;
111 }
```

### Rešenje 2.12

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

4  #define MAX 32

6  /* Funkcija učitava elemente kvadratne matrice sa
   standardnog ulaza */
8  void ucitaj_matricu(int m[][MAX], int n)
   {
10     int i, j;

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

16

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

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

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

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

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

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

46         /* Ako proizvod bar jedne vrste nije jednak jedinici, odmah
           zakljucujemo da matrica nije normirana */
48         if(proizvod!=1)
```

```

    return 0;
50 }

52 /* Proveravamo uslov ortogonalnosti, odnosno da li je proizvod
    dve bilo koje razlicite vrste matrice jednak nuli */
54 for(i=0; i<n-1; i++) {
    for(j=i+1; j<n; j++) {
56
58         /* Izracunavamo skalarni proizvod */
        proizvod = 0;

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

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

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

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

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

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

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

92
94     /* Ispisujemo rezultat rada funkcije */
    if(ortonormirana(A,n))
        printf("da\n");
96     else
        printf("ne\n");
98
100     return 0;
}

```

## Rešenje 2.13

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_V 10
5  #define MAX_K 10
6
7  /* Funkcija proverava da li su ispisani svi elementi iz matrice,
8     odnosno da li se nariusio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
10 {
11     return !(top <= bottom && left <= right);
12 }
13
14 /* Funkcija spiralno ispisuje elemente matrice */
15 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16 {
17     int i,j,top, bottom,left, right;
18
19     top=left = 0;
20     bottom=n-1;
21     right = m-1;
22
23     while( !krajIspisa(top, bottom, left, right) ) {
24         /* Ispisuje se prvi red*/
25         for(j=left; j<=right; j++)
26             printf("%d ",a[top][j]);
27
28         /* Spustamo prvi red */
29         top++;
30
31         if(krajIspisa(top,bottom,left,right))
32             break;
33
34         for(i=top; i<=bottom; i++ )
35             printf("%d ",a[i][right]);
36
37         /* Pomeramo desnu kolonu za naredni krug ispisa
38            blize levom kraju */
39         right--;
40
41         if(krajIspisa(top,bottom,left,right))
42             break;
43
44         /* Ispisujemo donju vrstu */
45         for(j=right; j>=left; j-- )
46             printf("%d ",a[bottom][j]);
47
48         /* Podizemo donju vrstu za naredni krug ispisa */
49         bottom--;
50
51         if(krajIspisa(top,bottom,left,right))
52             break;
53
54         /* Ispisujemo prvu kolonu*/
55         for(i=bottom; i>=top; i-- )
```

```

56     printf("%d ",a[i][left]);
58     /* Pripremamo levu kolonu za naredni krug ispisa */
59     left++;
60 }
61 putchar('\n');
62 }
63
64 void ucitaj_matricu(int a[][MAX_K], int n, int m)
65 {
66     int i, j;
67
68     for(i=0 ;i<n; i++)
69         for(j=0; j<m; j++)
70             scanf("%d", &a[i][j]);
71 }
72
73 int main( )
74 {
75     int a[MAX_V][MAX_K];
76     int m,n;
77
78     /* Ucitaj broj vrsta i broj kolona matrice */
79     scanf("%d",&n);
80     scanf("%d", &m);
81
82     if( n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
83         fprintf(stderr, "Greska: neodgovarajuće dimenzije ");
84         fprintf(stderr, "matrice.\n");
85         exit(EXIT_FAILURE);
86     }
87
88     ucitaj_matricu(a, n, m);
89     ispisi_matricu_spiralno(a, n, m);
90
91     return 0;
92 }

```

### Rešenje 2.14

### Rešenje 2.15

```

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

```

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

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

22

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

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

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

38  /* Ispisujemo elemente niza unazad */
    for (i = n - 1; i >= 0; i--)
40         printf("%d ", p[i]);
    printf("\n");
42

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

46  return 0;
}
```

### Rešenje 2.16

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

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

10    /* Velicina alocirane memorije */
    int alocirano;

12

    /* Broj elemenata niza */
14    int n;

16    /* Broj koji se ucitava sa ulaza */
    int x;
```

```
18  int i;
19  int* b = NULL;
20
21  /* Inicijalizacija */
22  alocirano = n = 0;
23
24  /* Unosimo brojeve sa ulaza */
25  scanf("%d", &x);
26
27  /* Sve dok je procitani broj razlicit od nule... */
28  while(x!=0) {
29
30      /* Ako broj ucitanih elemenata niza odgovara broju
31         alociranih mesta, za smestanje novog elementa treba
32         obezbediti dodatni prostor. Da se ne bi za svaki sledeci
33         element pojedinačno alocirala memorija, prilikom
34         alokacije se vrši rezervacija za još KORAK dodatnih
35         mesta za buduće elemente */
36      if(n == alocirano) {
37          /* Povecava se broj alociranih mesta */
38          alocirano = alocirano + KORAK;
39
40          /* Vrši se realokacija memorije sa novom velicinom */
41          /******
42          /* Resenje sa funkcijom malloc() */
43          /******
44          /* Vrši se alokacija memorije sa novom velicinom, a adresa
45             početka novog memorijskog bloka se čuva u
46             promenljivoj b */
47          b = (int*) malloc (alocirano * sizeof(int));
48
49          /* Ako prilikom alokacije dodje do neke greske */
50          if(b == NULL) {
51              /* poruku ispisujemo na izlaz za greske */
52              fprintf(stderr, "malloc(): ");
53              fprintf(stderr, "greska pri alokaciji memorije.\n");
54
55              /* Pre kraja programa moramo svu dinamički alociranu
56                 memoriju da oslobodimo. U ovom slučaju samo memoriju
57                 na adresi a */
58              free(a);
59
60              /* Završavamo program */
61              exit(EXIT_FAILURE);
62          }
63
64          /* Svih n elemenata koji počinju na adresi a prepisujemo
65             na novu adresu b */
66          for(i = 0; i < n; i++)
67              b[i] = a[i];
68
69          /* Posle prepisivanja oslobadjamo blok memorije sa početnom
70             adresom u a */
71          free(a);
72
73          /* Promenljivoj a dodeljujemo adresu početka novog, većeg
```

```
74         bloka koji je prilikom alokacije zapamcen u
           promenljivoj b */
76     a = b;

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

86     if(a == NULL) {
           fprintf(stderr, "realloc(): ");
88         fprintf(stderr, "greska pri alokaciji memorije.\n");
           exit(EXIT_FAILURE);
90     }
       */
92 }

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

96
98 /* i učitavamo sledeci element */
scanf("%d", &x);
100 }

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

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

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

### Rešenje 2.17

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX 1000
6
/* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
8   Ovako nesto se moze improvizovati tako sto se u funkciji
   dinamički kreira niz potrebne velicine, popuni se potrebnim
10   informacijama, a zatim se vrati njegova adresa. Imajuci u
   vidu cinjenicu da dinamički kreiran objekat ne nestaje
12   kada se izadje iz funkcije koja ga je kreirala, vraceni
   pokazivac se kasnije u pozivajucoj funkciji moze koristiti
14   za pristup "vracenom" nizu. Medjutim, pozivajuca funkcija
```



```

    ima odgovornost i da se brine o dealokaciji istog prostora */
16
/* Funkcija dinamički kreira niz karaktera u koji smesta
18 rezultat nadovezivanja niski. Adresa niza se vraća kao
    povratna vrednost. */
20 char *nadovezi(char *s, char *t) {
    /* Dinamički kreiramo prostor dovoljne velicine */
22 char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
                            * sizeof(char));
24
    /* Proveravamo uspeh alokacije */
26 if (p == NULL) {
        fprintf(stderr, "malloc(): ");
28 fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
30 }

32 /* Kopiramo i nadovezujemo stringove */

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

40 for(j=0; t[j]!='\0'; i++, j++)
    p[i]=t[j];
42
    p[i]='\0';
44 */

46 /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
    string.h */
48 strcpy(p, s);
    strcat(p, t);
50

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

54
int main() {
56 char *s = NULL;
    char s1[MAX], s2[MAX];
58

    /* Ucitavamo dve niske koje cemo da nadovezemo */
60 scanf("%s", s1);
    scanf("%s", s2);
62

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

66 /* Prikazujemo rezultat */
    printf("%s\n", s);
68

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

```

```
72     return 0;
    }
```

### Rešenje 2.18

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

int main()
{
    int i,j;

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

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

    /* Trag matrice */
    double trag = 0;

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

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

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

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

        if(A[i] == NULL) {
            /* Alokacija je neuspesna. Pre zavrsetka programa
               moramo da oslobodimo svih i-1 prethodno alociranih
               vrsta, i alociran niz pokazivaca */
            for( j=0; j<i; j++)
                free(A[j]);
            free(A);

            exit( EXIT_FAILURE);
        }
    }

    /* Unosimo sa standardnog ulaza brojeve u matricu.
       Popunjavamo vrstu po vrstu */
```

```

50     for(i = 0; i < n; i++)
51         for(j = 0; j < m; j++ )
52             scanf("%lf", &A[i][j]);

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

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

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

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

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

70     return 0;
}

```

### Rešenje 2.19

```

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

5  void ucitaj_matricu(int ** M, int n, int m)
6  {
7      int i, j;

9      /* Popunjavamo matricu vrstu po vrstu */
10     for(i=0; i<n; i++)
11         /* Popunjavamo i-tu vrstu matrice */
12         for(j=0; j<m; j++)
13             scanf("%d", &M[i][j]);
14 }

15 void ispisuje_elemente_ispod_dijagonale(int ** M, int n, int m)
16 {
17     int i, j;

19     for(i=0; i<n; i++) {
20         /* Ispisujemo elemente ispod glavne dijagonale matrice */
21         for(j=0; j<=i; j++)
22             printf("%d ", M[i][j]);
23         printf("\n");
24     }
25 }

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

```

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

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

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

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

55  ucitaj_matricu(matrica, n, m);

57  ispisi_elemente_ispod_dijagonale(matrica, n, m);

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

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

67  return 0;
69 }
```

### Rešenje 2.20

```
#include<stdio.h>

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

### Rešenje 2.21

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

```

4
/* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni (float** a, int n)
{
8     int i, j;

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

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

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

40    return zbir;
}

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

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

52
/* Funkcija ispisuje elemente kvadratne matrice dimenzije n
54    na standardni izlaz */
void ispisi_matricu(float** m, int n) {
56    int i, j;

58    for(i=0; i<n; i++){
        for(j=0; j<n; j++)

```

```
60     printf("%.2f ", m[i][j]);
61     printf("\n");
62 }
63 }
64
65 /* Funkcija alokira memoriju za kvadratnu matricu dimenzije n */
66 float** alociraj_memoriju(int n) {
67     int i, j;
68     float** m;
69
70     m = (float**) malloc(n * sizeof(float*));
71     if(m == NULL) {
72         fprintf(stderr, "malloc(): Neuspela alokacija\n");
73         exit(EXIT_FAILURE);
74     }
75
76     /* Za svaku vrstu matrice */
77     for(i=0; i<n; i++) {
78         /* Alociramo memoriju */
79         m[i] = (float*) malloc(n * sizeof(float));
80
81         /* Proveravamo da li je doslo do greske pri alokaciji */
82         if(m[i] == NULL) {
83             /* Ako jeste, ispisujemo poruku */
84             printf("malloc(): neuspela alokacija memorije!\n");
85
86             /* Oslobadjamo memoriju zauzetu do ovog koraka */
87             for(j=0; j<i; j++)
88                 free(m[j]);
89             free(m);
90             exit(EXIT_FAILURE);
91         }
92     }
93     return m;
94 }
95
96 /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom
97    dimenzije n */
98 void oslobodi_memoriju(float** m, int n)
99 {
100     int i;
101
102     for(i=0; i<n; i++)
103         free(m[i]);
104     free(m);
105 }
106
107 int main(int argc, char* argv[])
108 {
109     FILE* ulaz;
110     float** a;
111     int n;
112
113     /* Ako korisnik nije uneo trazene argumente,
114        prijavljujemo gresku */
115     if(argc < 2) {
```

```
116     printf("Greska: ");
117     printf("Nedovoljan broj argumenata komandne linije.\n");
118     printf("Program se poziva sa %s ime_dat.\n", argv[0]);
119     exit(EXIT_FAILURE);
120 }
121
122 /* Otvaramo datoteku za citanje */
123 ulaz = fopen(argv[1], "r");
124 if(ulaz == NULL) {
125     fprintf(stderr, "Greska: ");
126     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
127                                     argv[1]);
128     exit(EXIT_FAILURE);
129 }
130
131 /* citamo dimenziju matrice */
132 fscanf(ulaz, "%d", &n);
133
134 /* Alociramo memoriju */
135 a = alociraj_memoriju(n);
136
137 /* Ucitavamo elemente matrice */
138 ucitaj_matricu(ulaz, a, n);
139
140 float zbir = zbir_ispod_sporedne_dijagonale(a, n);
141
142 /* Pozivamo funkciju za modifikovanje elemenata */
143 izmeni(a, n);
144
145 /* Ispisujemo rezultat */
146 printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
147 printf("je %.2f.\n", zbir);
148
149 printf("Transformisana matrica je:\n");
150 ispisi_matricu(a,n);
151
152 /* Oslobadjamo memoriju */
153 oslobodi_memoriju(a, n);
154
155 /* Zatvaramo datoteku */
156 fclose(ulaz);
157
158 /* i prekidamo sa izvršavanjem programa */
159 return 0;
160 }
```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

## Rešenje 2.26

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



```

int main(int argc, char *argv[])
56 {
    double a, b;
58     int n;
    /* Imena funkcija koja ćemo navoditi su čkraa ili čtano duga
60     5 karaktera */
    char ime_fje[6];
62     /* Pokazivac na funkciju koja ima jedan argument tipa double i
        povratnu vrednost istog tipa */
64     double (*fp)(double);

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

        exit(EXIT_FAILURE);
74     }

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

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

```

```
112     }
113
114     printf("Unesite krajeve intervala:\n" );
115     scanf("%lf %lf", &a, &b);
116
117     printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
118     printf("(ukljucujuci krajeve intervala)?\n");
119     scanf("%d", &n );
120
121     /* Mreza mora da čukljuuje bar krajeve intervala,
122        tako da se mora uneti broj veci od 2 */
123     if (n < 2) {
124         fprintf(stderr, "Broj čtaaka žmree mora biti bar 2!\n");
125         exit(EXIT_FAILURE);
126     }
127
128     /* Ispisujemo ime funkcije */
129     printf("      x %10s(x)\n", ime_fje);
130
131     /* dProsleujemo funkciji tabela() funkciju zadatu kao
132        argument komandne linije */
133     tabela(a, b, n, fp);
134
135     exit(EXIT_SUCCESS);
136 }
```

Rešenje [2.27](#)

Rešenje [2.28](#)

Rešenje [2.29](#)

## Glava 3

# Algoritmi pretrage i sortiranja

### 3.1 Pretraživanje

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

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

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

*Test 1*

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

```
Izlaz:
```

```
Linearna pretraga
```

```
Element nije u nizu
```

```
-----
```

```
Binarna pretraga
```

```
Element nije u nizu
```

```
-----
```

```
Interpolaciona pretraga
```

```
Element nije u nizu
```

```
-----
```

**Zadatak 3.2** Napisati rekurzivne funkcije algoritama linearne, binarne i interpolacione pretrage i program koji ih testira za brojeve koji se unose sa standardnog

### 3 Algoritmi pretrage i sortiranja

---

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

<i>Test 1</i>	<i>Test 2</i>
Ulaz: 11 2 5 6 8 10 11 23	Ulaz: 14 10 32 35 43 66 89 100
Izlaz:	Izlaz:
Linearna pretraga	Linearna pretraga
Pozicija elementa je 5.	Element se ne nalazi u nizu.
Binarna pretraga	Binarna pretraga
Pozicija elementa je 5.	Element se ne nalazi u nizu.
Interpolaciona pretraga	Interpolaciona pretraga
Pozicija elementa je 5.	Element se ne nalazi u nizu.

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

*Test 1*

```
Datoteka:
20140003 Marina Petrovic
20140012 Stefan Mitrovic
20140032 Dejan Popovic
20140049 Mirko Brankovic
20140076 Sonja Stevanovic
20140104 Ivan Popovic
20140187 Vlada Stankovic
20140234 Darko Brankovic
Ulaz:
20140076
Popovic
Izlaz:
Indeks: 20140076, Ime i prezime: Sonja Stevanovic
Indeks: 20140032, Ime i prezime: Dejan Popovic
```

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

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

*Test 1*

```

|| Poziv: ./a.out dat.txt -x
|| Datoteka:
|| 12 53
|| 2.342 34.1
|| -0.3 23
|| -1 23.1
|| 123.5 756.12
|| Izlaz: -0.3 23

```

**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

```

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

*Test 1*

```

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

```

*Test 2*

```

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

```

*Test 3*

```

|| Ulaz: -100 -15 0 13
|| 155 124 258
|| 315 516 7000
|| Izlaz: 3

```

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

*Test 1*

```

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

```

*Test 2*

```

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

```

*Test 3*

```
|| Ulaz:  100 15 11 8 7 5 4 3 2
|| Izlaz: -1
```

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

- (a) Napisati funkciju, linearne složenosti, koja određuje logaritam pomeranjem broja udesno dok ne postane 0.
- (b) Napisati funkciju, logaritmske složenosti, koja određuje logaritam koristeći binarnu pretragu.

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

*Test 1*

```
|| Ulaz:  10
|| Izlaz: 3 3
```

*Test 2*

```
|| Ulaz:  4
|| Izlaz: 2 2
```

*Test 3*

```
|| Ulaz:  17
|| Izlaz: 4 3
```

*Test 4*

```
|| Ulaz:  1031
|| Izlaz: 10 10
```

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

*Test 1*

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

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

```
Osoba niz_osoba[]={{"Mika", "Antic"},
                   {"Dobrica", "Eric"},
```

```

{"Desanka", "Maksimovic"},
{"Dusko", "Radovic"},
{"Ljubivoje", "Rsumovic"};

```

*Test 1*

```

|| Ulaz:  R
|| Izlaz: Dusko Radovic

```

## 3.2 Sortiranje

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

*Test 1*

```

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

```

*Test 2*

```

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

```

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

*Test 1*

```

|| Ulaz:  anagram ramgana
|| Izlaz:  jesu

```

*Test 2*

```

|| Ulaz:  anagram anagrm
|| Izlaz:  nisu

```

**Zadatak 3.14** Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza, i neće biti duži od 256 i kraći od jednog elemenata. Uputstvo: prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.

*Test 1*

```

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

```

*Test 2*

```

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

```

### 3 Algoritmi pretrage i sortiranja

---

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

	<i>Test 1</i>		<i>Test 2</i>
	Ulaz: 34 134 4 1 6 30 23		Ulaz: 12 53 1 43 3 56 13
	Izlaz: da		Izlaz: ne

**Zadatak 3.16** Napisati funkciju potpisa `int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3, int dim3)` koja prima dva sortirana niza, i na osnovu njih pravi novi sortirani niz koji koji sadrži elemente oba niza. Treća dimenzija predstavlja veličinu niza u koji se smešta rezultat. Ako je ona manja od potrebne dužine, funkcija vraća -1, kao indikator neuspeha, inače vraća 0. Napisati i program koji testira funkciju, u kome se nizovi unose sa standardnog ulaza, sve dok se ne unese 0. Dimenzija nizova neće biti preko 256.

	<i>Test 1</i>
	Ulaz: 3 6 7 11 14 35 0 3 5 8 0
	Izlaz: 3 3 5 6 7 8 11 14 35

	<i>Test 2</i>
	Ulaz: 1 4 7 0 9 11 23 54 75 0
	Izlaz: 1 4 7 9 11 23 54 75

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



*Test 1*

```

Poziv: ./a.out prvi-deo.txt drugi-deo.txt
Ulazne datoteke:
    prvi-deo.txt:          drugi-deo.txt:

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

```

```

Izlazna datoteka (ceo-tok.txt):

```

```

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

```

**Zadatak 3.18** Napraviti biblioteku `sort.h` i `sort.c` koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži `selection`, `merge`, `quick`, `bubble`, `insertion` i `shell sort`. Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na već sortiranim nizovima i na naopako sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Vreme meriti programom `time`. Analizirati porast vremena sa porastom dimenzije `n`.

*Upotreba programa 1*

```

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

```

*Upotreba programa 2*

```

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

```

#### *Upotreba programa 3*

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

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

- (a) njihovog rastojanja od koordinatnog početka,
- (b) x koordinata tačaka,
- (c) y koordinata tačaka.

Napisati program koji učitava niz tačaka iz datoteke čije se ime zadaje kao drugi argument komandne linije, i u zavisnosti od prisutnih opcija (prvi argument) u komandnoj liniji (`-o`, `-x` ili `-y`), sortira tačke po jednom od prethodna tri kriterijuma i rezultat upisuje u datoteku čije se ime zadaje kao treći argument komandne linije. U ulaznoj datoteci nije zadato više od 128 tačaka.

#### *Test 1*

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

#### *Test 2*

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

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

*Test 1*

```

Ulazna datoteka (biracki-spisak.txt):
Andrija Petrovic
Anja Ilic
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Ivana Markovic
Lazar Micic
Marija Stankovic
Izlaz: 3

```

**Zadatak 3.21** Definisana je struktura podataka

```

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

```

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

*Test 1*

```

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

```

**Zadatak 3.22** Napisati funkciju koja sortira niz niski po broju suglasnika u niski, ukoliko reči imaju isti broj suglasnika tada po dužini niske, a ukoliko su i

### 3 Algoritmi pretrage i sortiranja

---

dužine jednake onda leksikografski. Napisati program koji testira ovu funkciju za niske koje se zadaju u datoteci `niske.txt`. Pretpostaviti da u nizu nema više od 128 elemenata, kao i da svaka niska sadrži najviše 31 karakter.

#### *Test 1*

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

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

*Upotreba programa 1*

```

Ulazna datoteka (artikli.txt):
1001 Keks Jaffa 120
2530 Napolitanke Bambi 230
0023 Medeno_srce Pionir 150
2145 Pardon Marbo 70
Interakcija programa:
Asortiman:
KOD          Naziv artikla      Ime proizvođača      Cena
      23          Medeno_srce      Pionir      150.00
    1001          Keks          Jaffa      120.00
    2145          Pardon      Marbo      70.00
    2530      Napolitanke      Bambi      230.00
-----
- Za kraj za kraj rada kase, pritisnite CTRL+D!
- Za nov racun unesite kod artikla!

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

  UKUPNO: 270.00 dinara.

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

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

  UKUPNO: 230.00 dinara.

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

Kraj rada kase!

```

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

### ***3 Algoritmi pretrage i sortiranja***

---

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

### Test 1

```
Ulazna datoteka (aktivnosti.txt):
Aleksandra Cvetic 4 6
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Marija Stankovic 1 3
Ognjen Peric 1 2
Uros Milic 2 5
Andrija Petrovic 2 5
Anja Ilic 3 1
Lazar Micic 1 3
Nenad Brankovic 2 4
Izlazna datoteka (dat1.txt):
Studenti sortirani po imenu leksikografski rastuce:
Aleksandra Cvetic 4 6
Andrija Petrovic 2 5
Anja Ilic 3 1
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Lazar Micic 1 3
Marija Stankovic 1 3
Nenad Brankovic 2 4
Ognjen Peric 1 2
Uros Milic 2 5
Izlazna datoteka (dat2.txt):
Studenti sortirani po broju zadataka opadajuce,
pa po duzini imena rastuce:
Aleksandra Cvetic 4 6
Uros Milic 2 5
Dragan Markovic 3 5
Andrija Petrovic 2 5
Nenad Brankovic 2 4
Lazar Micic 1 3
Bojan Golubovic 4 3
Marija Stankovic 1 3
Ognjen Peric 1 2
Anja Ilic 3 1
Ivana Stankovic 3 1
Izlazna datoteka (dat3.txt):
Studenti sortirani po prisustvu opadajuce,
pa po broju zadataka,
pa po prezimenima leksikografski opadajuce:
Aleksandra Cvetic 4 6
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Anja Ilic 3 1
Andrija Petrovic 2 5
Uros Milic 2 5
Nenad Brankovic 2 4
Marija Stankovic 1 3
Lazar Micic 1 3
Ognjen Peric 1 2
```

### 3 Algoritmi pretrage i sortiranja

---

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

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

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

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

Na standardni izlaz ispisati informacije o pesmama sortirane na opisan način.

- (a) Uraditi zadatak uz pretpostavku da je maksimalna dužina imena izvođača 20 karaktera, a imena naslova pesme 50 karaktera.
- (b) Uraditi zadatak bez pravljenja pretpostavki o maksimalnoj dužini imena izvođača i naslova pesme.

Jelena: Kako ovde da prikažem rešenje pod a) i b)? Milena: Mislim da bi bilo najbolje razdvojiti kod po datotekama i napisati dve main funkcije. Na taj način onda nedupliramo dva programa, već oba uključuju zajednicke delove, ukoliko ima dovoljno tih zajednickih delova. Ukoliko ih ima malo, onda svaki zadatak posebno.

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



*Test 3*

```

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

```

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

*Test 1*

```

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

```

**\*\* Zadatak 3.27** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze najmanja, najveća, itd... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene.

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

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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

#### Upotreba programa 1

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

#### Upotreba programa 2

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

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

#### Upotreba programa 1

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

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

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

Niske se učitavaju iz fajla `niske.txt`, neće ih biti više od 1000, i svaka će biti dužine najviše 30 karaktera. Program prvo leksikografski sortira niz, primenjuje binarnu pretragu (`bsearch`) zarad traženja niske unete sa standardnog ulaza, a

potom linearnu pretragu koristeći funkciju `lfind`. Na kraju, niske bivaju sortirane po dužini. Rezultate svih sortiranja i pretraga ispisati na standardni izlaz.

#### Test 1

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

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

**Zadatak 3.32** Napisati program koji korišćenjem bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova, ili poruka ukoliko nema takvog. Potom, sa standardnom ulaza, unosi se prezime traženog studenta, i prikazuje se osoba sa tim prezimenom, ili poruka da se nijedan student tako ne preziva. Za pretraživanje, koristiti odgovarajuće bibliotečke funkcije. Podaci o studentima čitaju se iz datoteke čije se ime zadaje preko argumenata komandne linije. Za svakog studenta u datoteci postoje ime, prezime i bodovi osvojeni na kolokvijumu. Pretpostaviti da neće biti više od 500 studenata, i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

#### Test 1

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

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

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

#### Test 1

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

#### Test 2

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

**Zadatak 3.35** Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr97125`, `mm09001`), ime i prezime i broj poena. Napisati program koji sortira (korišćenjem funkcije `qsort`) studente po

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

#### Test 1

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

**Zadatak 3.36** Definisana je struktura:

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

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

#### Test 1

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

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

#### Test 1

```
Interakcija programa:
  Unesite dimenzije matrice: 3 2
  Unesite elemente matrice po vrstama:
  6 -5
  -4 3
  2 1
  Sortirana matrica je:
  -4 3
  6 -5
  2 1
```

**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 standardni izlaz.

## 3.4 Rešenja

### Rešenje 3.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
7    -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz celih brojeva duzine n, trazeci u
10    njemu element x. Pretraga se vrši prostom iteracijom kroz
11    niz. Ako se element pronadje funkcija vraca indeks pozicije
12    na kojoj je pronadjen. Ovaj indeks je uvek nenegativan. Ako
13    element nije pronadjen u nizu, funkcija vraca -1, kao
14    indikator neuspesne 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
25    indeks pozicije nadjenog elementa ili -1, ako element nije
26    pronadjen */
27 int binarna_pretraga(int a[], int n, int x)
28 {
```

```

29  int levi = 0;
    int desni = n - 1;
31  int srednji;
    /* Dokle god je indeks levi levo od indeksa desni */
33  while (levi <= desni) {
    /* Racunamo srednji indeks */
35    srednji = (levi + desni) / 2;
    /* Ako je srednji element veci od x, tada se x mora nalaziti
37    u levoj polovini niza */
    if (x < a[srednji])
39      desni = srednji - 1;
    /* Ako je srednji element manji od x, tada se x mora
41    nalaziti u desnoj polovini niza */
    else if (x > a[srednji])
43      levi = srednji + 1;
    else
45      /* Ako je srednji element jednak x, tada smo pronasli x na
        poziciji srednji */
47      return srednji;
    }
49  /* Ako nije pronadjen vratamo -1 */
    return -1;
51 }

53 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
    indeks pozicije nadjenog elementa ili -1, ako element nije
55    pronadjen */
    int interpolaciona_pretraga(int a[], int n, int x)
57 {
    int levi = 0;
59    int desni = n - 1;
    int srednji;
61    /* Dokle god je indeks levi levo od indeksa desni... */
    while (levi <= desni) {
63      /* Ako je element manji od pocetnog ili veci od poslednjeg
        clana u delu niza a[levi],...,a[desni] tada nije u tom
65      delu niza. Ova provera je neophodna, da se ne bi dogodilo
        da se prilikom izracunavanja indeksa srednji izadje izvan
67      opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
69          return -1;
        /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
71      a[levi] i a[desni] jednaki, tada je jasno da je x jednako
        ovim vrednostima, pa vratamo indeks levi (ili indeks
73      desni, sve jedno je).Ova provera je neophodna, zato sto
        bismo inace prilikom izracunavanja srednji imali deljenje
75      nulom. */
        else if (a[levi] == a[desni])
77          return levi;
        /* Racunamo srednji indeks */
79      srednji =
        levi +
81      ((double) (x - a[levi]) / (a[desni] - a[levi])) *
        (desni - levi);
83      /* Napomena: Indeks srednji je uvek izmedju levi i desni,
        ali ce verovatno biti blize trazenoj vrednosti nego da

```

### 3 Algoritmi pretrage i sortiranja

```
85     smo prosto uvek uzimali srednji element. Ovo se moze
86     porediti sa pretragom rečnika: ako neko trazi rec na
87     slovo 'B', sigurno neće da otvori rečnik na polovini, vec
      verovatno negde blize pocetku. */
89     /* Ako je srednji element veci od x, tada se x mora nalaziti
      u levoj polovini niza */
91     if (x < a[srednji])
        desni = srednji - 1;
93     /* Ako je srednji element manji od x, tada se x mora
      nalaziti u desnoj polovini niza */
95     else if (x > a[srednji])
        levi = srednji + 1;
97     else
        /* Ako je srednji element jednak x, tada smo pronasli x na
99         poziciji srednji */
        return srednji;
101 }
102 /* Ako nije pronadjen vratamo -1 */
103 return -1;
104 }
105
106 /* Funkcija main */
107 int main(int argc, char **argv)
108 {
109     int a[MAX];
110     int n, i, x;
111     struct timespec time1, time2, time3, time4, time5, time6;
112     FILE *f;
113
114     /* Provera argumenata komandne linije */
115     if (argc != 3) {
116         fprintf(stderr,
117             "koriscenje programa: %s dim_niza trazen_br\n",
118             argv[0]);
119         exit(EXIT_FAILURE);
120     }
121
122     /* Dimenzija niza */
123     n = atoi(argv[1]);
124     if (n > MAX || n <= 0) {
125         fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
126         exit(EXIT_FAILURE);
127     }
128
129     /* Broj koji se trazi */
130     x = atoi(argv[2]);
131
132     /* Elemente niza odredjujemo slucajno, tako da je svaki
133     sledeci veci od prethodnog. srandom() funkcija obezbedjuje
134     novi seed za pozivanje random() funkcije. Kako nas niz ne
135     bi uvek isto izgledao seed smo postavili na tekuće vreme u
136     sekundama od Nove godine 1970. random()%100 daje brojeve
137     izmedju 0 i 99 */
138     srandom(time(NULL));
139     for (i = 0; i < n; i++)
140         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
```



```

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

169  /* Binarna pretraga */
171  printf("Binarna pretraga\n");
173  clock_gettime(CLOCK_REALTIME, &time3);
175  i = binarna_pretraga(a, n, x);
177  clock_gettime(CLOCK_REALTIME, &time4);
179  if (i == -1)
181      printf("Element nije u nizu\n");
183  else
185      printf("Element je u nizu na poziciji %d\n", i);
187  printf("-----\n");

189  /* Interpolaciona pretraga */
191  printf("Interpolaciona pretraga\n");
193  clock_gettime(CLOCK_REALTIME, &time5);
195  i = interpolaciona_pretraga(a, n, x);
197  clock_gettime(CLOCK_REALTIME, &time6);
199  if (i == -1)
201      printf("Element nije u nizu\n");
203  else
205      printf("Element je u nizu na poziciji %d\n", i);
207  printf("-----\n");

209  /* Upisujemo podatke o izvršavanju programa u log fajl */
211  if ((f = fopen("vremena.txt", "a")) == NULL) {
213      fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
215      exit(EXIT_FAILURE);
217  }

219  fprintf(f, "Dimenzija niza od %d elemenata.\n", n);
221  fprintf(f, "\tLinearna pretraga:%10ld ns\n",
223          (time2.tv_sec - time1.tv_sec) * 1000000000 +
224          time2.tv_nsec - time1.tv_nsec);
225  fprintf(f, "\tBinarna: %19ld ns\n",
227          (time4.tv_sec - time3.tv_sec) * 1000000000 +
228          time4.tv_nsec - time3.tv_nsec);
229  fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
231          (time6.tv_sec - time5.tv_sec) * 1000000000 +
232          time6.tv_nsec - time5.tv_nsec);

234  fclose(f);

```

### 3 Algoritmi pretrage i sortiranja

---

```
197     return 0;
199 }
```

#### Rešenje 3.2

```
#include <stdio.h>

2
int lin_pretgraga_rek(int a[], int n, int x)
4 {
    int tmp;
    /* Izlaz iz rekurzije */
    if (n <= 0)
        return -1;
    /* Ako je prvi element trazeni */
    if (a[0] == x) /* if (a[n-1] == x) */
        return 0; /* return n - 1; */
    /* Pretraga ostatka niza */
    tmp = lin_pretgraga_rek(a + 1, n - 1, x);
    return tmp < 0 ? tmp : tmp + 1;
    /* return lin_pretgraga_rek(a, n - 1, x); */
16 }

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

38
int interp_pretgraga_rek(int a[], int l, int d, int x)
40 {
    int p;
    if (x < a[l] || x > a[d])
        return -1;
    if (a[d] == a[l])
        return l;
    /* Pozicija na kojoj se trazi vrednost x */
    p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
    if (a[p] == x)
        return p;
48 }
```

```

50     if (a[p] < x)
51         return interp_pretgraga_rek(a, p + 1, d, x);
52     else
53         return interp_pretgraga_rek(a, 1, p - 1, x);
54 }

56 #define MAX 1024

58 int main()
59 {
60     int a[MAX];
61     int x;
62     int i, indeks;

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

68     /* Ucitavamo elemente niza sve do kraja ulaza - ocekujemo da
        korisnik pritisne CTRL+D za naznaku kraja */
70     i = 0;
71     printf("Unesite sortirani niz elemenata: ");
72     while (scanf("%d", &a[i]) == 1) {
73         i++;
74     }

76     printf("Linearna pretraga\n");
77     indeks = lin_pretgraga_rek(a, i, x);
78     if (indeks == -1)
79         printf("Element se ne nalazi u nizu.\n");
80     else
81         printf("Pozicija elementa je %d.\n", indeks);
82
83     printf("Binarna pretraga\n");
84     indeks = bin_pretgraga_rek(a, 0, i - 1, x);
85     if (indeks == -1)
86         printf("Element se ne nalazi u nizu.\n");
87     else
88         printf("Pozicija elementa je %d.\n", indeks);
89
90     printf("Interpolaciona pretraga\n");
91     indeks = interp_pretgraga_rek(a, 0, i - 1, x);
92     if (indeks == -1)
93         printf("Element se ne nalazi u nizu.\n");
94     else
95         printf("Pozicija elementa je %d.\n", indeks);
96
97     return 0;
98 }

```

### Rešenje 3.3

```

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
5 #define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
7
8 /* O svakom studentu imamo 3 informacije i njih objedinjujemo u
9    strukturu kojom cemo predstavljati svakog studenta. */
10 typedef struct {
11     /* Indeks mora biti tipa long jer su podaci u datoteci
12        preveliki za 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 sortirani prema indeksu, jer cemo
19    ih, redom, kako citamo smestati u niz, a u datoteci su vec
20    smesteni sortirani rastuce prema broju indeksa. Iz tog
21    razloga pretragu po indeksu cemo vrsiti binarnom pretragom,
22    dok pretragu po prezimenu moramo vrsiti linearno, jer nemamo
23    garancije da postoji uredjenje po prezimenu. */
24
25 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
26    indeks pozicije nadjenog elementa ili -1, ako element nije
27    pronadjen */
28 int binarna_pretraga(Student a[], int n, long x)
29 {
30     int levi = 0;
31     int desni = n - 1;
32     int srednji;
33     /* Dokle god je indeks levi levo od indeksa desni */
34     while (levi <= desni) {
35         /* Racunamo srednji indeks */
36         srednji = (levi + desni) / 2;
37         /* Ako je srednji element veci od x, tada se x mora nalaziti
38            u levoj polovini niza */
39         if (x < a[srednji].indeks)
40             desni = srednji - 1;
41         /* Ako je srednji element manji od x, tada se x mora
42            nalaziti u desnoj polovini niza */
43         else if (x > a[srednji].indeks)
44             levi = srednji + 1;
45         else
46             /* Ako je srednji element jednak x, tada smo pronasli x na
47                poziciji srednji */
48             return srednji;
49     }
50     /* Ako nije pronadjen vratimo -1 */
51     return -1;
52 }
53
54 int linearna_pretraga(Student a[], int n, char x[])
55 {
56     int i;
57     for (i = 0; i < n; i++)
58         /* Poredimo prezime i-tog studenta i poslato x */
59         if (strcmp(a[i].prezime, x) == 0)
```

```

        return i;
61     return -1;
    }
63
    /* Main funkcija mora imate argumente jer se ime datoteke dobija
65     kao argument komandne linije */
    int main(int argc, char *argv[])
67 {
    /* Ucitacemo redom sve studente iz datoteke u niz. */
69     Student dosije[MAX_STUDENATA];
    FILE *fin = NULL;
71     int i;
    int br_studenata = 0;
73     long trazen_indeks = 0;
    char trazeno_prezime[MAX_DUZINA];
75
    /* Proveravamo da li nam je korisnik prilikom poziva prosledio
77     ime_datoteke sa informacijama o studentima */
    if (argc != 2) {
79         fprintf(stderr,
            "Greska: Program se poziva sa %s ime_datoteke\n",
81             argv[0]);
        exit(EXIT_FAILURE);
83     }

85     /* Otvaramo datoteku */
    fin = fopen(argv[1], "r");
87     if (fin == NULL) {
        fprintf(stderr,
89             "Neuspesno otvaranje datoteke %s za citanje\n",
            argv[1]);
91         exit(EXIT_FAILURE);
    }
93
    /* Citamo sve dok imamo red sa informacijama o studentu */
95     i = 0;
    while (1) {
97         if (i == MAX_STUDENATA)
            break;
99         if (fscanf
            (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
101             dosije[i].prezime) != 3)
            break;
103         i++;
    }
105     br_studenata = i;

107     /* Nakon citanja datoteka nam vise nije neophodna i odmah je
        zatvaramo */
109     fclose(fin);

111     printf("Unesite indeks studenta cije informacije zelite: ");
    scanf("%ld", &trazen_indeks);
113     i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
    if (i == -1)
115         printf("Ne postoji student sa indeksom %ld\n",

```

### 3 Algoritmi pretrage i sortiranja

```
        trazen_indeks);
117 else
    printf("Indeks: %ld, Ime i prezime: %s %s\n",
119         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

121 printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
123 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
    if (i == -1)
125         printf("Ne postoji student sa prezimenom %s\n",
                trazeno_prezime);
    else
127         printf("Indeks: %ld, Ime i prezime: %s %s\n",
129                 dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

131 return 0;
}
```

#### Rešenje 3.5

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

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

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

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

34 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
    tacaka t dimenzije n */
```

```

36 Tacka najbliza_x_osi(Tacka t[], int n)
37 {
38     Tacka najbliza;
39     int i;
40     najbliza = t[0];
41     for (i = 1; i < n; i++) {
42         if (fabs(t[i].x) < fabs(najbliza.x)) {
43             najbliza = t[i];
44         }
45     }
46     return najbliza;
47 }
48
49 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih
50    tacaka t dimenzije n */
51 Tacka najbliza_y_osi(Tacka t[], int n)
52 {
53     Tacka najbliza;
54     int i;
55     najbliza = t[0];
56     for (i = 1; i < n; i++) {
57         if (fabs(t[i].y) < fabs(najbliza.y)) {
58             najbliza = t[i];
59         }
60     }
61     return najbliza;
62 }
63
64 #define MAX 1024
65
66 int main(int argc, char *argv[])
67 {
68     FILE *ulaz;
69     Tacka tacke[MAX];
70     Tacka najbliza;
71     int i, n;
72
73     /* Ocekujemo da korisnik unese barem ime izvrsne verzije
74        programa i ime datoteke sa tackama */
75     if (argc < 2) {
76         fprintf(stderr,
77             "koriscenje programa: %s ime_datoteke\n", argv[0]);
78         return EXIT_FAILURE;
79     }
80
81     /* Otvaramo datoteku za citanje */
82     ulaz = fopen(argv[1], "r");
83     if (ulaz == NULL) {
84         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
85             argv[1]);
86         return EXIT_FAILURE;
87     }
88
89     /* Sve dok ima tacaka u datoteci, smestamo ih u niz sa
90        tackama; i predstavlja indeks tekuce tacke */

```

### 3 Algoritmi pretrage i sortiranja

```
92  i = 0;
    while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94      i++;
    }
96  n = i;

98  /* Proveravamo koji su dodatni argumenti komandne linije. Ako
    nema dodatnih argumenata */
100 if (argc == 2)
    /* Trazimo najblizu tacku u odnosu na koordinatni pocetak */
102  najbliza = najbliza_koordinatnom(tacke, n);
    /* Inace proveravamo koji je dodatni argument. Ako je u
104  pitanju opcija -x */
    else if (strcmp(argv[2], "-x") == 0)
106  /* Racunamo rastojanje u odnosu na x osu */
        najbliza = najbliza_x_osi(tacke, n);
108  /* Ako je u pitanju opcija -y */
    else if (strcmp(argv[2], "-y") == 0)
110  /* Racunamo rastojanje u odnosu na y osu */
        najbliza = najbliza_y_osi(tacke, n);
112  else {
    /* Ako nije zadata opcija -x ili -y, ispisujemo obavestenje
114  za korisnika i prekidamo izvršavanje programa */
        fprintf(stderr, "Pogresna opcija\n");
116  return EXIT_FAILURE;
    }
118
    /* Stampamo koordinate trazene tacke */
120  printf("%g %g\n", najbliza.x, najbliza.y);
122  fclose(ulaz);
124  return 0;
}
```

#### Rešenje 3.6

```
1  #include <stdio.h>
    #include <math.h>
3
    /* Tacnost */
5  #define EPS 0.001
7
    int main()
    {
9      double l, d, s;

11     /* Posto je u pitanju interval [0, 2] leva granica je 0, a
        desna 2 */
13     l = 0;
        d = 2;
15
        /* Sve dok ne pronadjemo trazenu vrednost argumenta */
17     while (1) {
        /* Pronalazimo sredinu intervala */
```



```

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

```

### Rešenje 3.12

```

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
34 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja
    u sortiranom nizu celih brojeva */
36 int najmanje_rastojanje(int a[], int n)
{
38     int i, min;
    min = a[1] - a[0];
40     for (i = 2; i < n; i++)
        if (a[i] - a[i - 1] < min)
42         min = a[i] - a[i - 1];
    return min;
44 }

46
48 int main()
{
    int i, a[MAX];

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

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

60     /* Ispis rezultata */
    printf("%d\n", najmanje_rastojanje(a, i));
62
    return 0;
64 }
```

#### Rešenje 3.13

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

4 #define MAX_DIM 128

6 /* Funkcija za sortiranje niza */
void selectionSort(char s[], int n)
8 {
    int i, j, min;
10     char pom;
    for (i = 0; i < n; i++) {
12         min = i;
        for (j = i + 1; j < n; j++)
14             if (s[j] < s[min])
                min = j;
16         if (min != i) {
            pom = s[i];
18             s[i] = s[min];
            s[min] = pom;
20         }
    }
```

```

    }
22 }

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

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

34     n = n_s;

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

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

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

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

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

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

```

### Rešenje 3.14

```

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

```

### 3 Algoritmi pretrage i sortiranja

---

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

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

42
int main()
44 {
    int a[MAX_DIM], i;
46
    /* Ucitavaju se element niza sve do kraja ulaza */
48     i = 0;
    printf("Unesite elemente niza: ");
50     while (scanf("%d", &a[i]) != EOF)
        i++;
52
    /* Niz se sortira */
54     selectionSort(a, i);

56     /* Odredjuje se broj koji se najvise puta pojavio u nizu */
    printf("%d\n", najvise_puta(a, i));
58
    return 0;
```

```
60 }
```

### Rešenje 3.15

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

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

20
22 /* Funkcija za binarnu pretragu niza. funkcija vraca: 1 - ako se
    element x nalazi u nizu; 0 - inace. prepostavlja se da je
    niz sortiran u rastucem poretku */
24 int binarna_pretraga(int a[], int n, int x)
{
26     int levi = 0, desni = n - 1, srednji;

28     while (levi <= desni) {
        srednji = (levi + desni) / 2;
30         if (a[srednji] == x)
            return 1;
32         else if (a[srednji] > x)
            desni = srednji - 1;
34         else if (a[srednji] < x)
            levi = srednji + 1;
36     }
    return 0;
38 }

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

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

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

### 3 Algoritmi pretrage i sortiranja

```
52     i++;
53     n = i;
54
55     selectionSort(a, n);
56
57     for (i = 0; i < n; i++)
58         /* Za i-ti element niza binarno se pretražuje da li se u
59            ostatku niza nalazi element koji sabran sa njim ima
60            ucitanu vrednost zbira */
61         if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
62             printf("da\n");
63             return 0;
64         }
65     printf("ne\n");
66     return 0;
67 }
```

#### Rešenje 3.16

```
1  #include <stdio.h>
2  #define MAX_DIM 256
3
4  int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
5            int dim3)
6  {
7      int i = 0, j = 0, k = 0;
8      /* U slucaju da je dimenzija treceg niza manja od neophodne,
9         funkcija vraca -1 */
10     if (dim3 < dim1 + dim2)
11         return -1;
12
13     /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja
14        jednog od njih */
15     while (i < dim1 && j < dim2) {
16         if (niz1[i] < niz2[j])
17             niz3[k++] = niz1[i++];
18         else
19             niz3[k++] = niz2[j++];
20     }
21     /* Ostatak prvog niza prepisujemo u treci */
22     while (i < dim1)
23         niz3[k++] = niz1[i++];
24
25     /* Ostatak drugog niza prepisujemo u treci */
26     while (j < dim2)
27         niz3[k++] = niz2[j++];
28     return dim1 + dim2;
29 }
30
31 int main()
32 {
33     int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
34     int i = 0, j = 0, k, dim3;
35
36     /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
```

```

38     Pretpostavka je da na ulazu neće biti više od MAX_DIM
        elemenata */
printf("Unesite elemente prvog niza: ");
40 while (1) {
    scanf("%d", &niz1[i]);
42     if (niz1[i] == 0)
        break;
44     i++;
}
46 printf("Unesite elemente drugog niza: ");
while (1) {
48     scanf("%d", &niz2[j]);
    if (niz2[j] == 0)
50         break;
    j++;
52 }

54 dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

56 /* Ispis */
for (k = 0; k < dim3; k++)
58     printf("%d ", niz3[k]);
printf("\n");
60
62 return 0;
}

```

### Rešenje 3.17

```

#include <stdio.h>
2  #include <stdlib.h>
#include <string.h>
4
/* Programu su neophodni argumenti komandne linije */
6 int main(int argc, char *argv[])
{
8     FILE *fin1 = NULL, *fin2 = NULL;
    FILE *fout = NULL;
10     char ime1[11], ime2[11];
    char prezime1[16], prezime2[16];
12     int kraj1 = 0, kraj2 = 0;

14     if (argc < 3) {
        fprintf(stderr,
16             "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
        exit(EXIT_FAILURE);
18     }

20     fin1 = fopen(argv[1], "r");
    if (fin1 == NULL) {
22         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n\n",
            argv[1]);
24         exit(EXIT_FAILURE);
    }
26

```

### 3 Algoritmi pretrage i sortiranja

```
28 fin2 = fopen(argv[2], "r");
    if (fin2 == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke %s\n\n",
30             argv[2]);
        exit(EXIT_FAILURE);
32     }

34     fout = fopen("ceo-tok.txt", "w");
    if (fout == NULL) {
36         fprintf(stderr,
            "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n\n
");
38         exit(EXIT_FAILURE);
    }

40     /* Citamo narednog studenta iz prve datoteke */
42     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
        kraj1 = 1;

44     /* Citamo narednog studenta iz druge datoteke */
46     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
        kraj2 = 1;

48     /* Sve dok nismo dosli do kraja neke datoteke */
50     while (!kraj1 && !kraj2) {
        if (strcmp(ime1, ime2) < 0) {
52             /* Ime i prezime iz prve datoteke je leksikografski
                ranije, upisujemo ga u izlaznu datoteku */
54             fprintf(fout, "%s %s\n", ime1, prezime1);
            /* Citamo narednog studenta iz prve datoteke */
56             if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
                kraj1 = 1;
58         } else {
            /* Ime i prezime iz druge datoteke je leksikografski
                ranije, upisujemo ga u izlaznu datoteku */
60             fprintf(fout, "%s %s\n", ime2, prezime2);
            /* Citamo narednog studenta iz druge datoteke */
62             if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
                kraj2 = 1;
64         }
    }

66     /* Ako smo iz prethodne petlje izašli zato što se doslo do
        kraja druge datoteke, onda ima još imena u prvoj datoteci,
        i prepisujemo ih, redom, jer su već sortirani po imenu. */
70     while (!kraj1) {
72         fprintf(fout, "%s %s\n", ime1, prezime1);
        if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
74             kraj1 = 1;
    }

76     /* Ako smo iz prve petlje izašli zato što se doslo do kraja
        prve datoteke, onda ima još imena u drugoj datoteci, i
        prepisujemo ih, redom, jer su već sortirani po imenu. */
78     while (!kraj2) {
80         fprintf(fout, "%s %s\n", ime2, prezime2);
```



```

82     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
83         kraj2 = 1;
84 }
85
86 /* Zatvaramo datoteke */
87 fclose(fin1);
88 fclose(fin2);
89 fclose(fout);
90
91 return 0;
92 }

```

### Rešenje 3.18

```

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

```

```

1  /* Datoteka sort.c */
2
3  #include "sort.h"
4
5  void selectionsort(int a[], int n)
6  {
7      int i, j;
8      int min;
9      int pom;
10
11      /* U svakoj iteraciji ove petlje se pronalazi najmanji element
12       medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
13       poziciju i, dok se element na poziciji i premesta na
14       poziciju min, na kojoj se nalazio najmanji od gore
15       navedenih elemenata. */
16      for (i = 0; i < n - 1; i++) {
17          /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
18           nalazi najmanji od elemenata a[i],...,a[n-1]. */
19          min = i;
20          for (j = i + 1; j < n; j++)
21              if (a[j] < a[min])
22                  min = j;

```

### 3 Algoritmi pretrage i sortiranja

---

```
24      /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
      samo ako su (i) i min razliciti, inace je nepotrebno. */
26      if (min != i) {
          pom = a[i];
28          a[i] = a[min];
          a[min] = pom;
30      }
    }
32 }
34
36 /* Funkcija sortira niz celih brojeva metodom sortiranja
   umetanjem. Ideja algoritma je sledeca: neka je na pocetku
38 i-te iteracije niz prvih i elemenata (a[0],a[1],...,a[i-1])
   sortirano. U i-toj iteraciji zelimo da element a[i] umetnemo
40 na pravu poziciju medju prvih i elemenata tako da dobijemo
   niz duzine i+1 koji je sortiran. Ovo radimo tako sto i-ti
42 element najpre uporedimo sa njegovim prvim levim susedom
   (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i
44 niz a[0],a[1],...,a[i] je sortiran, pa mozemo preci na
   sledecu iteraciju. Ako je a[i-1] vece, tada zamenjujemo a[i]
46 i a[i-1], a zatim proveravamo da li je potrebno dalje
   potiskivanje elementa u levo, poredeci ga sa njegovim novim
48 levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
   pravo mesto u nizu. */
50 void insertionsort(int a[], int n)
   {
52     int i, j;

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

58         /* U ovoj petlji redom potiskujemo element a[i] u levo
           koliko je potrebno, dok ne zauzme pravo mesto, tako da
           niz a[0],...,a[i] bude sortiran. Indeks j je trenutna
           pozicija na kojoj se element koji umecemo nalazi. Petlja
           se zavrшава ili kada element dodje do levog kraja (j==0)
           ili dok ne naidjemo na element a[j-1] koji je manji od
           a[j]. */
64         for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
            int temp = a[j];
66            a[j] = a[j - 1];
68            a[j - 1] = temp;
        }
70     }
   }
72
74 /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
   algoritma je sledeca: prolazimo kroz niz redom poredeci
76 susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
   poretku. Ovim se najveći element poput mehurica istiskuje na
78 "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je
```

```
potrebno ovaj postupak ponoviti nad nizom a[0],...,a[n-2],
80  tj. nad prvih n-1 elemenata niza bez poslednjeg koji je
    postavljen na pravu poziciju. Nakon toga se istu postupak
82  ponavlja nad sve kracim i kracim prefiksima niza, cime se
    jedan po jedan istiskuju elementi na svoje prave pozicije. */
84 void bubblesort(int a[], int n)
    {
86     int i, j;
88     int ind;

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

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

102                /* Promenljiva ind registruje da je bilo premestanja.
                    Samo u tom slučaju ima smisla ici na sledeću
                    iteraciju, jer ako nije bilo premestanja, znaci da su
104                    svi elementi već u dobrom poretku, pa nema potrebe
                        prelaziti na kraći prefiks niza. Moglo je naravno i
                        bez ovoga, algoritam bi radio ispravno, ali bi bio
106                        manje efikasan, jer bi često nepotrebno vrsio mnoga
                            upoređivanja, kada je već jasno da je sortiranje
                            završeno. */
108                ind = 1;
110            }
    }

112 /* Selsort je jednostavno proširenje sortiranja umetanjem koje
114 dopušta direktnu razmenu udaljenih elemenata. Proširenje se
    sastoji u tome da se kroz algoritam umetanja prolazi više
116 puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
    koji je manji od n (sto omogućuje razmenu udaljenih
118 elemenata) i tako se dobija h-sortiran niz, tj. niz u kome su
    elementi na rastojanju h sortirani, mada susedni elementi to
120 ne moraju biti. U drugom prolazu kroz isti algoritam sprovodi
    se isti postupak ali za manji korak h. Sa prolazima se
122 nastavlja sve do koraka h = 1, u kome se dobija potpuno
    sortirani niz. Izbor početne vrednosti za h, i načina
124 njegovog smanjivanja menja u nekim slučajevima brzinu
    algoritma, ali bilo koja vrednost će rezultovati ispravnim
126 sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
    vrednost 1. */
128 void shellsort(int a[], int n)
    {
130     int h = n / 2, i, j;
132     while (h > 0) {
        /* Insertion sort sa korakom h */
        for (i = h; i < n; i++) {
134             int temp = a[i];
```

```

    j = i;
136     while (j >= h && a[j - h] > temp) {
        a[j] = a[j - h];
138         j -= h;
    }
140     a[j] = temp;
    }
142     h = h / 2;
    }
144 }

146 #define MAX 1000000

148 /* Sortiranje ucesljavanjem */
void mergesort(int a[], int l, int d)
150 {
    int s;
152     static int b[MAX];          /* Pomocni niz */
    int i, j, k;

154     /* Izlaz iz rekurzije */
156     if (l >= d)
        return;

158     /* Odredjujemo sredisnji indeks */
160     s = (l + d) / 2;

162     /* Rekurzivni pozivi */
    mergesort(a, l, s);
164     mergesort(a, s + 1, d);

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

172     /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
174     while (i <= s && j <= d) {
        if (a[i] < a[j])
176             b[k++] = a[i++];
        else
178             b[k++] = a[j++];
    }

180     while (i <= s)
182         b[k++] = a[i++];

184     while (j <= d)
        b[k++] = a[j++];

186     /* Prepisujemo "ucesljani" niz u originalni niz */
188     for (k = 0, i = l; i <= d; i++, k++)
        a[i] = b[k];
190 }
```

```

192 }
193
194 /* Funkcija menja mesto i-tom i j-tom elementu niza a */
194 void swap(int a[], int i, int j)
195 {
196     int tmp = a[i];
197     a[i] = a[j];
198     a[j] = tmp;
199 }
200
201
202 /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r */
202 void quicksort(int a[], int l, int r)
203 {
204     int i, pivot_position;
205
206     /* Izlaz iz rekurzije -- prazan niz */
207     if (l >= r)
208         return;
209
210
211     /* Particionisanje niza. Svi elementi na pozicijama <=
212        pivot_position (izuzev same pozicije l) su strogo manji od
213        pivota. Kada se pronadje neki element manji od pivota,
214        uvecava se pivot_position i na tu poziciju se premesta
215        nadjeni element. Na kraju ce pivot_position zaista biti
216        pozicija na koju treba smestiti pivot, jer ce svi elementi
217        levo od te pozicije biti manji a desno biti veci ili
218        jednaki od pivota. */
219     pivot_position = l;
220     for (i = l + 1; i <= r; i++)
221         if (a[i] < a[l])
222             swap(a, ++pivot_position, i);
223
224     /* Postavljamo pivot na svoje mesto */
225     swap(a, l, pivot_position);
226
227     /* Rekurzivno sortiramo elemente manje od pivota */
228     quicksort(a, l, pivot_position - 1);
229     /* Rekurzivno sortiramo elemente vece pivota */
230     quicksort(a, pivot_position + 1, r);
231 }
232

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "sort.h"
5
6 /* Maksimalna duzina niza */
7 #define MAX 1000000
8
9 int main(int argc, char *argv[])
10 {
11     /*
12        tip_sortiranja == 0 => selectionsort (default)
13        tip_sortiranja == 1 => insertionsort (-i opcija komandne

```

### 3 Algoritmi pretrage i sortiranja

```
14     linije) tip_sortiranja == 2 => bubblesort (-b opcija
komandne linije) tip_sortiranja == 3 => shellsort (-s
16     opcija komandne linije) tip_sortiranja == 4 => mergesort
(-m opcija komandne linije) tip_sortiranja == 5 =>
18     quicksort (-q opcija komandne linije) */
int tip_sortiranja = 0;
20 /*
    tip_niza == 0 => slucajno generisani nizovi (default)
22     tip_niza == 1 => sortirani nizovi (-r opcija komandne
linije) tip_niza == 2 => obrnuto soritrani nizovi (-o
24     opcija komandne linije) */
int tip_niza = 0;
26
/* Dimenzija niza koji se sortira */
28 int dimenzija;
int i;
30 int niz[MAX];
32
if (argc < 2) {
    fprintf(stderr,
34         "Program zahteva bar 2 argumenta komandne linije!\n");
    exit(EXIT_FAILURE);
36 }

38 /* Ocitavamo opcije i argumente prilikom poziva programa */
for (i = 1; i < argc; i++) {
40     /* Ako je u pitanju opcija... */
    if (argv[i][0] == '-') {
42         switch (argv[i][1]) {
            case 'i':
44             tip_sortiranja = 1;
            break;
46             case 'b':
                tip_sortiranja = 2;
48             break;
            case 's':
50             tip_sortiranja = 3;
            break;
52             case 'm':
                tip_sortiranja = 4;
54             break;
            case 'q':
56             tip_sortiranja = 5;
            break;
58             case 'r':
                tip_niza = 1;
60             break;
            case 'o':
62             tip_niza = 2;
            break;
64             default:
                printf("Pogresna opcija -%c\n", argv[i][1]);
66             return 1;
            break;
68         }
    }
}
```

```

70  /* Ako je u pitanju argument, onda je to duzina niza koji
    treba da se sortira */
72  else {
    dimenzija = atoi(argv[i]);
74    if (dimenzija <= 0 || dimenzija > MAX) {
        fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
76        exit(EXIT_FAILURE);
    }
78  }
    }
80
82  /* Elemente niza odredjujemo slucajno, ali vodeci racuna o
    tipu niza dobijenom iz komandni linije. srandom funkcija
    obezbedjuje novi seed za pozivanje random funkcije, i kako
84    nas niz ne bi uvek isto izgledao seed smo postavili na
    tekuce vreme u sekundama od Nove godine 1970. random()%100
86    daje brojeve izmedju 0 i 99 */
    srandom(time(NULL));
88    if (tip_niza == 0)
        for (i = 0; i < dimenzija; i++)
90        niz[i] = random();
    else if (tip_niza == 1)
92        for (i = 0; i < dimenzija; i++)
            niz[i] =
94            i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
    else
96        for (i = 0; i < dimenzija; i++)
            niz[i] =
98            i == 0 ? random() % 100 : niz[i - 1] - random() % 100;

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

104
106 /* Sortiramo niz na odgovarajuci nacin */
    if (tip_sortiranja == 0)
        selectionsort(niz, dimenzija);
108    else if (tip_sortiranja == 1)
        insertionsort(niz, dimenzija);
110    else if (tip_sortiranja == 2)
        bubblesort(niz, dimenzija);
112    else if (tip_sortiranja == 3)
        shellsort(niz, dimenzija);
114    else if (tip_sortiranja == 4)
        mergesort(niz, 0, dimenzija - 1);
116    else
        quicksort(niz, 0, dimenzija - 1);

118
120 /* Ispisujemo elemente niza */
    /******
    printf("Sortiran niz je:\n");
122    for (i = 0; i < dimenzija; i++)
        printf("%d\n", niz[i]);
124    *****/

```

```
126     return 0;
    }
```

#### Rešenje 3.19

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  #define MAX_BR_TACAKA 128
7
8  typedef struct Tacka {
9      int x;
10     int y;
11 } Tacka;
12
13 /* Funkcija racuna rastojanje zadate tacke od koordinatnog
14    pocetka (0,0) */
15 float rastojanje(Tacka A)
16 {
17     return sqrt(A.x * A.x + A.y * A.y);
18 }
19
20 /* Funkcija koja sortira niz tacaka po rastojanju od
21    koordinatnog pocetka */
22 void sortiraj_po_rastojanju(Tacka t[], int n)
23 {
24     int min, i, j;
25     Tacka tmp;
26
27     for (i = 0; i < n - 1; i++) {
28         min = i;
29
30         for (j = i + 1; j < n; j++) {
31             if (rastojanje(t[j]) < rastojanje(t[min])) {
32                 min = j;
33             }
34         }
35
36         if (min != i) {
37             tmp = t[i];
38             t[i] = t[min];
39             t[min] = tmp;
40         }
41     }
42 }
43
44 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
45 void sortiraj_po_x(Tacka t[], int n)
46 {
47     int min, i, j;
```



```
51  Tacka tmp;
53  for (i = 0; i < n - 1; i++) {
55      min = i;
57      for (j = i + 1; j < n; j++) {
59          if (abs(t[j].x) < abs(t[min].x)) {
61              min = j;
63          }
65      }
67      if (min != i) {
69          tmp = t[i];
71          t[i] = t[min];
73          t[min] = tmp;
75      }
77  }
79
91  /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
93  void sortiraj_po_y(Tacka t[], int n)
95  {
97      int min, i, j;
99      Tacka tmp;
101      for (i = 0; i < n - 1; i++) {
103          min = i;
105          for (j = i + 1; j < n; j++) {
107              if (abs(t[j].y) < abs(t[min].y)) {
109                  min = j;
111              }
113          }
115          if (min != i) {
117              tmp = t[i];
119              t[i] = t[min];
121              t[min] = tmp;
123          }
125      }
127  }
129
131  int main(int argc, char *argv[])
133  {
135      FILE *ulaz;
137      FILE *izlaz;
139      Tacka tacke[MAX_BR_TACAKA];
141      int i, n;
143
145      /* Proveravamo broj argumenata komandne linije: ocekujemo ime
147         izvrsnog programa, opciju, ime ulazne datoteke i ime
149         izlazne datoteke tj. ocekujemo 4 argumenta */
```

```
107 if (argc != 4) {
109     fprintf(stderr,
111         "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
113     return 0;
115 }
117 /* Otvaramo datoteku u kojoj su zadate tacke */
119 ulaz = fopen(argv[2], "r");
121 if (ulaz == NULL) {
123     fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
125         argv[2]);
127     return 0;
129 }
131 /* Otvaramo datoteku u koju treba upisati rezultat */
133 izlaz = fopen(argv[3], "w");
135 if (izlaz == NULL) {
137     fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
139         argv[3]);
141     return 0;
143 }
145 /* Sve dok ne stignemo do kraja ulazne datoteke ucitavamo
147 koordinatu tacaka i smestamo ih na odgovarajucu poziciju
149 odredjenu brojacem i; prilikom ucitavanja oslanjamo se na
151 svojstvo funkcije fscanf povratka EOF vrednosti kada stigne
153 do kraja ulaza */
155 i = 0;
157 while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
159     i++;
161 }
163 /* Cuvamo broj procitanih tacaka */
165 n = i;
167 /* Analiziramo zadatu opciju: kako ocekujemo da je argv[1]
169 "-x" ili "-y" ili "-o" sigurni smo da je argv[1][0] crtica
171 (karakter -) i dalje proveravamo sta je na sledecoj
173 poziciji tj. sta je argv[1][1] */
175 switch (argv[1][1]) {
177 case 'x':
179     /* Ako je u pitanju karakter x, pozivamo funkciju za
181 sortiranje po vrednosti x koordinate */
183     sortiraj_po_x(tacke, n);
185     break;
187 case 'y':
189     /* Ako je u pitanju karakter y, pozivamo funkciju za
191 sortiranje po vrednosti y koordinate */
193     sortiraj_po_y(tacke, n);
195     break;
197 case 'o':
199     /* Ako je u pitanju karakter o, pozivamo funkciju za
```

```

163     sortiranje po udaljenosti od koordinatnog pocetka */
164     sortiraj_po_rastojanju(tacke, n);
165 }
166
167 /* Upisujemo dobijeni niz u izlaznu datoteku */
168 for (i = 0; i < n; i++) {
169     fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
170 }
171
172 /* Zatvaramo otvorene datoteke */
173 fclose(ulaz);
174 fclose(izlaz);
175
176 /* I završavamo sa programom */
177 return 0;
178 }

```

### Rešenje 3.20

```

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

#define MAX 1000
#define MAX_DUZINA 16

typedef struct gr {
    char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
} Gradjanin;

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

    for (i = 0; i < n - 1; i++) {
        /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
           nalazi najmanji od elemenata a[i].ime,...,a[n-1].ime. */
        min = i;
        for (j = i + 1; j < n; j++)
            if (strcmp(a[j].ime, a[min].ime) < 0)
                min = j;

        /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
           samo ako su (i) i min razliciti, inace je nepotrebno. */
        if (min != i) {
            pom = a[i];
            a[i] = a[min];
            a[min] = pom;
        }
    }
}

```

### 3 Algoritmi pretrage i sortiranja

---

```
38 /* Fja sortira niz gradjana rastuce po prezimenima */
void sort_prezime(Gradjanin a[], int n)
40 {
42     int i, j;
43     int min;
44     Gradjanin pom;
46     for (i = 0; i < n - 1; i++) {
47         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
48            nalazi najmanji od elemenata
49            a[i].prezime,...,a[n-1].prezime. */
50         min = i;
51         for (j = i + 1; j < n; j++)
52             if (strcmp(a[j].prezime, a[min].prezime) < 0)
53                 min = j;
54
55         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
56            samo ako su (i) i min razliciti, inace je nepotrebno. */
57         if (min != i) {
58             pom = a[i];
59             a[i] = a[min];
60             a[min] = pom;
61         }
62     }
63 }
64
65 /* Pretraga niza Gradjana */
66 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
67 {
68     int i;
69     for (i = 0; i < n; i++)
70         if (strcmp(a[i].ime, x->ime) == 0
71             && strcmp(a[i].prezime, x->prezime) == 0)
72             return i;
73     return -1;
74 }
75
76 int main()
77 {
78     Gradjanin spisak1[MAX], spisak2[MAX];
79     int isti_rbr = 0;
80     int i, n;
81     FILE *fp = NULL;
82
83     /* Otvaranje datoteke */
84     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
85         fprintf(stderr,
86             "Neuspesno otvaranje datoteke biracki-spisak.txt.\n");
87         exit(EXIT_FAILURE);
88     }
89
90     /* Citanje sadrzaja */
91     for (i = 0;
```

```

194         fscanf(fp, "%s %s", spisak1[i].ime,
                spisak1[i].prezime) != EOF; i++)
        spisak2[i] = spisak1[i];
196     n = i;

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

100     sort_ime(spisak1, n);

102     /******
104     printf("Biracki spisak [uredjen prema imenima]:\n");
        for(i=0; i<n; i++)
106         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
        ******/

108     sort_prezime(spisak2, n);

110     /******
112     printf("Biracki spisak [uredjen prema prezimenima]:\n");
        for(i=0; i<n; i++)
114         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
        ******/

116     /* Linearno pretrazivanje nizova */
118     for (i = 0; i < n; i++)
        if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
120         isti_rbr++;

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

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

```

### Rešenje 3.22

```

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

4
#define MAX_BR_RECII 128
6  #define MAX_DUZINA_RECII 32

8
/* Funkcija koja izracunava broj suglasnika u reci */
10 int broj_suglasnika(char s[])

```

```
{
12  char c;
13  int i;
14  int suglasnici = 0;
15  /* Obilazimo karakter po karakter zadate niske */
16  for (i = 0; s[i]; i++) {
17      /* Ako je u pitanju slovo */
18      if (isalpha(s[i])) {
19          /* Pretvaramo ga u veliko da bismo mogli da pokrijemo
20             slucaj i malih i velikih suglasnika */
21          c = toupper(s[i]);
22          /* Ukoliko slovo nije samoglasnik */
23          if (c != 'A' && c != 'E' && c != 'I' && c != 'O'
24              && c != 'U') {
25              /* Uvecavamo broj suglasnika */
26              suglasnici++;
27          }
28      }
29  }
30  /* Vracamo izracunatu vrednost */
31  return suglasnici;
32 }

33 /* Funkcija koja sortira reci po zadatom kriterijumu - OPREZ:
34    informacija o duzini reci se mora proslediti zbog pravilnog
35    upravljanja memorijom */
36 void sortiraj_reci(char reci[][MAX_DUZINA_RECI], int n)
37 {
38     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
39         duzina_j, duzina_min;
40     char tmp[MAX_DUZINA_RECI];
41     for (i = 0; i < n - 1; i++) {
42         min = i;
43         for (j = i; j < n; j++) {
44             /* Prvo uporedjujemo broj suglasnika */
45             broj_suglasnika_j = broj_suglasnika(reci[j]);
46             broj_suglasnika_min = broj_suglasnika(reci[min]);
47             if (broj_suglasnika_j < broj_suglasnika_min)
48                 min = j;
49             else if (broj_suglasnika_j == broj_suglasnika_min) {
50                 /* Zatim, reci imaju isti broj suglasnika uporedjujemo
51                    duzine */
52                 duzina_j = strlen(reci[j]);
53                 duzina_min = strlen(reci[min]);
54
55                 if (duzina_j < duzina_min)
56                     min = j;
57                 else
58                     /* A ako reci imaju i isti broj suglasnika i iste
59                        duzine, uporedjujemo ih leksikografski */
60                     if (duzina_j == duzina_min
61                         && strcmp(reci[j], reci[min]) < 0)
62                         min = j;
63             }
64         }
65     }
66     if (min != i) {
```

```
        strcpy(tmp, reci[min]);
68        strcpy(reci[min], reci[i]);
        strcpy(reci[i], tmp);
70    }
    }
72 }

74 int main()
{
76     FILE *ulaz;
78     int i = 0, n;

80     /* Niz u kojem ce biti smestane reci. Prvi broj oznacava broj
        reci, a drugi maksimalnu duzinu pojedinačne reci */
82     char reci[MAX_BR_RECII][MAX_DUZINA_RECII];

84     /* Otvaramo datoteku niske.txt za citanje */
    ulaz = fopen("niske.txt", "r");
86     if (ulaz == NULL) {
        fprintf(stderr,
88             "Greska prilikom otvaranja datoteke niske.txt!\n");
        return 0;
90     }

92     /* Sve dok mozemo da procitamo sledecu rec */
    while (fscanf(ulaz, "%s", reci[i]) != EOF) {
94        /* Proveravamo da li smo ucitali najvise dozvoljenih reci i
        ako jesmo, prekidamo ucitavanje */
96        if (i == MAX_BR_RECII)
            break;
98        /* Pripremamo brojac za narednu iteraciju */
        i++;
100    }

102    /* n je duzina naseg niza reci i predstavlja poslednju
        vrednost koriscenog brojaca */
104    n = i;
    /* Pozivamo funkciju za sortiranje reci - OPREZ: nacin
106    prosledjivanja niza reci */
    sortiraj_reci(reci, n);

108    /* Ispisujemo sortirani niz reci */
110    for (i = 0; i < n; i++) {
        printf("%s ", reci[i]);
112    }
    printf("\n");

114    /* Zatvaramo datoteku */
116    fclose(ulaz);

118    /* I zavrsavamo sa programom */
    return 0;
120 }
```

#### Rešenje 3.23

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_ARTIKALA 100000
6
7  typedef struct art {
8      long kod;
9      char naziv[20];
10     char proizvodjac[20];
11     float cena;
12 } Artikal;
13
14 int binarna_pretraga(Artikal a[], int n, long x)
15 {
16     int levi = 0;
17     int desni = n - 1;
18
19     /* Dokle god je indeks levi levo od indeksa desni */
20     while (levi <= desni) {
21         /* Racunamo sredisnji indeks */
22         int srednji = (levi + desni) / 2;
23         /* Ako je sredisnji element veci od x, tada se x mora
24            nalaziti u levoj polovini niza */
25         if (x < a[srednji].kod)
26             desni = srednji - 1;
27         /* Ako je sredisnji element manji od x, tada se x mora
28            nalaziti u desnoj polovini niza */
29         else if (x > a[srednji].kod)
30             levi = srednji + 1;
31         else
32             /* Ako je sredisnji element jednak x, tada smo pronasli x
33                na poziciji srednji */
34             return srednji;
35     }
36     /* Ako nije pronadjen vracamo -1 */
37     return -1;
38 }
39
40 void selection_sort(Artikal a[], int n)
41 {
42     int i, j;
43     int min;
44     Artikal pom;
45
46     for (i = 0; i < n - 1; i++) {
47         min = i;
48         for (j = i + 1; j < n; j++)
49             /* Sortiranje vrsimo po kodovima i onda i poredjenje
50                vrsimo nad kodovima */
51             if (a[j].kod < a[min].kod)
52                 min = j;
53
54         if (min != i) {
```



```

55     pom = a[i];
56     a[i] = a[min];
57     a[min] = pom;
58 }
59 }
60 }
61
62 int main()
63 {
64     Artikl asortiman[MAX_ARTIKALA];
65     long kod;
66     int i, n;
67     float racun;
68
69     FILE *fp = NULL;
70
71     /* Otvaranje datoteke */
72     if ((fp = fopen("artikli.txt", "r")) == NULL) {
73         fprintf(stderr,
74             "Neuspesno otvaranje datoteke artikli.txt.\n");
75         exit(EXIT_FAILURE);
76     }
77
78     /* Ucitavanje artikala */
79     i = 0;
80     while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
81         asortiman[i].naziv, asortiman[i].proizvodjac,
82         &asortiman[i].cena) == 4)
83         i++;
84
85     /* Zatvaranje datoteke */
86     fclose(fp);
87
88     n = i;
89
90     /* Sortiracemo celokupan asortiman prodavnice prema kodovima
91     jer ce pri kucanju racuna prodavac unositi kod artikla.
92     Prilikom kucanja svakog racuna pretrazuje se asortiman, da
93     bi se utvrdila cena artikla. Kucanje racuna obuhvata vise
94     pretraga asortimana i u interesu nam je da ta operacija
95     bude sto efikasnija. Zelimo da koristimo algoritam binarne
96     pretrage priliko pretrazivanje po kodu artikla. Iz tog
97     razloga, potrebno je da nam asortiman bude sortiran po
98     kodovima i to cemo uraditi primenom selection sort
99     algoritma. Sortiramo samo jednom na pocetku, ali zato posle
100    brzo mozemo da pretrazujemo prilikom kucanja proizvoljno
101    puno racuna. Vreme koje se utrosi na sortiranje na pocetku
102    izvorsavanja programa, kasnije se isplati jer za brojna
103    trazanja artikla mozemo umesto linearne da koristimo
104    efikasniju binarnu pretragu. */
105
106    selection_sort(asortiman, n);
107
108    printf
109    ("Asortiman:\nKOD          Naziv artikla      Ime
110     proizvodjaca      Cena\n");

```

### 3 Algoritmi pretrage i sortiranja

```
111 for (i = 0; i < n; i++)
    printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113         asortiman[i].naziv, asortiman[i].proizvodjac,
        asortiman[i].cena);

115 kod = 0;
while (1) {
117     printf("-----\n");
    printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
119     printf("- Za nov racun unesite kod artikla!\n\n");
    if (scanf("%ld", &kod) == EOF)
121         break;

123     racun = 0;
    while (1) {
125         if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
            printf
127                 ("\tGRESKA: Ne postoji proizvod sa trazanim kodom!\n");
        } else {
129             printf("\tTrazili ste:\t%s %s %12.2f\n",
                asortiman[i].naziv, asortiman[i].proizvodjac,
131                 asortiman[i].cena);

            racun += asortiman[i].cena;
133         }

135         printf("Unesite kod artikla [ili 0 za prekid]: \t");
137         scanf("%ld", &kod);

139         if (kod == 0)
            break;
141     }

143     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
    }

145     printf("Kraj rada kase!\n");
147     exit(EXIT_SUCCESS);
149 }
```

#### Rešenje 3.24

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX 500
6
/* Struktura koja nam je neophodna za sve informacije o
8   pojedinacnom studentu */
typedef struct {
10     char ime[20];
    char prezime[25];
12     int prisustvo;
```

```
    int zadaci;
14 } Student;

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

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

36         if (min != i) {
38             pom = niz[min];
40             niz[min] = niz[i];
42             niz[i] = pom;
44         }
46     }
48 }

50 /* Funkcija kojom sortiramo niz struktura po ukupnom broju
    uradjenih zadataka opadajuće, ukoliko neki studenti imaju
    isti broj uradjenih zadataka sortiraju se po dužini imena
    rastuce. */
52 void sort_zadatke_pa_imena(Student niz[], int n)
    {
54     int i, j;
56     int max;
58     Student pom;

60     for (i = 0; i < n - 1; i++) {
62         max = i;
64         for (j = i + 1; j < n; j++)
66             if (niz[j].zadaci > niz[max].zadaci)
68                 max = j;
69             else if (niz[j].zadaci == niz[max].zadaci
70                     && strlen(niz[j].ime) < strlen(niz[max].ime))
71                 max = j;
72         if (max != i) {
74             pom = niz[max];
76             niz[max] = niz[i];
78             niz[i] = pom;
80         }
82     }
84 }

86 /* Funkcija kojom sortiramo niz struktura po broju casova na
    kojima su bili opadajuće, a ukoliko * neki studenti imaju
    isti broj casova, sortiraju se opadajuće po broju uradjenih
    zadataka, * a ukoliko se i po broju zadataka poklapaju
    sortirati ih po prezimenu opadajuće. */
88 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
```

```
{
70  int i, j;
71  int max;
72  Student pom;
73  for (i = 0; i < n - 1; i++) {
74      max = i;
75      for (j = i + 1; j < n; j++)
76          if (niz[j].prisustvo > niz[max].prisustvo)
77              max = j;
78          else if (niz[j].prisustvo == niz[max].prisustvo
79                  && niz[j].zadaci > niz[max].zadaci)
80              max = j;
81          else if (niz[j].prisustvo == niz[max].prisustvo
82                  && niz[j].zadaci == niz[max].zadaci
83                  && strcmp(niz[j].prezime, niz[max].prezime) > 0)
84              max = j;
85      if (max != i) {
86          pom = niz[max];
87          niz[max] = niz[i];
88          niz[i] = pom;
89      }
90  }
91 }

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

```

126     }
127     fprintf
128         (fp,
129          "Studenti sortirani po imenu leksikografski rastuce:\n");
130     for (i = 0; i < br_studenata; i++)
131         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
132                praktikum[i].prezime, praktikum[i].prisustvo,
133                praktikum[i].zadaci);
134     fclose(fp);
135
136     /* Na drugom su sortirani po ukupnom broju uradjenih zadataka
137        opadajuće, ukoliko neki studenti imaju isti broj uradjenih
138        zadataka sortiraju se po dužini imena rastuce. */
139     sort_zadatke_pa_imena(praktikum, br_studenata);
140     /* Otvaranje datoteke za pisanje */
141     if ((fp = fopen("dat2.txt", "w")) == NULL) {
142         fprintf(stderr, "Neuspješno otvaranje datoteke dat2.txt.\n");
143         exit(EXIT_FAILURE);
144     }
145     fprintf(fp,
146            "Studenti sortirani po broju zadataka opadajuće,\n");
147     fprintf(fp, "pa po dužini imena rastuce:\n");
148     for (i = 0; i < br_studenata; i++)
149         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
150                praktikum[i].prezime, praktikum[i].prisustvo,
151                praktikum[i].zadaci);
152     fclose(fp);
153
154     /* Na trećem spisku su sortirani po broju časova na kojima su
155        bili opadajuće, a ukoliko neki studenti imaju isti broj
156        časova, sortiraju se opadajuće po broju uradjenih zadataka,
157        a ukoliko se i po broju zadataka poklapaju sortirati ih po
158        prezimenu opadajuće. */
159     sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum,
160                                              br_studenata);
161     /* Otvaranje datoteke za pisanje */
162     if ((fp = fopen("dat3.txt", "w")) == NULL) {
163         fprintf(stderr, "Neuspješno otvaranje datoteke dat3.txt.\n");
164         exit(EXIT_FAILURE);
165     }
166     fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
167     fprintf(fp, "pa po broju zadataka,\n");
168     fprintf(fp, "pa po prezimenima leksikografski opadajuće,\n");
169     for (i = 0; i < br_studenata; i++)
170         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
171                praktikum[i].prezime, praktikum[i].prisustvo,
172                praktikum[i].zadaci);
173     fclose(fp);
174     return 0;
175 }

```

### Rešenje 3.25

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

### 3 Algoritmi pretrage i sortiranja

```
2 #include <stdlib.h>
3 #include <string.h>
4 #define KORAK 10
5
6 /* Struktura koja opisuje jednu pesmu */
7 typedef struct {
8     char *izvodjac;
9     char *naslov;
10    int broj_gledanja;
11 } Pesma;
12
13 /* Funkcija za uporedjivanje pesama po broju gledanosti
14    (potrebna za rad qsort funkcije) */
15 int uporedi_gledanost(const void *pp1, const void *pp2)
16 {
17     Pesma *p1 = (Pesma *) pp1;
18     Pesma *p2 = (Pesma *) pp2;
19
20     return p2->broj_gledanja - p1->broj_gledanja;
21 }
22
23 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad
24    qsort funkcije) */
25 int uporedi_naslove(const void *pp1, const void *pp2)
26 {
27     Pesma *p1 = (Pesma *) pp1;
28     Pesma *p2 = (Pesma *) pp2;
29
30     return strcmp(p1->naslov, p2->naslov);
31 }
32
33 /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za
34    rad qsort funkcije) */
35 int uporedi_izvodjace(const void *pp1, const void *pp2)
36 {
37     Pesma *p1 = (Pesma *) pp1;
38     Pesma *p2 = (Pesma *) pp2;
39
40     return strcmp(p1->izvodjac, p2->izvodjac);
41 }
42
43 int main(int argc, char *argv[])
44 {
45     FILE *ulaz;
46     Pesma *pesme;
47     /* Pokazivac na deo memorije za
48        cuvanje pesama */
49     int alocirano_za_pesme;
50     /* Broj mesta alociranih za
51        pesme */
52     int i;
53     /* Redni broj pesme cije se
54        informacije citaju */
55     int n;
56     /* Ukupan broj pesama */
57     int j, k;
58     char c;
59     int alocirano;
60     /* Broj mesta alociranih za
61        propratne informacije o
```

```

58                                     pesmama */
int broj_gledanja;
60
/* Pripremamo datoteku za citanje */
62 ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
if (ulaz == NULL) {
64     printf
        ("Problem sa citanjem datoteke pesme_bez_pretpostavki.txt!\n
");
66     return 0;
}
68
/* Citamo informacije o pesmama */
70 pesme = NULL;
alocirano_za_pesme = 0;
72 i = 0;
74 while (1) {
76     /* Proveravamo da li smo stigli do kraja datoteke */
    c = fgetc(ulaz);
78     if (c == EOF) {
        /* Ako smo dobili kao rezultat EOF, jesmo, nema vise
80         sadrzaja za citanje */
        break;
82     } else {
        /* Ako nismo, vracamo procitani karakter nazad */
84         ungetc(c, ulaz);
    }
86
88     /* Proveravamo da li imamo dovoljno memorije za citanje nove
    pesme */
90     if (alocirano_za_pesme == i) {
92         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
        alociramo novih KORAK mesta */
94         alocirano_za_pesme += KORAK;
        pesme =
96             (Pesma *) realloc(pesme,
                                alocirano_za_pesme * sizeof(Pesma));
98
        /* Proveravamo da li je nova memorija uspesno realocirana */
100        if (pesme == NULL) {
            /* Ako nije ... */
102            /* Ispisujemo obavestenje */
            printf("Problem sa alokacijom memorije!\n");
104
            /* I oslobadjamo svu memoriju zauzetu do ovog koraka */
106            for (k = 0; k < i; k++) {
                free(pesme[k].izvodjac);
108                free(pesme[k].naslov);
            }
            free(pesme);
            return 0;
110        }
112    }

```

```
114     }
116     /* Ako jeste, nastavljamo sa citanjem pesama ... */
117     /* Citamo ime izvodjaca */
118
119     j = 0;                                /* Oznacava poziciju na koju
120                                           treba smestiti procitani
121                                           karakter */
122     alocirano = 0;                        /* Oznacava broj alociranih
123                                           mesta */
124     pesme[i].izvodjac = NULL;            /* Memorija koju mozemo
125                                           koristiti za smestanje
126                                           procitanih karaktera */
127
128     /* Sve dok ne stignemo do prve beline u liniji - beline koja
129        se nalazi nakon imena izvodjaca - citamo karaktere iz
130        datoteke */
131     while ((c = fgetc(ulaz)) != ' ') {
132
133         /* Proveravamo da li imamo dovoljno memorije za smestanje
134            procitanog karaktera */
135         if (j == alocirano) {
136
137             /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
138                alociramo novih KORAK mesta */
139             alocirano += KORAK;
140             pesme[i].izvodjac =
141                 (char *) realloc(pesme[i].izvodjac,
142                                 alocirano * sizeof(char));
143
144             /* Proveravamo da li je nova alokacija uspesna */
145             if (pesme[i].izvodjac == NULL) {
146                 /* Ako nije... */
147                 /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
148                 for (k = 0; k < i; k++) {
149                     free(pesme[k].izvodjac);
150                     free(pesme[k].naslov);
151                 }
152                 free(pesme);
153                 /* I prekidamo sa izvorsavanjem programa */
154                 return 0;
155             }
156         }
157
158         /* Ako imamo dovoljno memorije, smestamo procitani
159            karakter */
160         pesme[i].izvodjac[j] = c;
161         j++;
162         /* I nastavljamo sa citanjem */
163     }
164
165     /* Upisujemo terminirajucu nulu na kraju reci */
166     pesme[i].izvodjac[j] = '\0';
167
168 }
```



```
170  /* Citamo - */
    fgetc(ulaz);

172  /* Citamo razmak */
    fgetc(ulaz);

174

176  /* Citamo naslov pesme */
    j = 0;                                /* Oznacava poziciju na koju
                                           treba smestiti procitani
                                           karakter */
178
    alocirano = 0;                        /* Oznacava broj alociranih
                                           mesta */
180
    pesme[i].naslov = NULL;              /* Memorija koju mozemo
                                           koristiti za smestanje
                                           procitanih karaktera */
182
184

186  /* Sve dok ne stignemo do zareza - zareza koji se nalazi
    nakon naslova pesme - citamo karaktere iz datoteke */
188

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

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

210                /* I prekidamo izvršavanje programa */
212                return 0;
            }
214        }

216        /* Ako imamo dovoljno memorije, smestamo procitani
           karakter */
218        pesme[i].naslov[j] = c;
        j++;
220        /* I nastavljamo dalje sa citanjem */
    }
222    /* Upisujemo terminirajucu nulu na kraju reci */
    pesme[i].naslov[j] = '\0';
224
```

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

228
    /* Citamo broj gledanja */
230
    broj_gledanja = 0;
232
    /* Sve dok ne stignemo do znaka za novi red - kraja linije -
       citamo karaktere iz datoteke */
234     while ((c = fgetc(ulaz)) != '\n') {
236         broj_gledanja = broj_gledanja * 10 + (c - '0');
    }
238     pesme[i].broj_gledanja = broj_gledanja;

240     /* Prelazimo na citanje sledece pesme */
    i++;
242
    }

244     /* Cuvamo informaciju o broju pesama koje smo procitali */
246     n = i;

248     /* Zatvaramo datoteku jer nam nece vise trebati */
    fclose(ulaz);

250
    /* Analiziramo argumente komandne linije */
252     if (argc == 1) {

254         /* Nema dodatnih opcija - sortiramo po broju gledanja */
        qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
256     } else {

258         if (argc == 2 && strcmp(argv[1], "-n") == 0) {
            /* Sortiramo po naslovu */
260            qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
        } else {
262            if (argc == 2 && strcmp(argv[1], "-i") == 0) {
                /* Sortiramo po izvodjacu */
264                qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
            } else {
266                printf("Nedozvoljeni argumenti!\n");
                free(pesme);
268                return 0;
            }
        }
270    }
    }

272
    /* Ispisujemo rezultat */
274     for (i = 0; i < n; i++) {
        printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
276             pesme[i].broj_gledanja);
    }

278
    /* Oslobadjamo memoriju */
280     for (i = 0; i < n; i++) {
```

```

    free(pesme[i].izvodjac);
282    free(pesme[i].naslov);
    }
284    free(pesme);

286    /* Prekidamo izvršavanje programa */
    return 0;
288 }

```

### Rešenje 3.28

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
7  #define MAX 100
   /* Program demonstrira upotrebu funkcija qsort() i bsearch().
9   Ove funkcije su definisane u standardnoj biblioteci i za
   njihovo koriscenje dovoljno je ukljuciti zaglavlje stdlib.h.
11  Funkcija qsort() vrsi sortiranje niza metodom sortiranja
   razdvajanjem. Deklaracija ove funkcije je sledeca:
13
15  void qsort(void *b, int n, int s, int(*comp)(const void *,
   const void *));
17
19  Prvi argument je adresa pocetka niza koji se sortira. S
   obzirom da se ne zna tacan tip elemenata niza, koristi se
   genericki pokazivac (void *). Drugi argument je broj
21  elemenata niza, a treci velicina svakog od elemenata niza.
   Poslednji argument je pokazivac na funkciju poredjenja. Ova
23  funkcija treba da prihvata adrese elemenata niza koji se
   porede, i da vraca >0 ako je prvi element veci, <0 ako je
25  prvi element manji, a vraca 0 ako su elementi koji se porede
   jednaki. Na ovaj nacin se moze sortirati bilo koji niz,
27  dovoljno je da je na neki nacin funkcijom poredjenja
   definisan potpuni poredak medju elementima niza. Argumenti
   funkcije poredjenja su takodje genericki pokazivaci, opet
29  zato sto ne znamo tacno kog su tipa elementi niza. Ovi
   pokazivaci su jos kvalifikovani kljucnom recju const.
31
33  Funkcija bsearch() vrsi pretrazivanje sortiranog niza metodom
   binarne pretrage. Funkcija ima sledecu deklaraciju:
35
37  void *bsearch(const void *x, const void *b, int n, int s, int
   (*comp)(const void *, const void *));
39
41  Prvi argument je pokazivac na podatak koji se trazi u nizu.
   Drugi argument je adresa pocetka niza, treci velicina niza, a
   cetvrti velicina elementa niza. Poslednji argument je
43  pokazivac na funkciju poredjenja koja definise poredak u
   skladu sa kojim je sortiran niz. Funkcija vraca adresu
   elementa u nizu koji je jednak trazenom elementu, ili NULL
   ukoliko element nije pronadjen.

```

```
45     Osim ovih funkcija, postoji funkcija za linearnu pretragu.
47
49     void *lfind(const void *x, void *b, int *n, int s, int
        (*comp)(const void *, const void *));
51
53     ima potpuno iste argumente kao i bsearch() -- jedina razlika
        je u tome sto se kao treci argument ne predaje duzina niza vec
        adresa celobrojne promenljive u kojoj se nalazi duzina niza.
55     Funkcija za uporedjivanje elemenata na koju pokazuje pokazivac
        comp treba da zadovoljava ista pravila kao i kod prethodnih
        funkcija. Medjutim, s obzirom da se kod linearne pretrage
57     koristi iskljucivo poredjenje na jednakost, dovoljno je da
        funkcija za uporedjivanje vraca 0, ako su objekti koji se
59     uporeduju jednaki, a razlicitu vrednost od nule u suprotnom.
        Funkcija vraca pokazivac na pronadjeni element ili NULL u
61     slucaju neuspesne pretrage. */
63
65     /* NAPOMENA: Pokazivac na void je pokazivac koji moze sadrzati
        adresu bilo kog podatka u memoriji. Svaki pokazivac se moze
        bez konverzije dodeliti ovom pokazivacu, kao i obrnuto. Moze
        se koristiti za cuvanje adrese podatka za koji unapred ne
67     znamo kog ce biti tipa. Njegova glavna karakteristika je da
        se ne moze dereferencirati, zato sto se ne zna kog je tipa
        ono na sta on pokazuje. Programer mora na drugi nacin da
69     utvrdi kog je tipa podatak na toj adresi, pa da najpre
        konvertuje void pokazivac u odgovarajuci tip pokazivaca, a
71     zatim da ga tako konvertovanog dereferencira. */
73
75     /* Funkcija poredi dva cela broja */
76     int compare_int(const void *a, const void *b)
77     {
78         /* Konvertujemo void pokazivace u int pokazivace koje zatim
            dereferenciramo, dobijamo int-ove koje oduzimamo i razliku
79         vracamo. */
81
83         /* Zbog moguceg prekoracenja opsega celih brojeva necemo ih
            oduzimati return *((int *)a) - *((int *)b); */
85
86         int b1 = *((int *) a);
87         int b2 = *((int *) b);
89
90         if (b1 > b2)
91             return 1;
92         else if (b1 < b2)
93             /* Ovo uredjenje favorizujemo jer zelimo rastuci poredak */
94             return -1;
95         else
96             return 0;
97     }
98
99     int compare_int_desc(const void *a, const void *b)
100    {
101        /* Za obrnuti poredak mozemo samo oduzimati a od b */
102        /* return *((int *)b) - *((int *)a); */
103    }
```

```
101  /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
      funkcija */
103  return -compare_int(a, b);
105  }
107  /* Test program */
108  int main()
109  {
110      size_t n;
111      int i, x;
112      int a[MAX], *p = NULL;
113
114      /* Unosimo dimenziju */
115      printf("Uneti dimenziju niza: ");
116      scanf("%ld", &n);
117      if (n > MAX)
118          n = MAX;
119
120      /* Unosimo elemente niza */
121      printf("Uneti elemente niza:\n");
122      for (i = 0; i < n; i++)
123          scanf("%d", &a[i]);
124
125      /* Sortiramo niz celih brojeva */
126      qsort(a, n, sizeof(int), &compare_int);
127
128      /* Prikazujemo sortirani niz */
129      printf("Sortirani niz u rastucem poretku:\n");
130      for (i = 0; i < n; i++)
131          printf("%d ", a[i]);
132      putchar('\n');
133
134      /* Pretrazivanje niza */
135      /* Vrednost koju cemo traziti u nizu */
136      printf("Uneti element koji se trazi u nizu: ");
137      scanf("%d", &x);
138
139      printf("Binarna pretraga: \n");
140      p = bsearch(&x, a, n, sizeof(int), &compare_int);
141
142      if (p == NULL)
143          printf("Elementa nema u nizu!\n");
144      else
145          printf("Element je nadjen na poziciji %ld\n", p - a);
146
147      printf("Linearna pretraga (lfind): \n");
148      p = lfind(&x, a, &n, sizeof(int), &compare_int);
149      if (p == NULL)
150          printf("Elementa nema u nizu!\n");
151      else
152          printf("Element je nadjen na poziciji %ld\n", p - a);
153
154      return 0;
155  }
```

#### Rešenje 3.29

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <search.h>
5
6 #define MAX 100
7
8 /* Funkcija racuna broj delilaca broja x */
9 int no_of_deviders(int x)
10 {
11     int i;
12     int br;
13
14     /* Ako je argument negativan broj menjamo mu znak */
15     if (x < 0)
16         x = -x;
17     if (x == 0)
18         return 0;
19     if (x == 1)
20         return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22     br = 2;
23     /* Sve dok je */
24     for (i = 2; i < sqrt(x); i++)
25         if (x % i == 0)
26             /* Ako i deli x onda su delioci: i, x/i */
27             br += 2;
28     /* Ako je broj bas kvadrat, onda smo iz petlje izašli kada je
29     i bilo bas jednako korenu od x, tada x ima jos jednog
30     delioca */
31     if (i * i == x)
32         br++;
33
34     return br;
35 }
36
37 /* Funkcija poredjenja dva cela broja po broju delilaca */
38 int compare_no_deviders(const void *a, const void *b)
39 {
40     int ak = *(int *) a;
41     int bk = *(int *) b;
42     int n_d_a = no_of_deviders(ak);
43     int n_d_b = no_of_deviders(bk);
44
45     if (n_d_a > n_d_b)
46         return 1;
47     else if (n_d_a < n_d_b)
48         return -1;
49     else
50         return 0;
51 }
52
53 /* Test program */
54 int main()
```

```

{
56     size_t n;
    int i;
58     int a[MAX];

60     /* Unosimo dimenziju */
    printf("Uneti dimenziju niza: ");
62     scanf("%ld", &n);
    if (n > MAX)
64         n = MAX;

66     /* Unosimo elemente niza */
    printf("Uneti elemente niza:\n");
68     for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
70

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

74     /* Prikazujemo sortirani niz */
    printf
76         ("Sortirani niz u rastucem poretku prema broju delilaca:\n");
    for (i = 0; i < n; i++)
78         printf("%d ", a[i]);
    putchar('\n');
80

    return 0;
82 }

```

### Rešenje 3.30

```

#include <stdio.h>
2  #include <stdlib.h>
#include <string.h>
4  #include <search.h>
#define MAX_NISKI 1000
6  #define MAX_DUZINA 30

8  /* Naredne dve funkcije bice koriscenje za sortiranje niza
   nizova karaktera. Svaka od njih ce biti pozivana za
10  poredjenje dva elementa takvog niza, tj dva niza karaktera.
   Prilikom poziva ove funkcije za poredjenje, npr i-tog i j-tog
12  elementa slale bi se, kao i inace adrese elemenata. Kako
   adresa niza i samo ime niza imaju istu vrednost, pa i slanje
14  &niske[i] u pozivu funkcije je isto kao da se salje niske[i]
   i nije neophodno dalje dereferenciranje da bi se doslo do
16  i-te niske niza. */

18  /* Funkcija koju koristimo za poredjenje na dve niske iz niza a
   i b su const void pokazivaci, ali kako mi znamo da cemo ovu
20  funkciju koristiti za poredjenje dve niske iz niza
   eksplicitno im menjamo tipove u (char *) To je neophodno jer
22  poredimo niske leksikografski i funkciji strcmp moramo
   proslediti bas char* pokazivace, da bi uporedila 2 niske. */
24  int poredi_leksikografski(const void *a, const void *b)

```

```
{
26  return strcmp((char *) a, (char *) b);
}

28
/* Funkcija koju cemo prosledjivati za sortiranje niski po
30  duzini. Dakle a i b ce biti uvek adrese niski koje se porede.
   Menjamo im, eksplicitno, tip u char* jer nam treba duzina
32  svake od niski, a to cemo dobiti pozivom funkcije strlen koja
   ocekuje bas char * pokazivac. */
34  int poredi_duzine(const void *a, const void *b)
   {
36      return strlen((char *) a) - strlen((char *) b);
   }

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

48   /* Otvaranje datoteke */
   if ((fp = fopen("niske.txt", "r")) == NULL) {
50       fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
       exit(EXIT_FAILURE);
52   }

54   /* Citanje sadrzaja datoteke */
   for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);

56
   /* Zatvaranje datoteke */
58   fclose(fp);
   n = i;

60
   /* Sortiramo niske leksikografski, tako sto biblioteckoj
62   funkciji qsort prosledjujemo funkciju kojom se zadaje
       kriterijum poredjenja 2 niske po duzini */
64   qsort(niske, n, MAX_DUZINA * sizeof(char),
       &poredi_leksikografski);

66
   printf("Leksikografski sortirane niske:\n");
68   for (i = 0; i < n; i++)
       printf("%s ", niske[i]);
70   printf("\n");

72   /* Unosimo trazeni nisku */
   printf("Uneti trazenu nisku: ");
74   scanf("%s", x);

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



```

82  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
84         p, (p - (char *) niske) / MAX_DUZINA);
    else
86     printf("Niska nije pronadjena u nizu\n");

88  /* Linearna pretraga */
/* Prosledjujemo pokazivac na funkciju poredi_leksikografski
90   jer nam je niz sortiran leksikografski. */
p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
92         &poredi_leksikografski);

94  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
96         p, (p - (char *) niske) / MAX_DUZINA);
    else
98     printf("Niska nije pronadjena u nizu\n");

100 /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
    funkciju poredjenja */
102 qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

104 printf("Niske sortirane po duzini:\n");
for (i = 0; i < n; i++)
106     printf("%s ", niske[i]);
    printf("\n");
108
    exit(EXIT_SUCCESS);
110 }

```

### Rešenje 3.31

```

#include <stdio.h>
2  #include <stdlib.h>
#include <string.h>
4  #include <search.h>
#define MAX_NISKI 1000
6  #define MAX_DUZINA 30

8  /* Fje poredjenja za qsort */
int poredi_leksikografski(const void *a, const void *b)
10 {
    return strcmp(*(char **) a, *(char **) b);
12 }

14 int poredi_duzine(const void *a, const void *b)
{
16     return strlen(*(char **) a) - strlen(*(char **) b);
}

18 /* Fja poredjenja za bsearch */
20 int poredi_leksikografski_b(const void *a, const void *b)
{
22     return strcmp((char *) a, *(char **) b);

```

```
}
24
int main()
26 {
    int i;
28    size_t n;
    FILE *fp = NULL;
30    char *niske[MAX_NISKI];
    char **p = NULL;
32    char x[MAX_DUZINA];

34    /* Otvaranje datoteke */
    if ((fp = fopen("niske.txt", "r")) == NULL) {
36        fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
        exit(EXIT_FAILURE);
38    }

40    /* Citanje sadrzaja datoteke */
    i = 0;
42    while (fscanf(fp, "%s", x) != EOF) {
        /* Alociranje dovoljne memorije */
44        if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
            fprintf(stderr, "Greska pri alociranju niske\n");
46            exit(EXIT_FAILURE);
        }
48        /* Kopiranje procitane niske na svoje mesto */
        strcpy(niske[i], x);
50        i++;
    }

52    /* Zatvaranje datoteke */
54    fclose(fp);
    n = i;

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

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

66    /* Unosimo trazeni nisku */
68    printf("Uneti trazenu nisku: ");
    scanf("%s", x);

70    /* Binarna pretraga */
72    /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
        jer nam je niz sortiran leksikografski. */
74    p = bsearch(&x, niske, n, sizeof(char *),
                &poredi_leksikografski_b);

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

```

        *p, p - niske);
80  else
    printf("Niska nije pronadjena u nizu\n");
82
    /* Linearna pretraga */
84  /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
    jer nam je niz sortiran leksikografski. */
86  p = lfind(&x, niske, &n, sizeof(char *),
    &poredi_leksikografski_b);
88
    if (p != NULL)
89     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
    *p, p - niske);
90  else
    printf("Niska nije pronadjena u nizu\n");
92
    /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
    funkciju poredjenja */
94  qsort(niske, n, sizeof(char *), &poredi_duzine);
96
    printf("Niske sortirane po duzini:\n");
98  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
100 printf("\n");
102
    /* Oslobadjanje zauzete memorije */
104 for (i = 0; i < n; i++)
    free(niske[i]);
106
    exit(EXIT_SUCCESS);
108 }

```

### Rešenje 3.32

```

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

6  #define MAX 500

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

16 /* Funkcija poredjenja koju cemo koristiti za sortiranje po
    broju bodova, a studente sa istim brojevem bodova dodatno
18  sortiramo leksikografski po prezimenu */
int compare(const void *a, const void *b)
20 {
    Student *prvi = (Student *) a;

```

### 3 Algoritmi pretrage i sortiranja

```
22 Student *drugi = (Student *) b;

24 if (prvi->bodovi > drugi->bodovi)
    return -1;
26 else if (prvi->bodovi < drugi->bodovi)
    return 1;
28 else
    /* Jednaki su po broju bodova, treba ih uporediti po
30    prezimenu */
    return strcmp(prvi->prezime, drugi->prezime);
32 }

34 /* Funkcija za poredjenje koje ce porediti samo po broju bodova
    Prvi parametar je ono sto trazimo u nizu, ovde je to broj
36    bodova, a drugi parametar ce biti element niza ciji se bodovi
    porede. */
38 int compare_zabsearch(const void *a, const void *b)
{
40     int bodovi = *(int *) a;
    Student *s = (Student *) b;
42     return s->bodovi - bodovi;
}

44

46 /* Funkcija za poredjenje koje ce porediti samo po prezimenu
    Prvi parametar je ono sto trazimo u nizu, ovde je to prezime,
    * a drugi parametar ce biti element niza cije se prezime
48    poredi. */
int compare_zalinearnaprezimena(const void *a, const void *b)
50 {
    char *prezime = (char *) a;
52     Student *s = (Student *) b;
    return strcmp(prezime, s->prezime);
54 }

56

58 int main(int argc, char *argv[])
{
    Student kolokvijum[MAX];
60     int i;
    size_t br_studenata = 0;
62     Student *nadjen = NULL;
    FILE *fp = NULL;
64     int bodovi;
    char prezime[21];
66

    /* Ako je program pozvan sa nedovoljnim brojem argumenata
68     informisemo korisnika kako se program koristi i prekidamo
    izvršavanje. */
70     if (argc < 2) {
        fprintf(stderr,
72             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
                argv[0]);
74         exit(EXIT_FAILURE);
    }
76

    /* Otvaranje datoteke */
```

```

78  if ((fp = fopen(argv[1], "r")) == NULL) {
79      fprintf(stderr, "Neuspšno otvaranje datoteke %s\n", argv[1]);
80      exit(EXIT_FAILURE);
81  }
82
83  /* Ucitavanje sadrzaja */
84  for (i = 0;
85      fscanf(fp, "%s%s%d", kolokvijum[i].ime,
86              kolokvijum[i].prezime,
87              &kolokvijum[i].bodovi) != EOF; i++);
88
89  /* Zatvaranje datoteke */
90  fclose(fp);
91  br_studenata = i;
92
93  /* Sortiramo niz studenata po broju bodova, pa unutar grupe
94     studenata sa istim brojem bodova sortiranje se vrši po
95     prezimenu */
96  qsort(kolokvijum, br_studenata, sizeof(Student), &compare);
97
98  printf("Studenti sortirani po broju poena opadajuće, ");
99  printf("pa po prezimenu rastuće:\n");
100  for (i = 0; i < br_studenata; i++)
101      printf("%s %s %d\n", kolokvijum[i].ime,
102              kolokvijum[i].prezime, kolokvijum[i].bodovi);
103
104  /* Pretrazivanje studenata po broju bodova se vrši binarnom
105     pretragom jer je niz sortiran po broju bodova. */
106  printf("Unesite broj bodova: ");
107  scanf("%d", &bodovi);
108
109  nadjen =
110      bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
111              &compare_za_bsearch);
112
113  if (nadjen != NULL)
114      printf
115          ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n"
116           ,
117           nadjen->ime, nadjen->prezime, nadjen->bodovi);
118  else
119      printf("Nema studenta sa unetim brojem bodova\n");
120
121  /* Pretraga po prezimenu se mora vršiti linearnom pretragom
122     jer nam je niz sortiran po bodovima, globalno gledano. */
123  printf("Unesite prezime: ");
124  scanf("%s", prezime);
125
126  nadjen =
127      lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
128              &compare_za_linearna_prezime);
129
130  if (nadjen != NULL)
131      printf
132          ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
133           nadjen->ime, nadjen->prezime, nadjen->bodovi);

```

### 3 Algoritmi pretrage i sortiranja

---

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

#### Rešenje 3.33

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

5  #define MAX 128

7  /* Funkcija poredi dva karaktera */
   int uporedi_char(const void *pa, const void *pb)
9  {
   return *(char *) pa - *(char *) pb;
11 }

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

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

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

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

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

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

45     return 0;
}
```

## Rešenje 3.34

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 10
6 #define MAX_DUZINA 32
7
8 /* Funkcija poredi dve niske (stringove) */
9 int uporedi_niske(const void *pa, const void *pb)
10 {
11     return strcmp((char *) pa, (char *) pb);
12 }
13
14 int main()
15 {
16     int i, n;
17     char S[MAX][MAX_DUZINA];
18
19     printf("Unesite broj niski:");
20     scanf("%d", &n);
21
22     printf("Unesite niske:\n");
23     for (i = 0; i < n; i++)
24         scanf("%s", S[i]);
25
26     /* Sortiramo niz niski */
27     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
28
29     /******
30     printf("Sortirane niske su:\n");
31     for(i = 0; i < n; i++)
32         printf("%s ", S[i]);
33     *****/
34
35     /* Ako postoje dve iste niske u nizu, onda ce one nakon
36        sortiranja niza biti jedna do druge */
37     for (i = 0; i < n - 1; i++)
38         if (strcmp(S[i], S[i + 1]) == 0) {
39             printf("ima\n");
40             return 0;
41         }
42
43     printf("nema\n");
44     return 0;
45 }

```

## Rešenje 3.35

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 typedef struct student {

```

```

    char nalog[8];
7   char ime[21];
    char prezime[21];
9   int poeni;
} Student;
11

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

21
23 /* Funkcija poredi studente prvo prema godini, zatim prema smeru
    i na kraju prema indeksu */
int uporedi_nalog(const void *a, const void *b)
25 {
27     Student s = *(Student *) a;
    Student t = *(Student *) b;
    /* Za svakog studenta iz naloga izdvajamo godinu upisa, smer i
29     broj indeksa */
    int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
31     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
    char smer1 = s.nalog[1];
33     char smer2 = t.nalog[1];
    int indeks1 =
35         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
        s.nalog[6] - '0';
37     int indeks2 =
        (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
39         t.nalog[6] - '0';
    if (godina1 != godina2)
41         return godina1 - godina2;
    else if (smer1 != smer2)
43         return smer1 - smer2;
    else
45         return indeks1 - indeks2;
}

47
int uporedi_bsearch(const void *a, const void *b)
49 {
    /* Nalog studenta koji se trazi */
51     char *nalog = (char *) a;
    /* Kljuc pretrage */
53     Student s = *(Student *) b;

55     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
    int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
57     char smer1 = nalog[1];
    char smer2 = s.nalog[1];
59     int indeks1 =
        (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] -
61     '0';

```



```

63     int indeks2 =
        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
        s.nalog[6] - '0';
65     if (godina1 != godina2)
        return godina1 - godina2;
67     else if (smer1 != smer2)
        return smer1 - smer2;
69     else
        return indeks1 - indeks2;
71 }

73 int main(int argc, char **argv)
{
75     Student *nadjen = NULL;
    char nalog_trazeni[8];
77     Student niz_studenata[100];
    int i = 0, br_studenata = 0;
79     FILE *in = NULL, *out = NULL;

81     /* Ako je broj argumenata komandne linije razlicit i od 2 i od
        3, korisnik nije ispravno pozvao program i prijavljujemo
83     gresku: */
    if (argc != 2 && argc != 3) {
85         fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out -opcija (nalog)!\n");
87         exit(EXIT_FAILURE);
    }

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

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

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

111     br_studenata = i;

113     /* Ako je student uneo opciju -p, vrsi se sortiranje po
        poenima */
115     if (strcmp(argv[1], "-p") == 0)

```

### 3 Algoritmi pretrage i sortiranja

```
117     qsort(niz_studenata, br_studenata, sizeof(Student),
          &uporedi_poeni);
119 /* A ako je uneo opciju -n, vrsi se sortiranje po nalogu */
120 else if (strcmp(argv[1], "-n") == 0)
121     qsort(niz_studenata, br_studenata, sizeof(Student),
          &uporedi_nalog);
123
124 /* Sortirani studenti se ispisuju u izlaznu datoteku */
125 for (i = 0; i < br_studenata; i++)
126     fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
127           niz_studenata[i].ime, niz_studenata[i].prezime,
128           niz_studenata[i].poeni);
129
130 /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
131    studenta... */
132 if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
133     strcpy(nalog_trazeni, argv[2]);
134
135     /* ... pronalazi se student sa tim nalogom... */
136     nadjen =
137         (Student *) bsearch(nalog_trazeni, niz_studenata,
138                           br_studenata, sizeof(Student),
139                           &uporedi_bsearch);
140
141     if (nadjen == NULL)
142         printf("Nije nadjen!\n");
143     else
144         printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
145               nadjen->prezime, nadjen->poeni);
146 }
147
148 fclose(in);
149 fclose(out);
150 return 0;
151 }
```

#### Rešenje 3.37

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
5    standardnog ulaza */
6 void ucitaj_matricu(int **a, int n, int m)
7 {
8     printf("Unesite elemente matrice po vrstama:\n");
9     int i, j;
10
11     for (i = 0; i < n; i++) {
12         for (j = 0; j < m; j++) {
13             scanf("%d", &a[i][j]);
14         }
15     }
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
28     return zbir;
29 }
30
31 /* Funkcija koja sortira vrste matrice na osnovu zbira
    koriscenjem selection 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
39         for (j = i + 1; j < n; j++) {
40             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
41                 min = j;
42             }
43         }
44         if (min != i) {
45             int *tmp;
46             tmp = a[i];
47             a[i] = a[min];
48             a[min] = tmp;
49         }
50     }
51 }
52
53 /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
    standardni izlaz */
54 void ispisi_matricu(int **a, int n, int m)
55 {
56     int i, j;
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 alocira memoriju za matricu dimenzija nxm */
67 int **alociraj_memoriju(int n, int m)
68 {
69     int i, j;
```

### 3 Algoritmi pretrage i sortiranja

---

```
74  int **a;

76  a = (int **) malloc(n * sizeof(int *));
78  if (a == NULL) {
80      printf("Problem sa alokacijom memorije!\n");
      exit(EXIT_FAILURE);
82  }

84  /* Za svaku vrstu ponasob */
86  for (i = 0; i < n; i++) {

88      /* Alociramo memoriju */
89      a[i] = (int *) malloc(m * sizeof(int));

90      /* Proveravamo da li je doslo do greske prilikom alokacije */
91      if (a[i] == NULL) {
92          /* Ako jeste, ispisujemo poruku */
93          printf("Problem sa alokacijom memorije!\n");

94          /* I oslobadjamo memoriju zauzetu do ovog koraka */
95          for (j = 0; j < i; j++) {
96              free(a[j]);
97          }
98          free(a);
99          exit(EXIT_FAILURE);
100      }
101  }

102  return a;
103  }

104  /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije
105     nxm */
106  void oslobodi_memoriju(int **a, int n, int m)
107  {
108      int i;
109
110      for (i = 0; i < n; i++) {
111          free(a[i]);
112      }
113      free(a);
114  }

116

118  int main(int argc, char *argv[])
119  {
120      int **a;
121      int n, m;

122      /* Citamo dimenzije matrice */
123      printf("Unesite dimenzije matrice: ");
124      scanf("%d %d", &n, &m);
125  }
```

```
130  /* Alociramo memoriju */  
    a = alociraj_memoriju(n, m);  
132  
    /* Ucitavamo elemente matrice */  
134  ucitaj_matricu(a, n, m);  
  
136  /* Pozivamo funkciju koja sortira vrste matrice prema zbiru */  
    sortiraj_vrste(a, n, m);  
138  
    /* Ispisujemo rezultujucu matricu */  
140  printf("Sortirana matrica je:\n")  
        ispisi_matricu(a, n, m);  
142  
    /* Oslobadjamo memoriju */  
144  oslobodi_memoriju(a, n, m);  
  
146  /* I prekidamo sa izvršavanjem programa */  
    return 0;  
148 }
```



# Glava 4

## Dinamičke strukture podataka

### 4.1 Liste

**Zadatak 4.1** Napisati program koji koristi jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (a) Definirati strukturu `Cvor` koja predstavlja čvor liste.
- (b) Milena: Da li ovde treba dodati i funkciju koja kreira cvor? Nemam resenje kod sebe pa ne znam kako to vec ide, ali mislim da bi trebalo
- (c) Napisati funkciju koja dodaje novi element na početak liste.
- (d) Napisati funkciju koja dodaje novi element na kraj liste.
- (e) Milena: Da li bi ovo trebalo izdvojiti u poseban zadatak? Nekako, ako dodajemo na pocetak i kraj, nemamo garanciju sortiranosti liste, tako da mi to nekako deluje da smo dva zadatka strpali u jedan. Napisati funkciju koja dodaje novi element u listu tako da lista ostane rastuće sortirana.
- (f) Napisati funkciju koja oslobađa memoriju koju je zauzela lista.
- (g) Milena: Ova funkcija bi mogla razlicito da se implementira sa pretpostavkom da je lista sortirana i da nije sortirana, i zato mi dodatno deluje da bi ta dva zadatka trebalo razdvojiti Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- (h) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- (i) Milena: Da li ovde nedostaje funkcija koja oslobadja celu memoriju?

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekursivno.

**Zadatak 4.2** Napisati program koji koristi dvostruko povezanu listu za čuvanje celih brojeva koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu se prekida učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. **I ovde isto mozda razdvojiti sortiranost od obične liste.**

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

Sve funkcije za rad sa listom implementirati iterativno.

**Zadatak 4.3** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz. **Milena: promenjeni test primeri, voditi racuna u resenjima sta se stampa!**

*Test 1*

```
|| Datoteka: {[23 + 5344] * (24 - 234)} - 23
|| Izlaz:   Zagrade su ispravno uparene.
```

*Test 2*

```
|| Datoteka: {[23 + 5] * (9 * 2)} - {23}
|| Izlaz:   Zagrade su ispravno uparene.
```

*Test 3*

```
|| Datoteka: {[2 + 54] / (24 * 87)} + (234 + 23)
|| Izlaz:   Zagrade nisu ispravno uparene.
```

**Zadatak 4.4** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. **Milena: A sta ako se ne navede argument komandne linije?** Uputstvo: za rešavanje problema koristiti stek implementiran preko listi čiji su čvorovi HTML etikete.



*Test 1*

```

Poziv: ./a.out datoteka.html
Datoteka.html:                                Izlaz:
<html>                                          Ispravno uparene etikete.
  <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>

```

*Test 2*

```

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

```

*Test 3*

```

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

```

**Zadatak 4.5** Milena: Problem sa ovim zadatkom je sto je program najpre na usluzi korisnicima, a zatim na usluzi sluzbeniku i to nekako zbunjuje u formulaicji. Formulacija mi nije bila jasna bez citanja resenja, pokusala sam da je preciziran, u nastavku je izmenjena formulacija.

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

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

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

prekida unošenjem karaktera za kraj ulaza (EOF).

Službenik redom proziva korisnike čitanjem njihovog `jmbg` broja, a zatim odlučuje da li korisnika vraća na kraj reda ili ga odmah uslužuje. Službeniku se postavlja pitanje `Da li korisnika vracate na kraj reda?` i ukoliko on da odgovor `Da`, korisnik se vraća na kraj reda. Ukoliko odgovor nije `Da`, tada službenik čita korisnikov zahtev. Posle svakog 10 usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu.

Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.

**Zadatak 4.6 Milena:** Dodati sta se desava ako nije zadat argument komandne linije ili ako datoteka ne postoji 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

<b>Poziv:</b> <code>./a.out datoteka.html</code>	
<b>Datoteka.html:</b>	<b>Izlaz:</b>
<code>&lt;html&gt;</code>	<code>a - 4</code>
<code>&lt;head&gt;&lt;title&gt;Primer&lt;/title&gt;&lt;/head&gt;</code>	<code>br - 1</code>
<code>&lt;body&gt;</code>	<code>h1 - 2</code>
<code>&lt;h1&gt;Naslov&lt;/h1&gt;</code>	<code>body - 2</code>
<code>Danas je lep i suncan dan. &lt;br&gt;</code>	<code>title - 2</code>
<code>A sutra ce biti jos lepsi.</code>	<code>head - 2</code>
<code>&lt;a link="http://www.google.com"&gt; Link 1&lt;/a&gt;</code>	<code>html - 2</code>
<code>&lt;a link="http://www.math.rs"&gt; Link 2&lt;/a&gt;</code>	
<code>&lt;/body&gt;</code>	
<code>&lt;/html&gt;</code>	

**Zadatak 4.7 Milena:** i ovde dodati sta ako nema argumenata i ako nema datoteka, kao i u svim ostalim zadacima, a ne bih stalno ovaj komentar ponavljala. Takodje, malo me muci u ovom zadatku sto nema neki smisao. Naime, ako se samo vrsi učitavanje iz datoteka i ispisivanje, onda su ove liste zapravo visak jer isti rezultat moze da se dobije i bez koriscenja listi. Zato mi fali da program uradi nesto sto ne bi mogao da uradi bez koriscenja listi, npr da na osnovu unetog broja ispisuje svaki n-ti broj rezultujuce liste pa to u nekoj petlji da korisnik moze da ispisuje za razlicite unete n ili tako nesto...

Napisati program koji objedinjuje dve sortirane liste. Funkcija ne treba da kreira nove čvorove, već da samo postojeće čvorove preraspodeli. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije

se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

*Test 1*

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz: 2 4 5 6 6 10 11 12 14 15
      16
```

**Zadatak 4.8** Napisati funkciju koja formira listu studenata tako što se podaci o studentima učitavaju iz datoteke čije se ime zadaje kao argument komandne linije. U svakom redu datoteke nalaze se podaci o studentu i to broj indeksa, ime i prezime. Napisati rekurzivnu funkciju koja određuje da li neki student pripada listi ili ne. Ispisati zatim odgovarajuću poruku i rekurzivno osloboditi memoriju koju je data lista zauzimala. Student se traži na osnovu broja indeksa, koji se zadaje sa standardnog ulaza.

*Test 1*

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

**Milena:** Imamo dva zadatka sa labelom 608!

**Zadatak 4.9** Neka su date dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od tih lista formira novu listu L koja sadrži alternirajući raspoređene elemente lista L1 i L2 (prvi element iz L1, prvi element iz L2, drugi element L1, drugi element L2, itd). Ne formirati nove čvorove, već samo postojeće čvorove rasporediti u jednu listu. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. **Milena:** Sta ako je neka lita duza? To precizirati. I ovde me muci sto nedostaje neki smisao zadatku, nesto sto ne bi moglo da se uradi da nismo kristili liste.

*Test 1*

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz: 2 5 4 6 6 11 10 12 15 14
      16
```

**Zadatak 4.10** Data je datoteka `brojevi.txt` koja sadrži cele brojeve, po jedan u svakom redu.

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

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

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

### Test 1

Ulaz: <code>brojevi.txt</code>	Izlaz: <code>Rezultat.txt</code>
43	12
12	15
15	16
16	
4	
2	
8	

**Zadatak 4.11** Grupa od  $n$  plesača na kostimima imaju brojeve od 1 do  $n$ , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi  $k$ -ti plesač. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. Uputstvo: u implementaciji koristiti kružnu listu.

### Test 1

Ulaz: 5 3
Izlaz: 3 1 5 2 4

Milena: Bilo bi lepo dodati i prethodni zadatak u kojem se smer izbacivanja stalno menja, tako da se onda koristi dvostruko povezana kružna lista.

## 4.2 Stabla

**Zadatak 4.12** Napisati program za rad sa binarnim pretraživačkim stablima.

- (a) Definirati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog stabla koja sadrži ceo broj `b` i pokazivače `levo` i `desno` redom na levo i desno

podstablo<sup>1</sup>.

- (b) Napisati funkciju `Cvor* napravi_cvor(int b)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `b`.
- (c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int b)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `b`.
- (d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int b)` koja proverava da li se ceo broj `b` 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 b)` koja briše čvor koji sadrži vrednost `b` 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.

---

<sup>1</sup>U zadacima ove glave u kojima nije eksplicitno naglašen sadržaj čvorova stabla, podrazumevaće se ova struktura.

### Test 1

```
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
```

### Test 2

```
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
```

**Zadatak 4.13** Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski prema rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku *Nedostaje ime ulazne datoteke!*.

*Milena: dodati i test primer sa pokretanjem bez ulazne datoteke*

### Test 1

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

Ulazna datoteka test.txt:
Sunce utorak raCunar SUNCE
    programiranje jabuka
    PROGramiranje sunCE JABUka

Izlaz:
jabuka: 2
programiranje: 2
racunar: 1
sunce: 3
utorak: 1
```

### Test 2

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

Ulazna datoteka suma.txt:
lipa zova hrast ZOVA breza LIPA

Izlaz:
breza: 1
hrast: 1
lipa: 2
zova: 2
```

**Zadatak 4.14** 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

unosu 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.

*Test 1*

```

Ulazna datoteka imenik.txt:
  Pera Peric 011/3240-987
  Marko Maric 064/1234-987
  Mirko Maric 011/589-333
  Sanja Savkovic 063/321-098
  Zika Zikic 021/759-858

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

```

**Zadatak 4.15** 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*

```

Datoteka prijemni.txt:
  Marko Markovic 45.4 12.3 11
  Milan Jevremovic 35.2 1.3 9
  Maja Agic 60 19 20
  Nadica Zec 54.2 10 15.8
  Jovana Milic 23.3 2 5.6

Izlaz:
1. Maja Agic 60 19 20 99
2. Nadica Zec 54.2 10 15.8 80
3. Marko Markovic 45.4 12.3 11 68.7
4. Milan Jevremovic 35.2 1.3 9 45.5
-----
5. Jovana Milic 23.3 2 5.6 30.9

```

**Zadatak 4.16** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadanog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.YYYY.`

### Test 1

```
Ulazna datoteka rodjendani.txt:
Marko Markovic 12.12.1990.
Milan Jevremovic 04.06.1989.
Maja Agic 23.04.2000.
Nadica Zec 01.01.1993.
Jovana Milic 05.05.1990.
...
Unesite datum: 23.04.
Slavljenik: Maja Agic
Unesite datum: 01.01.
Slavljenik: Nadica Zec
Unesite datum: 01.05.
Slavljeni: Jovana Milic 05.05.
Unesite datum: CTRL+D
```

**Zadatak 4.17** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0.

### Test 1

```
Ulaz:
Prvo stablo: 10 5 15 3 2 4 30 12 14 13 0
Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0

Izlaz:
Stabla jesu identicna.
```

### Test 2

```
Ulaz:
Prvo stablo: 10 5 15 4 3 2 30 12 14 13 0
Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0

Izlaz:
Stabla nisu identicna.
```



**Zadatak 4.18** 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 uračunate tačno po jednom. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

*Test 1*

```

Ulaz:
Prvo stablo: 1 7 8 9 2 2
Drugo stablo: 3 9 6 11 1

Izlaz:
Unija: 1 2 3 6 7 8 9 11
Presek: 1 9
Razlika: 2 7 8

```

**Zadatak 4.19** Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

*Test 1*

```

Ulaz:
n: 7
a: 1 11 8 6 37 25 30

Izlaz:
1 6 8 11 25 30 37

```

*Test 2*

```

Ulaz:
n: 55

Izlaz:
Greska: pogresna dimenzija niza!

```

**Zadatak 4.20** 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.

- (f) Napisati funkciju koja izračunava dubinu stabla.
- (g) Napisati funkciju koja izračunava broj čvorova na  $i$ -tom nivou stabla.
- (h) Napisati funkciju koja ispisuje sve elemente na  $i$ -tom nivou stabla.
- (i) Napisati funkciju koja izračunava maksimalnu vrednost na  $i$ -tom nivou stabla.
- (j) Napisati funkciju koja izračunava zbir čvorova na  $i$ -tom nivou stabla.
- (k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti  $x$ .

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

#### Test 2

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

**Zadatak 4.21** Dato je binarno pretraživačko stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa maksimalnim proizvodom vrednosti iz desnog podstabla.
- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjom sumom vrednosti iz levog podstabla.
- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

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

*Test 1*

```

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

```

**Zadatak 4.22** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima.

*Test 1*

```

Ulaz:
10 5 15 3 2 4 30 12 14 13
Izlaz:
0.nivo: 10
1.nivo: 5 15
2.nivo: 3 12 30
3.nivo: 2 4 14
4.nivo: 13

```

**Zadatak 4.23** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

*Test 1*

```

Ulaz:
Prvo stablo: 11 20 5 3
Drugo stablo: 8 14 30 1
Izlaz:
Stabla su slicna kao u ogledalu.

```

### Test 2

```
Ulaz:
Prvo stablo: 11 20 5 3
Drugo stablo: 8 20 15

Izlaz:
Stabla nisu slicna kao u ogledalu
.
```

**Zadatak 4.24** AVL-stablo je binarno stablo pretrage kod koga apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

### Test 1

```
Ulaz:
10 5 15 2 11 16 1 13
Izlaz:
7
```

### Test 2

```
Ulaz:
16 30 40 24 10 18 45 22
Izlaz:
6
```

**Zadatak 4.25** Binarno stablo se naziva HEAP ako je kompletno (svaki njegov čvor, izuzev listova, ima i levog i desnog potomka) i za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva HEAP. Napisati zatim i glavni program koji ispisuje rezultat `heap` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

### Test 1

```
Ulaz:
100 19 36 17 3 25 1 2 7
Izlaz:
Da
```

## 4.3 Rešenja

### Rešenje 4.1

```
#include<stdio.h>

2
int main(){
4   printf("Hello bitovi!\n");
```

```
    return 0;
6 }
```

### Rešenje 4.2

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

### Rešenje 4.8

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

### Rešenje 4.4

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

### Rešenje 4.5

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

### Rešenje 4.6

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

### Rešenje 4.7

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

### Rešenje 4.8

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

### Rešenje 4.9

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

### Rešenje 4.10

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

### Rešenje 4.11

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

### Rešenje 4.12

### Rešenje 4.13

### Rešenje 4.14

Rešenje [4.15](#)

Rešenje [4.16](#)

Rešenje [4.17](#)

Rešenje [4.18](#)

Rešenje [4.19](#)

Rešenje [4.20](#)

Rešenje [4.21](#)

Rešenje [4.22](#)

Rešenje [4.23](#)

Rešenje [4.24](#)

Rešenje [4.25](#)





# 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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>ž Sadraj datoteke: 5 Programiranje Matematika 12345 dInAmiCnArEc Ispit Izlaz: Ispit Matematika Programiranje</pre>	<pre>ž Sadraj datoteke: 2 maksimalano poena Izlaz:</pre>	<pre>Problem: datoteka ne postoji Izlaz: -1</pre>

#### 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.

## 5 Ispitni rokovi

---

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  $-1$ .

<i>Test 1</i>	<i>Test 2</i>
<pre>Ulaz: 2 8 10 3 6 14 13 7 4 0 Izlaz: 20</pre>	<pre>Ulaz: 0 50 14 5 2 4 56 8 52 7 1 0 Izlaz: 50</pre>

**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:</pre>	<pre>Ulaz: -2 Izlaz (na stderr): -1</pre>

## 5.2 Programiranje 2, praktični deo ispita, jul 2015.

### Zadatak 5.4

Napisati program koji kao prvi argument komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati  $-1$  na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

### Test 1

```

Poziv: ./a.out fajl.txt test
Datoteka: Ovo je test primer.
          U njemu se rec test
          javlja
          vise puta. testtesttest
Izlaz: 5
    
```

### Test 2

```

Poziv: ./a.out
Izlaz (na stderr): -1
    
```

### Test 3

```

Poziv: ./a.out fajl.txt foo
Datoteka: (ne postoji)
Izlaz (na stderr): -1
    
```

### Test 4

```

Poziv: ./a.out fajl.txt .
Datoteka: (prazna)
Izlaz: 0
    
```

### Zadatak 5.5

Na početku datoteke "trouglovi.txt" nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

### Test 1

```

Datoteka: 4
           0 0 0 1.2 1 0
           0.3 0.3 0.5 0.5 0.9 1
           -2 0 0 0 0 1
           2 0 2 2 -1 -1
Izlaz:    2 0 2 2 -1 -1
           -2 0 0 0 0 1
           0 0 0 1.2 1 0
           0.3 0.3 0.5 0.5 0.9 1
    
```

### Test 2

```

Datoteka: 3
           1.2 3.2 1.1
           4.3
Izlaz:    -1
    
```

### Test 3

```

Datoteka: (nema datoteke)
Izlaz:    -1
    
```

### Test 4

```

Datoteka: 0
Izlaz:
    
```

**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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Ulaz:</code> 1 5 3 6 1 4 7 9 <code>Izlaz:</code> 1	<code>Ulaz:</code> 2 5 3 6 1 0 4 7 9 <code>Izlaz:</code> 2	<code>Ulaz:</code> 0 4 2 5 <code>Izlaz:</code> 0
<i>Test 4</i>	<i>Test 5</i>	
<code>Ulaz:</code> 3 <code>Izlaz:</code> 0	<code>Ulaz:</code> -1 4 5 1 7 <code>Izlaz:</code> 0	

### 5.3 Rešenja

#### Rešenje 5.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define MAX 50
5
6 void greska(){
7     printf("-1\n");
8     exit(EXIT_FAILURE);
9 }
10
11 int main(int argc, char* argv[]){
12
13     FILE* ulaz;
14     char** linije;
15     int i, j, n;
16
17     /* Proveravamo argumente komandne linije.
18     */
19     if(argc!=2){
20         greska();
21     }
22
23     /* Otvaramo datoteku čije 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     /* čitavamo broj linija. */
31     fscanf(ulaz, "%d", &n);
32
33     /* Alociramo memoriju na osnovu čitanog broja linija.*/
34     linije=(char**)malloc(n*sizeof(char*));
```

```

35     if(linije==NULL){
        greska();
    }
37     for(i=0; i<n; i++){
        linije[i]=malloc(MAX*sizeof(char));
39         if(linije[i]==NULL){
            for(j=0; j<i; j++){
41                 free(linije[j]);
            }
            free(linije);
            greska();
43         }
45     }
47     /* čUitavamo svih n linija iz datoteke. */
49     for(i=0; i<n; i++){
        fscanf(ulaz, "%s", linije[i]);
51     }

53     /* Ispisujemo u ćodgovarajuem poretku ćuitane linije koje
zadovoljavaju kriterijum. */
55     for(i=n-1; i>=0; i--){
        if(isupper(linije[i][0])){
            printf("%s\\n", linije[i]);
57         }
    }
59     /* đOslobaamo memoriju koju smo ćdinamiki alocirali. */
61     for(i=0; i<n; i++){
        free(linije[i]);
63     }

65     free(linije);

67     /* Zatvaramo datoteku. */
    fclose(ulaz);
69

71     /* šZavravamo sa programom. */
    return 0;
73 }

```

## Rešenje 5.2

```

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

4
int sumirajN (Cvor * koren, int n){
6     if(koren==NULL){
        return 0;
8     }

10    if(n==0){
        return koren->broj;
    }
}

```

```
12     }
14     return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
16 }
18 int main(){
19     Cvor* koren=NULL;
20     int n;
21     int nivo;
22
23     /* Čitamo vrednost nivoa */
24     scanf("%d", &nivo);
25
26     while(1){
27
28         /* Čitamo broj sa standardnog ulaza */
29         scanf("%d", &n);
30
31         /* Ukoliko je korisnik uneo 0, prekidamo dalje čitanje. */
32         if(n==0){
33             break;
34         }
35
36         /* A ako nije, dodajemo procitani broj u stablo. */
37         dodaj_u_stablo(&koren, n);
38
39     }
40
41     /* Ispisujemo rezultat rada žtraene funkcije */
42     printf("%d\n", sumirajN(koren,nivo));
43
44     /* dOslobaamo memoriju */
45     oslobodi_stablo(&koren);
46
47
48     /* Prekidamo šizvravanje programa */
49     return 0;
50 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 Cvor* napravi_cvor(int b ) {
6     Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
7     if( novi == NULL)
8         return NULL;
9
10    /* Inicijalizacija polja novog čvora */
11    novi->broj = b;
12    novi->levo = NULL;
13    novi->desno = NULL;
14
15    return novi;
```

```

17 }
19 void oslobodi_stablo(Cvor** adresa_korena) {
21     /* Prazno stablo i nema šta da se doslobaa */
23     if( *adresa_korena == NULL)
25         return;
27     /* Rekurzivno doslobaamo najpre levo, a onda i desno podstablo*/
29     if( (*adresa_korena)->levo )
31         oslobodi_stablo(&(*adresa_korena)->levo);
33     if( (*adresa_korena)->desno)
35         oslobodi_stablo(&(*adresa_korena)->desno);
37     free(*adresa_korena);
39     *adresa_korena =NULL;
41 }
43 void prover_i_alokaciju( Cvor* novi) {
45     if( novi == NULL) {
47         fprintf(stderr, "Malloc greska za nov cvor!\n");
49         exit(EXIT_FAILURE);
51     }
53 }
55 void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
57     /* Postojece stablo je prazno*/
59     if( *adresa_korena == NULL){
61         Cvor* novi = napravi_cvor(broj);
63         prover_i_alokaciju(novi);
65         *adresa_korena = novi; /* Kreirani čvor novi će biti od
67 sada koren stabla*/
69         return;
71     }
73     /* Brojeve šsmetamo u đureeno binarno stablo, pa
75 ako je broj koji ubacujemo manji od broja koji je u korenu */
77     if( broj < (*adresa_korena)->broj) /* dodajemo u levo
79 podstablo */
81         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
83     /* ako je broj manji ili jednak od broja koji je u korenu stabla
85 , dodajemo nov čvor desno od korena */
87     else
89         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
91 }

```

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura kojom se predstavlja čvor drveta */
5  typedef struct dcvor{
6      int broj;
7      struct dcvor* levo, *desno;
8  } Cvor;
9

```

```

11 /* Funkcija alokira prostor za novi čvor drveta, inicijalizuje polja
    strukture i čvraa čpokaziva na nov čvor */
12 Cvor* napravi_cvor(int b );
13
14 /* đOslobaamo čdinamiki alocirani prostor za stablo
15 * Nakon đoslobaanja se u čpozivajuoj funkciji koren
16 * postavlja NULL, jer je stablo prazno */
17 void oslobodi_stablo(Cvor** adresa_korena);
18
19
20 /* Funkcija proverava da li je novi čvor ispravno alocirani,
21 * i nakon toga prekida program */
22 void prover_i_alokaciju( Cvor* novi);
23
24
25 /* Funkcija dodaje novi čvor u stablo i
26 * žaurira vrednost korena stabla u čpozivajuoj funkciji.
27 */
28 void dodaj_u_stablo(Cvor** adresa_korena, int broj);
29
30 #endif

```

### Rešenje 5.3

```

1 #include <stdio.h>
2 #define MAX 50
3
4
5
6 int main(){
7     int m[MAX][MAX];
8     int v, k;
9     int i, j;
10    int max_broj_negativnih, max_indeks_kolone;
11    int broj_negativnih;
12
13    /* čUitavamo dimenzije matrice */
14    scanf("%d", &v);
15    scanf("%d", &k);
16
17    if(v<0 || v>MAX || k<0 || k>MAX){
18        fprintf(stderr, "-1\n");
19        return 0;
20    }
21
22    /* čUitavamo elemente matrice */
23    for(i=0; i<v; i++){
24        for(j=0; j<k; j++){
25            scanf("%d", &m[i][j]);
26        }
27    }
28
29    /*Pronalazimo kolonu koja žsadri čnajvei broj negativnih
    elemenata */
30    max_indeks_kolone=0;

```



```

max_broj_negativnih=0;
32 for(i=0; i<v; i++){
    if(m[i][0]<0){
34         max_broj_negativnih++;
    }
36 }
38
    for(j=0; j<k; j++){
        broj_negativnih=0;
        for(i=0; i<v; i++){
40             if(m[i][j]<0){
42                 broj_negativnih++;
44             }
            if(broj_negativnih>max_broj_negativnih){
46                 max_indeks_kolone=j;
            }
48         }
50     }

52 /* Ispisujemo žtraeni rezultat */
    printf("%d\n", max_indeks_kolone);
54
56 /* šZavravamo program */
    return 0;
}

```

#### Rešenje 5.4

```

1 #include <stdio.h>
  #include <stdlib.h>
3 #include <string.h>
  #define MAX 128
5
  int main(int argc, char **argv) {
7     FILE *f;
    int brojac = 0;
9     char linija[MAX], *p;

11     if (argc != 3) {
        fprintf(stderr, "-1\n");
13         exit(EXIT_FAILURE);
    }

15
    if ((f = fopen(argv[1], "r")) == NULL) {
17         fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
19     }

21     while (fgets(linija, MAX, f) != NULL) {
        p = linija;
23         while (1) {
            p = strstr(p, argv[2]);
25             if (p == NULL)

```

```
27     break;
    brojac++;
    p = p + strlen(argv[2]);
29 }
31 }
    fclose(f);
33 printf("%d\n", brojac);
35 return 0;
37 }
```

Rešenje [5.5](#)

Rešenje [5.6](#)